Bases de Datos NoSQL: Consistencia

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.
Despacho: 209
Telf: 881816463
Mail: jrr.viqueira@usc.es
Skype: jrviqueira
URL: https://citius.gal/team/jose-ramon-rios-viqueira

Curso 2023/2024

Guion

- Introducción
- Consistencia en modificaciones
- Consistencia en lecturas
- Relajando la consistencia
- Relajando durabilidad
- Quorums
- Versiones

Introducción

- **Base de Datos relacionales** proporcionan alta consistencia (ACID)
  - Atomicidad: Se ejecutan todas las instrucciones o ninguna
    - Registro histórico
  - Consistencia: En escenario de uso aislado
    - Usuario garantiza la consistencia
    - SGBDs proporciona ayudas para la definición y verificación de restricciones
      - Claves primarias y forárenas, otras restricciones (check), disparadores, etc.
  - Aislamiento: En un escenario de varios usuarios en concurrencia
    - SGBDs garantiza la ejecución aislada
      - Ejemplo: **Aislamiento de instantáneas**
  - Durabilidad: Cuando una transacción termina, sus efectos perduran
    - Registro histórico + copias
- **Bases de datos NoSQL**
  - **¿Que clase de consistencia necesita la aplicación?**
    - Consistencia eventual. Teorema CAP

Consistencia en las modificaciones

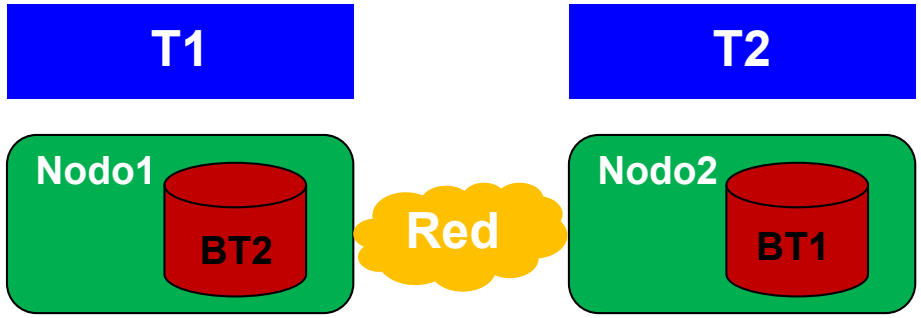
- Dos usuarios intentan modificar el mismo elemento en paralelo
  - Problema de la modificación perdida
  - Conflicto de tipo write-write

Efecto haber ejecutado T2 se pierde

T1	T2

leer(B);
 	
escribir(B);
	escribir(B);

- **Estrategias optimistas y pesimistas**
  - **Pesimistas**: Realizan acciones para que no se den casos de inconsistencias
    - Ejemplos: Protocolos basados en bloqueos y en marcas de tiempo
  - **Optimistas**: Permiten la ejecución normal de la transacción y actúan en el momento del compromiso solo si se detectan problemas
    - Ejemplos: Protocolos basados en validación y en versiones
- **Ejemplo con replicación peer-to-peer**
  - Dos réplicas pueden aplicar las mismas modificaciones en distinto orden



T1	T2

leer(B, nodo1);
 	
escribir(B, nodo1);
sincro(B, nodo2);
	escribir(B, nodo2);
	
	sincro(B, nodo1)

Consistencia en las modificaciones

- Control de concurrencia en distribuido
  - Consistencia secuencial
    - Aplicar las operaciones en el mismo orden en todos los nodos
- Forma optimista de gestionar conflictos de tipo write-write
  - Almacenar ambos cambios
  - Indicar que existe un conflicto
  - Intentar mezclar ambas versiones para obtener una versión consistente
    - Soluciones del ámbito del control de versiones
    - Soluciones muy específicas del ámbito de aplicación
- **Compromiso entre consistencia y eficiencia**
  - Aproximaciones pesimistas
    - generalmente poco eficientes
      - poco concurrentes
    - pueden tener interbloqueos
- Mayoría de modelos de distribución usan solo una copia para modificar
  - Simplifica las soluciones que eviten conflictos write-write
  - Excepción: **Replicación peer-to-peer**

