TECNOLOGÍAS DE LOS SISTEMAS DE INFORMACIÓN DE LA RED

Tema 6

Miquel Gómez Corral

Ing. Informática ETSINF, UPV

Tema 6

Escalabilidad

TEOREMA CAP

En un sistema distribuido **NO se puede mantener a la vez** Consistencia Fuerte, Disponibilidad y la posibilidad de que haya particiones. Una de las 3 cosas NO se puede. Según lo que nos haga falta hacemos una cosa u otra.

NO tener consistencia C (PA): **Es escalable**, *los nodos siguen teniendo trabajo y los usuarios son atendidos*. Cuando se produce la partición, si quiero que **TODOS los usuarios sigan accediendo al servicio**, al estar haciendo modificaciones en partes del sistema que no se pueden comunicar entre ellas (*xq la partición*) voy a dejar de tener consistencia. *Los cambios de unos nodos NO se propagan al resto.*

Esto hará que cada partición diverja, cuando se resuelva el problema habrá que juntar estados: CONSISTENCIA FINAL

NO tener disponibilidad A (PC): **Escala MAL**, *algunos nodos no tienen trabajo y hay usuarios sin atender*. Cuando se produce la partición, si NO quiero que los estados diverjan y que no haya problemas, haga que SOLO una de las particiones ofrezca el servicio. Eso hace que usuarios NO accedan, por lo que NO tiene tanta disponibilidad.

NO tener particiones P (AC): Si NO se quiere/puede renunciar a la consistencia o disponibilidad, habría que evitar que ocurrieran particiones. Pero eso es algo **imposible** **si el sistema es muy grande**. *Solo redes chikitas*.

REPLICACIÓN MULTI-MÁSTER

Los modelos básicos de replicación, pasiva y activa, NO ESCALAN BIEN. O xq el añadir más pasivos NO hace nada (xq el trabajo lo sigue haciendo 1) o xq al haber muchos activos, los retardos te joden y si falla alguno ya flipas.

La solución es **extender la replicación pasiva** al modelo Multi-Máster: Hacer que, **para cada subservicio**, **una de las replicas se la principal**. *Si hay diferentes actividades a la vez, cada copia se encarga de una y **distribuye al resto**.*

Perezoso: Todos los nodos, cuando hacen cambios los propagan al resto, no lo hace enseguida, es "Perezoso", propaga cada unos pocos cambios. Así genera **más rápido** la **respuesta** al cliente y no pierde tanto tiempo.

Cada nodo es **PRIMARIO** para algunas cosas y **SECUNDARIO** para otras.

Ventajas: **Altamente escalable**. **Mínima sobrecarga**, cada petición es procesada por una sola réplica. Admite **operaciones no deterministas** solo son ejecutadas por un máster. *MAS ESCALABLE PERO MENOS ROBUSTO (FALLOS)*

Inconvenientes: **Problemas en caso de fallos**, si cae un máster las peticiones se pueden perder. La gestión de fallos es igual que el modelo pasivo, un poco kk. Pueden surgir inconsistencias fácilmente.

ALMACENES NO SQL

Los **modelos relacionales** son simples y potentes. Son declarativos, por lo que le dices el QUÉ quieres hacer, pero NO el cómo. Esto soporta transacciones (muchas operaciones seguidas). Lo que pasa es que **NO es escalable**. Queremos algo que pueda recibir muchas actualizaciones y lecturas mientras es persistente.

Simplificar el esquema: **Sustituir múltiples tablas** con múltiples columnas por tablas clave/valor. No puedes buscar por los valores que no sean clave. Eso hace más simple el sistema. También ocupa menos: Cabe en MP.

Eliminar transacciones: **No se puede ejecutar "un grupo"** de operaciones, **solo individuales**. Esto crea atomicidad en las operaciones y evita bloqueos (*mientras yo hago 500 operaciones nadie toca, pues ahora NO*).

Almacene NoSQL: *Almacenes clave-valor*

Esquema compuesto por dos campos: clave y valor. Valor es toda la información de lo que me guardo aunque es difícil usarla. No puedes buscar por los valores que no sean clave.

Almacene NoSQL: *Almacenes de documentos*

Esquema compuesto por objetos con un número variable de atributos. En algunos casos estos atributos pueden ser, a su vez, objetos. **Aquí sí que se puede buscar por esos atributos**.

Almacene NoSQL: *Almacenes de registros extensibles.*

Esquemas formados por tablas que pueden tener un núm variable de columnas. **Separan datos por filas o columnas y estas separan en diferentes nodos**. Por lo que para acceder a ciertos datos te vas a uno u a otro: **sharding**.

ELASTICIDAD

Es elástico cuando además de ser escalable, es eficiente: Puede usar y dejar de usar los recursos cuando sea conveniente. Ser eficiente usando los menores recursos posibles y si a un servicio le hacen falta más nodos que a otro, pues le quito a uno y le pongo a otro.

Requiere: Un **sistema de monitorización** de la **carga** y el **rendimiento**, para ver donde hace faltan más y menos recursos. **También Un sistema de actuación** para **automatizar** la reconfiguración de los recursos y servicios.

CONTENCIÓN Y CUELLOS DE BOTELLA

No sirve de nada que todo esté super escalado si luego tengo algo que es muy lento. Aun que solo sea un componente del servicio, si me lo ralentiza todo me quedo igual. *A esto se le llama **Contención***.

Causas de la contención y solución

- **Cuando se centraliza todo**, y que pase por un nodo dándole **tareas "pesadas"** (*solo centralizar si el bróker hace cosas simples*). Solución: si las tareas son pesadas pues no centralizar xd.
- Usar **herramientas de sincronización** para evitar condiciones de carrera al compartir cosas. Solución: Usar asincronía.
- Cuando hay **tráfico excesivo**. Solución: **Replicar los recursos** y mantenerlos consistentes. Hacer que **accesos remotos** se transforman en accesos locales