Universidad Nacional de Rosario

Trabajo Final: Monografía
Bases de Datos Avanzadas

Bases de Datos en Tiempo Real

Autor:
Agustín Gurvich

Docentes:
Claudia Susana Deco,
Cristina Marta Bender

Departamento de Ciencias de la Computación
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Av. Pellegrini 250, Rosario, Santa Fe, Argentina

16 de febrero de 2023

Índice general

Resumen

Este trabajo fue desarrollado en el marco de la materia "Bases de Datos Avanzadas", con el objetivo de brindar un cierre al proceso de investigación desempeñado durante el cuatrimestre. El objetivo de esta monografía es recopilar la información expuesta en las clases de la materia.
A lo largo de este texto se verá una breve introducción a los conceptos claves que integran a las bases de datos en tiempo real, así como aplicaciones de las mismas, trabajos realizados bajo este marco y posibles líneas de investigación a futuro.

Todo list

Capítulo 1

Introducción

1.1. Bases de Datos en Tiempo Real

En el último tiempo, la demanda por sistemas que reaccionen a los flujos constantes de información se volvió cada vez más grande. La creciente cantidad de información disponible, poder de cómputo y complejización de los procesos dieron como resultado una necesidad imperiosa de adaptar los sistemas informáticos a un ritmo mucho más dinámico. En este sentido, las bases de datos tradicionales comienzan a flaquear dado su planteamiento original.
Por un lado, se pueden recordar las ventajas que otorgan las bases de datos tradicionales:

- Descripción de los datos: otorgan una forma sencilla y concisa de representar información
- Correctitud e integridad: permiten mantener estas cualidades de los datos
- Acceso eficiente: es posible acceder a la información de forma eficiente y veloz
- Propiedades ACID: las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) otorgan muchas garantías sobre los datos

Si bien esto puede parecer suficiente, es posible encontrar ciertos problemas:

- Información volátil: la información en tiempo real suele ser volátil, es decir, luego de cierto período de tiempo deja de ser útil. Como las bases de datos tradicionales están pensadas para almacenar datos duraderos, se puede pensar que necesitamos nuevos mecanismos de almacenamiento. A partir de este punto se infiere que es importante tener en cuenta la marca temporal de los datos, es decir, cuándo fueron medidos
- Puntualidad antes que correctitud: en los sistemas en tiempo real es más importante cumplir con las restricciones de tiempo que dar información

> absolutamente correcta, cosa que las bases de datos tradicionales no pueden brindar. En base a esto, es correcto afirmar que la atomicidad de las transacciones en tiempo real no es tan importante

Por estos problemas, se hace evidente que las bases de datos tradicionales necesitan una mejora. Por esta necesidad es que se decidió tomar inspiración en los sistemas de tiempo real. Según [16]:

> Un sistema de tiempo real es un sistema informático que interacciona con su entorno físico y responde a los estímulos del entorno dentro de un plazo de tiempo determinado.

A lo anteriormente dicho se añade que, además de estos requerimientos, es importante que las tareas se ejecuten dentro de un intervalo de tiempo determinado.

Aún no existe una definición formal de bases de datos en tiempo real (de ahora en más BDTR), pero se puede pensar en ellas como una base de datos que, además de cumplir restricciones sobre la información que almacena, debe cumplir las transacciones dentro de un rango de tiempo estipulado.

Una definición interesante es la propuesta por Stankovic y Hansson en [18]:

> Un sistema de base de datos en tiempo real es un sistema de base de datos donde las transacciones llevan asociadas restricciones de tiempo y los datos tienen intervalos de tiempo específicos para los cuales son válidos.
>
> Las restricciones de tiempo de las transacción pueden ser deadlines para completar, tiempos por las que deben comenzar, invocaciones periódicas, deadlines, etc.
>
> No es necesario que cada transacción tenga una restricción de tiempo, solo que algunas transacciones las tienen. Además de los requisitos de tiempo, las transacciones tienen semántica temporal involucrada.