Consistencia en las lecturas

- **Conflictos read-write**
  - Consistencia lógica
    - Asegurarse de que varios elementos de datos tienen consistencia cuando se tratan de forma conjunta
      - A + B en el ejemplo

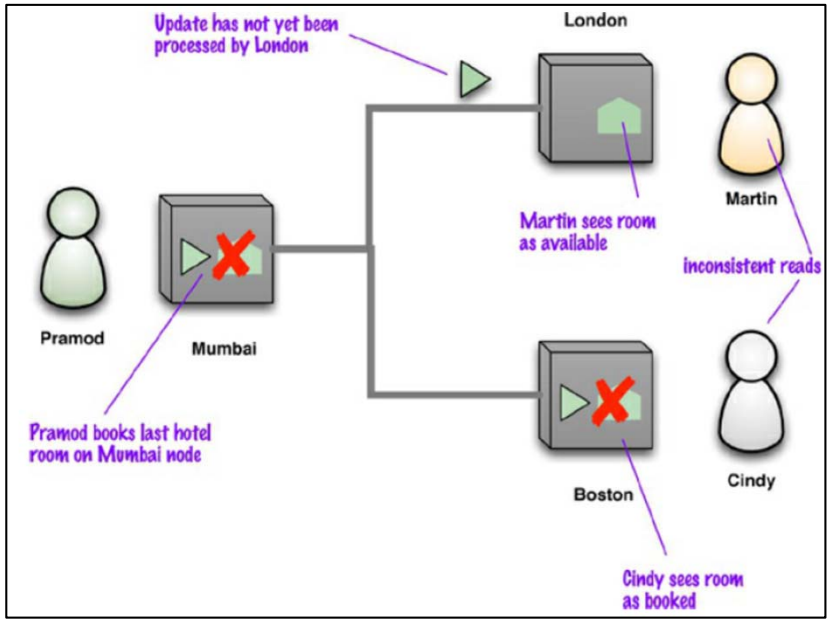
T1	T2

leer(A);
A:= A – 50;
escribir(A);
 	
leer(B);
B:=B+50;
escribir(B);
	leer(A)
	leer(B)
	visualizar(A+B)

- Las **Bases de datos de Grafos** suelen dar soporte para ACID
- **Bases de datos con agregados**
  - Modificaciones atómicas dentro del mismo agregado
  - Transacciones con varios agregados
    - Ventana de inconsistencia
      - Tiempo durante el cual se pueden producir lecturas inconsistentes
      - Suelen ser períodos muy cortos (menos de 1 segundo)

Consistencia en las lecturas

- Cuando se introduce replicación aparecen nuevos potenciales problemas de falta de consistencia
  - Regla general
    - **+ redundancia -> - consistencia**
- **Ejemplo**
  - Usuario reserva la última habitación de un hotel y se actualiza este hecho en una de las réplicas
  - Dos nuevos usuarios pueden ver dos estados distintos para la habitación, en función de que réplica consulten



- **Consistencia de replicación**
  - Asegurar que el mismo dato se lee igual desde todas las réplicas
  - Situación temporal.
    - Consistencia eventual
  - Problema similar en las memorias cache
- Ajustar el nivel de consistencia deseado a la aplicación
  - Mayoría de operaciones pueden realizarse con niveles bajos

Consistencia en las lecturas

- **Problemas de consistencia con el mismo usuario**
  - Ejemplo: Publicar comentarios en un Blog
    - Al refrescar la lista un comentario publicado anteriormente puede no estar
  - Consistencia de tipo read-your-write
  - Solución: Consistencia de sesión
    - Read-your-write durante la sesión de usuario
- **Implementación de la consistencia de sesión**
  - Sticky session: La sesión se vincula a un nodo
    - Al mantener read-your-write en el nodo se mantiene en la sesión
    - Problema: Reduce la capacidad del balanceador de carga
  - Version stamps: Uso de marcas de versiones
    - Asegurar que cada operación actúa sobre última versión
      - En caso contrario esperar para poder responder
  - Problema con Sticky session y replicación maestro-esclavo
    - Se puede leer de los esclavos, pero se modifica en el maestro siempre
    - Solución 1: Modificar en el esclavo y hacerlo responsable de actualizar el maestro
    - Solución 2: Mover la sesión al maestro mientras se realiza la modificación
      - La sesión vuelve al esclavo cuando se actualizan los cambios en el esclavo