Se pueden destacar algunos conceptos que se desarrollarán a lo largo de este trabajo:

- Intervalos de validez
- Deadlines
- Mezcla de transacciones
- Semántica de tiempo (tiempo de validez de los datos)

1.2. Conceptos equivocados de BDTR

Antes de entrar en detalle sobre la teoría que integra las BDTR, queremos destacar algunas ideas equivocadas que suele generar el concepto de BDTR según lo visto en [18].

Una BDTR no implica velocidad. Que contenga la frase tiempo real en su nombre, no implica que efectivamente tenga que ser instantánea. Simplemente significa que tiene especial atención a los eventos del entorno. Los avances en tecnología de procesamiento y almacenamiento no implican que se cumplan los requerimientos, pero sí ayudan enormemente.

Por otro lado, se puede pensar que los problemas se solucionan llevando toda la base de datos a memoria principal. Esto no es suficiente, ya que existen otras fuentes de retrasos (locks, scheduler, procesamiento distribuido) que no pueden ser solucionados por esta estrategia.

Finalmente, existe cierta confusión entre BDTR y bases de datos temporales. Si bien ambas trabajan con el concepto de tiempo, las primeras se ven afectadas teniendo que generar restricciones sobre la información mientras que las segundas se encargan de soportar el tiempo asociado a la información (validez y transacción). Por ejemplo, si un sensor toma datos cada T unidades de tiempo y esos datos tienen un tiempo de vida t, entonces se deberá volver a sensar dentro de $min(T, t)$ unidades de tiempo para que la información sea relevante en una BDTR.

Capítulo 2

Marco teórico

A lo largo de este capítulo se desarrollan algunos conceptos para lograr comprender las bases de las BDTR. Cabe resaltar que este área de estudio es muy extensa, y en este trabajo se contemplan solo algunas de las ideas necesarias para el entendimiento del tema.

2.1. Consistencia

Las BDTR imponen nuevas restricciones. Un dato ya no debe ser solo lógicamente consistente (satisfaciendo las restricciones de integridad de la información) sino que también debe serlo temporalmente (satisfaciendo las restricciones de tiempo). La consistencia temporal consiste de dos componentes: la consistencia absoluta (el estado de los datos en la BDTR deben estar actualizados con respecto al estado del entorno) y la consistencia relativa (el dato debe ser considerado temporalmente válido). Siguiendo esta idea, Ramamritham propone formalizar la consistencia [16]:
Consideremos un ítem en la base de datos como una tupla $d : (valor, avi, timestamp)$ donde:

- d_{value}: el estado actual del dato d
- d_{avi}: el intervalo de validez absoluta, es decir, por cuánto tiempo es válido el dato luego de que ingresa en la BDTR
- $d_{timestamp}$: el momento en que ingresó el dato

Definimos ahora un conjunto de consistencia relativa R como el conjunto de datos que tienen una validez relativa R_{rvi}. Ya nos encontramos en condiciones de definir formalmente la consistencia.
Sea $d \in R$ un dato en la base de datos. d es consistente si:

1. d_{value} es lógicamente consistente
2. d es temporalmente consistente:

a) Consistencia absoluta: $(tiempoActual - d_{timestamp}) \leq d_{avi}$

b) Consistencia relativa: $\forall d' \in R | d_{timestamp} - d'_{timestamp} | \leq R_{rvi}$

Analizando esta definición es posible entender su significado. La consistencia lógica está provista por los esquemas tradicionales de bases de datos, sobre las cuales construimos las BDTR. La consistencia temporal requiere cumplir con dos condiciones. La consistencia absoluta de d implica que el dato sigue siendo válido (que no haya expirado). La consistencia relativa de d implica que si hay otro dato dentro del conjunto consistente R, entonces ingresó dentro de un tiempo prudencial a la base de datos con respecto al resto de los datos con los que se relaciona. Para comprender mejor estos conceptos, veamos algunos ejemplos.

Supongamos que nuestra base de datos se emplea para una cinta transportadora en una fábrica, y que el objetivo es catalogar los objetos que pasan delante de una cámara. Cada objeto pasa 5 segundos dentro del campo de visión de la cámara hasta que aparece uno nuevo. En este caso podemos pensar que $R = \{Objeto\}, Objeto_{avi} = 5, R_{rvi} = 5$. Esto conlleva tres consecuencias: solo clasificaremos el objeto que pasa frente a la cámara, el avi de los datos es de 5 segundos (pues cada 5 segundos cambia el objeto en el campo de visión) y el rvi del conjunto también es de 5 segundos (pues se compone de un solo objeto).

Para considerar un caso más complejo, podemos en una base de datos que se utiliza para guardar mediciones del clima en un campo. La temperatura debe muestrearse a lo sumo cada cinco minutos , y cada diez minutos la presión atmosférica. Es posible tomar mediciones en intervalos de tiempo menor. El intervalo de validez relativa del conjunto es de dos minutos. De esta forma tenemos $R = \{t, p\}, t_{avi} = 5, p_{avi} = 10, R_{rvi} = 2$.

Supongamos que $tiempoActual = 85$ (pasaron 85 minutos desde el inicio del sistema), y sean dos mediciones $t = (30, 5, 81)$ y $p = (10007, 10, 84)$. Estos datos son absolutamente consistentes (transcurrió un intervalo prudencial desde que se realizó la muestra) pero no son relativamente consistentes pues no cumplen con las restricciones de validez relativa.

El ejemplo anterior pone en evidencia dos hechos. El primero es que mantener la consistencia absoluta es relativamente sencillo, basta con ingresar datos frescos a la base de datos de manera frecuente (lo cual se puede complejizar dependiendo el proceso de obtención de los mismos). El segundo muestra la complejidad de mantener la consistencia relativa pues requiere que los datos ingresados se observen en intervalos cercanos entre ellos. Ramamritham explica que la consistencia relativa solo es importante cuando los datos se utilizan entre ellos para derivar información adicional.

2.2. Aplicaciones

En esta sección se presentan tres requerimientos reales presentados en [13]. En dicho análisis de requerimientos se consideran las siguientes métricas: tamaño de la base de datos, tiempo de respuesta para lectura y escritura, consistencia absoluta, consistencia relativa y durabilidad de la base de datos (cuánto tiempo debe tolerar el sistema antes de realizar un reinicio por problemas de rendimiento). Los dominios analizados son aviones, naves espaciales e industria.

Database size	3,000 entities
Read/write response time	0.05 ms. minimum 1.00 ms. maximum
External consistency	0.05 sec.
Temporal consistency	0.20 sec.
Durability	4 hours

Table 3.1 Aircraft Mission Computer Data Characteristics

Database size	5,000 entities
Read/write response time	0.05 ms. minimum 1.00 ms. maximum
External consistency	0.20 sec.
Temporal consistency	1.00 sec.
Durability	25 years

Table 3.2 Spacecraft Onboard Computer Data Characteristics

Database size	100,000 entities
Read/write response time	1.00 ms. minimum 10.00 ms. maximum
External consistency	0.20 sec.
Temporal consistency	0.50 sec.
Durability	5 days

Table 3.4 Industrial Control Data Characteristics

Figura 2.1: Datos relevados de diversos dominios

Como se puede observar en la Figura 3.1, los requerimientos tienen distintos niveles de complejidad. Por ejemplo, la consistencia relativa en los tres casos. Para los aviones, es muy pequeña (0.2 segundos) debido a las condiciones volátiles y demandantes del ambiente. En el caso de las naves espaciales es de 1 segundo, pues en el espacio hay mucho más margen para corregir errores y es factible utilizar datos expirados ya que no se deberían registrar muchos cambios en el entorno. Finalmente para la industria es de 0.5 segundos, lo cual puede considerarse razonable siempre y cuando el proceso no sea extremadamente crítico.

2.3. Deadlines

Una deadline es un límite de tiempo en el cual debe cumplirse la transacción. Se corresponden con situaciones de la vida real y pueden ser representadas con funciones de valor. Una herramienta usual de los algoritmos de eliminación

de concurrencia es abortar transacciones problemáticas, pero esto va en contra de las necesidades de tiempo real. Abortar una transacción puede hacer que se pierda la deadline y generar graves problemas.