Consistencia en las lecturas

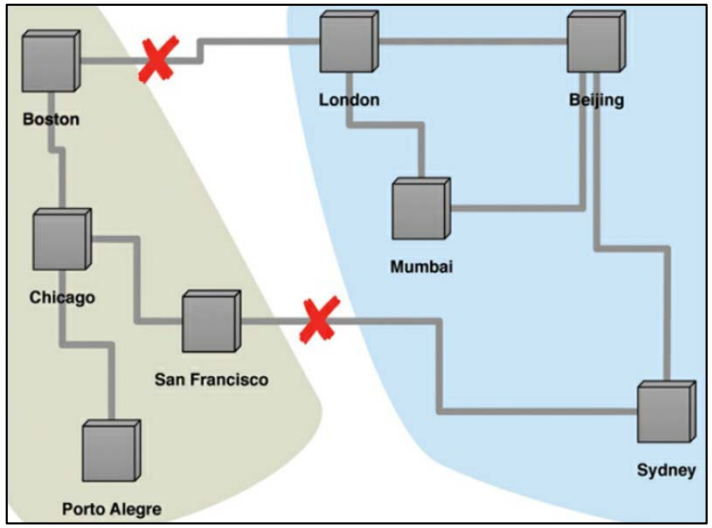
- **Consistencia en la interacción con el usuario**
  - Ejemplo: Reserva de la habitación del hotel
    - Dos intentos de reserva de forma concurrente
    - Cuando se pulsa el botón de reservar, el estado puede haber cambiado ya y no estar disponible
    - Solución: dividir la transacción en dos partes
      - Parte 1: Interacción con el usuario
      - Parte 2: Finalización de la transacción
        - Ejecución atómica en una transacción de BD
        - Verificación de la consistencia de los datos

Relajando la consistencia

- A veces es necesario sacrificar la consistencia
  - Para mejorar otras propiedades
    - Solución de compromiso
    - Dependiente de la aplicación
- También en sistemas centralizados
  - Recordar niveles de consistencia en SQL
    - Secuenciable: Ejecución secuencial (casi siempre...)
    - Lectura repetible: Lectura de datos comprometidos y lectura repetible.
    - Lectura comprometida: Lectura de datos comprometidos.
    - Lectura no comprometida: Lectura de datos no comprometidos.
- Muchos sistemas evitan el uso de transacciones completamente
  - MySQL muy popular cuando no tenía gestión de transacciones
    - Aplicaciones web
  - Sitios web muy grandes con necesidad de sharding (eBay)
  - Necesidad de interacción con sistemas remotos

Relajando la consistencia

- **Teorema CAP**
  - Propuesto por Eric Brewer en el 2000 (conjetura de Brewer). Prueba formal por Seth Gilber y Nancy Lynch en 2002.
  - **Teorema**
    - Un sistema solo puede tener dos de las tres siguientes características
      - Consistencia
      - Disponibilidad (Availability)
        - Significado aquí un poco distinto al habitual
        - Si podemos comunicar con un nodo del cluster, entonces podemos leer y escribir datos.
      - Tolerancia al particionamiento
        - El cluster puede sobrevivir a fallos de comunicación entre los nodos que lo separan en varias partes.

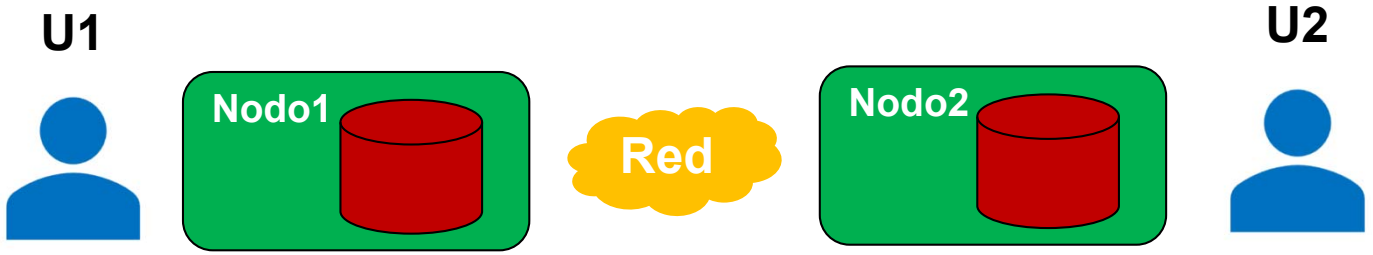


Relajando la consistencia

- **Teorema CAP**
  - Sistema centralizado
    - Consistencia y Disponibilidad (CA)
    - No tolerancia al particionamiento: una sola máquina no puede particionar.
    - Si el nodo está levantado, entonces está disponible
  - Cluster
    - Un cluster puede particionarse en dos pedazos no conectados
    - Es posible tener un cluster CA
      - Si hay una partición del cluster, debemos paralo por completo para que no pueda usarse
        - Definición habitual de disponibilidad significaría falta de disponibilidad
        - En el contexto de CAP disponibilidad significa: Cada petición recibida por un nodo que no falla debe ser respondida.
    - Detener todos los nodos cuando hay una partición es muy costoso
      - Incluso detectar las particiones a tiempo es un problema en si mismo

Relajando la consistencia

- **Teorema CAP**
  - **Como se interpreta el teorema en la práctica**
    - Si un sistema pueden sufrir particiones (sistema distribuido), hay un compromiso entre la consistencia y la disponibilidad.
    - **Ejemplo**
      - Dos intentos de reserva de la misma habitación concurrentes en sistema con replicación peer-to-peer
      - Nodo1 y Nodo2 necesitan comunicarse para serializar ambas peticiones
        - Incrementa la consistencia
        - Baja disponibilidad: Si la red falla, ninguno de los nodos puede responder
      - Mejorar la disponibilidad: Elegir un maestro y dirigir todas las modificaciones al maestro
      - Mejorar más la disponibilidad: Aceptar peticiones incluso si la red falla.
        - Perdemos consistencia: ambos podrían reservar la última habitación
        - Podríamos admitir esto: overbooking, mantener habitaciones libres para estos casos



Relajando la consistencia

- **Teorema CAP**
  - **Como se interpreta el teorema en la práctica**
    - Si un sistema pueden sufrir particiones (sistema distribuido), hay un compromiso entre la consistencia y la disponibilidad.
    - **Ejemplo: Carro de la compra (Amazon Dynamo)**
      - Siempre se permite escribir en el carro de la compra: incluso si falla la red.
      - Un proceso final puede unir todos los carros generados en uno
        - Si hay problemas el usuario toma el control para verificar los carros antes de realizar el pedido
  - **Conclusión**
    - Programadores buscan consistencia. Pero se puede relajar para conseguir disponibilidad y mejorar eficiencia
      - Normalmente se necesita conocimiento del dominio de aplicación
    - Consistencia de lectura
      - Importancia del dato para la aplicación: inversión bolsa vs noticia en blog
        - Tolerancia a las lecturas viejas
        - Duración de la ventana de inconsistencia

Relajando la consistencia

- **Teorema CAP**
  - **NoSQL: Propiedades BASE**
    - Propiedades (sin definición clara): Basically Available, Soft State, Eventual Consistency
    - Compromiso entre ACID y BASE (soluciones intermedias)
  - Compromiso entre **consistencia** y **latencia (más claro)**
    - Mejoramos consistencia introduciendo más nodos en la interacción
      - **Ejemplos**:
        - Si al leer, leemos de 5 nodos estaremos más seguros de la consistencia de la lectura que si leemos de solo 2
        - Si al escribir, no devolvemos el control hasta escribir en 5 nodos tendremos una escritura más consistente que si escribimos en solo 2 y confiamos en que el resto se escribirán de forma asíncrona más tarde.
      - Empeoramos eficiencia: Cada nuevo nodo incrementa el tiempo de respuesta
    - Disponibilidad como límite de la latencia que podemos tolerar en la aplicación
      - Esta idea relaciona disponibilidad con eficiencia
      - Cuando la latencia es muy alta, nos rendimos y asumimos que la operación no puede terminar.

Relajando la durabilidad

- **Atomicidad: clave para la consistencia**
  - Definición de unidades atómicas de ejecución (**Transacciones**)
- ¿Se puede relajar la durabilidad?
  - Perder las actualizaciones de algunas transacciones terminadas con éxito
  - Compromiso entre **durabilidad** y **rendimiento**
    - Pérdida de datos en memoria si hay un fallo en el sistema
    - Perder información de sesión en una web puede no ser muy importante
- Especificar en cada escritura si necesitamos durabilidad o no
  - Podríamos perder algunos datos insertados por un sensor
- **Durabilidad de replicación**
  - Ejemplo de problema por falta de durabilidad de replicación
    - Maestro falla. Esclavo busca otro maestro. Algunos datos del maestro caído no se habían replicado a un esclavo. Esclavo sigue recibiendo cambios a través del nuevo maestro. Maestro antiguo se recupera. Podemos encontrar conflictos
  - Esperar a tener las réplicas generadas para comprometer transacción mejora la durabilidad de replicación.
    - Empeora rendimiento de las escrituras
    - Disminuye la disponibilidad (aumenta la probabilidad de fallo de la escritura)

Quorums

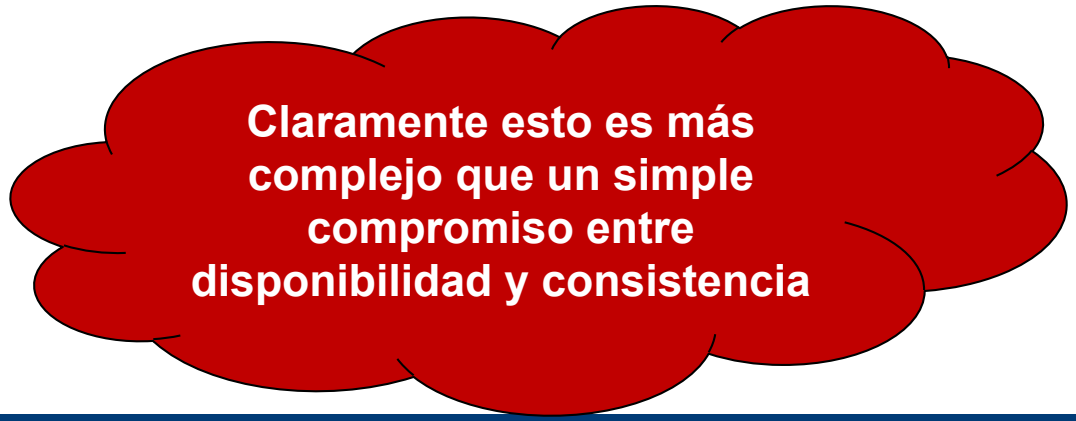
- Soluciones reales para la consistencia o durabilidad
  - Búsqueda del nivel deseado en la aplicación. **¿Cuantos nodos implicar?**
    - Si tenemos tres réplicas, ¿Cuántas debemos involucrar para obtener una respuesta?
      - ¿Llega con la mayoría?, dos en este caso
  - Si tenemos escrituras en conflicto, solo una puede haber realizado la mayoría de escrituras
  - **Quorum de escritura**
    - **W > N/2**
      - **W**: Nodos que participan en la escritura
      - **N**: Nodos totales involucrados en la replicación (factor de replicación)
  - **Quorum de lectura**
    - ¿Cuántos nodos tenemos que consultar para asegurar que tenemos la copia más reciente?
      - N = 3, W = 2. Necesitamos leer dos nodos para asegurar que leemos el último dato. R = 2.
      - N = 3, W = 1. Necesitamos leer tres nodos. R = 3.
    - **R + W > N**

Asumiendo Replicación peer-to-peer

Maestro-esclavo llegaría con interactura con el maestro

Quorums

- No confundir el factor de replicación con el número de nodos del cluster
  - Podemos estar realizando 3 copias de cada dato en un cluster de 12 nodos
- Factor de replicación 3 es muy común
  - Permite un fallo de un nodo mientras no se realiza una nueva réplica
- Ajustar W y R a las necesidades de la aplicación
  - Lecturas consistentes y rápidas
    - W tiene que ser alto, para que R sea bajo
    - Las escrituras serán lentas
  - Escrituras rápidas y consistencia baja
    - W tiene que ser bajo.
    - Para que R sea bajo también tenemos que usar un N bajo
      - La consistencia será baja por lo tanto



Versiones

- BD NoSQL suelen proporcionar atomicidad dentro del agregado
- Transacciones del sistema no funcionan bien con la interacción con el usuario
  - Uso de marcas de versiones (comprobar si la versión leída ha cambiado)
- **Transacciones de negocio y de sistema**
  - Ya hemos visto la diferencia
  - Transacción de negocio dividida en dos partes
    - Parte I: Interacción con el usuario
    - Parte II: Transacción de sistema (bloqueo de recursos durante poco tiempo)
  - Técnicas de concurrencia offline: Ejemplo Optimistic Offline Lock
    - Leer de nuevo todo para comprobar si ha cambiado
    - Utilizar una marca de versión para comprobarlo
    - Implementación de las marcas
      - Contadores
      - secuencias aleatorias
      - Hashing de los elementos
      - Marcas de tiempo
    - Utilización para proporcionar consistencia de sesión

CouchDB combina un Contador con un Hash del elemento

Versiones

- **Marcas de versión en múltiples nodos**
  - Marcas de versión funcionan bien en un nodo o en maestro-esclavo
    - Solo un nodo controla las versiones
  - En peer-to-peer dos nodos pueden tener valores distintos por distintas razones
    - Actualización llegó a uno pero no al otro
      - Elegir la más reciente
    - Puede haber otro tipo de inconsistencia en la modificación (necesita solución)
  - Forma más simple de implementación: **Contador**
  - Si tenemos más de un maestro, se necesita un solución más avanzada
    - Todos los nodos tengan historial de versiones
    - Cliente mantiene versiones / servidor responde con versiones
    - No se usa en NoSQL. Típico en sistemas de control de versiones
  - **Uso de timestamps**
    - Problemático: necesita consistencia temporal entre nodos
    - Si las modificaciones se hacen con mucha rapidez puede haber problemas

Versiones

- **Marcas de versión en múltiples nodos**
  - **Vector Stamp** (solución muy usada)
    - Un contador por nodo en un vector
      - (3, 5, 6)
      - Si el segundo nodo modifica el dato, el vector cambia a (3, 6, 6)
    - Cuando dos nodos se comunican se sincronizan los vectores
      - Diferentes formas de sincronización
    - Un vector será más reciente que otro, si todos los contadores son mayores o iguales
    - Si ambos vectores tienen un valor mayor que el otro entonces existe un conflicto de tipo write-write
    - Si un valor de un vector falta se trata como cero
      - Permite añadir nuevos nodos
    - Muestra las inconsistencias, pero no las resuelve
    - La resolución de inconsistencias depende bastante del dominio de aplicación
      - Compromiso consistencia / latencia
        - Asumir que los fallos de red harán que el sistema falle
        - O por lo contrario detectar y tratar las inconsistencias

Bases de Datos NoSQL: Consistencia

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.
Despacho: 209
Telf: 881816463
Mail: jrr.viqueira@usc.es
Skype: jrviqueira
URL: https://citius.gal/team/jose-ramon-rios-viqueira

Curso 2023/2024