Las deadlines se clasifican en las siguientes categorías:

- Deadline dura: es una transacción cuyo incumplimiento tiene consecuencias serias (inclusive catastróficas). Son críticas para la seguridad del sistema, y son la razón por la que se evalúa perder las certezas de las bases de datos convencionales.
- Deadline blanda: es una transacción cuyo cumplimiento es deseado, pero no indispensable. En general no suele ser un problema su incumplimiento, sino que se puede llegar a ganar valor de la transacción aún al demorarse en resolver.
- Deadline firme: es un tipo especial de deadline blanda, cuyo incumplimiento no agrega ni remueve valor al sistema. Es deseable que sean resueltas a tiempo, pero no es un inconveniente si no se logra.

Las funciones de valor utilizadas para representar las deadlines son las siguientes:

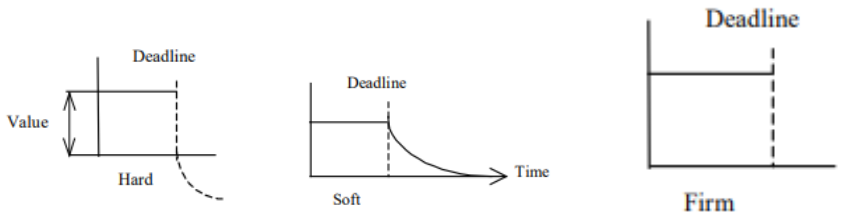

Figura 2.2: Representación de deadlines con funciones de valor

En [1], Aldarmi explica que cuando se genera una situación de sobrecarga de tareas en la base de datos el incumplimiento de las deadlines blandas aumenta exponencialmente. En cambio, el incumplimiento de deadlines firmes solo aumenta polinomialmente. Esto puede atribuirse a las decisiones tomadas por el scheduler a la hora de afrontar un aumento de la demanda.

2.4. Tareas en el sistema

Al trabajar con sistemas en tiempo real es muy importante determinar la frecuencia de los eventos. Si bien no suele ser factible prever todas las situaciones posibles, es posible modelar con cierta estimación las acciones a realizar. De esta forma se pueden clasificar las tareas en tres tipos. Por un lado se

presentan las tareas periódicas, que son aquellas que el sistema realiza constantemente. Son tareas en las cuales su periodicidad y tiempo de ejecución son conocidos (siempre estimando el peor caso), y están asociadas a deadlines duras. Luego están las tareas aperiódicas, aquellas cuyo tiempo de ejecución no puede ser estimado y suelen ser activadas en respuesta a eventos externos. Están asociadas con deadlines blandas. Finalmente se tienen las tareas esporádicas, tareas aperiódicas que tienen deadlines duras y representan situaciones de emergencia en el sistema.

Es importante tener en cuenta la cantidad de operaciones de cada tipo a la hora de elegir un scheduler adecuado para la BDTR.

2.5. Scheduler

Durante esta sección se desarrolla en profundidad el scheduler en una BDTR. Uno de los puntos críticos para cumplir con las deadlines es la elección de un scheduler adecuado. En [1] se define al scheduler como un algoritmo o política para el ordenamiento de las tareas en un procesador de acuerdo a algún criterio predefinido. Para una elección óptima es necesario estimar el tiempo de ejecución de las tareas en el sistema utilizando distintas técnicas (deterministas, estimación de peor caso, probabilísticas). Un scheduler debe saber si una transacción es factible (si es posible de resolverla antes de su deadline) y se dice que es óptimo si puede programar todas las tareas que otros schedulers puedan. Es importante tener en cuenta que esto no involucra que se deban poder programar todas las tareas, pues algunas son de por sí imposibles de resolver a tiempo.

Se dice que un scheduler es apropiativo si tiene la capacidad de cancelar otras tareas, y no apropiativo si una vez iniciada la tarea esta no puede cancelarse. Existe una tercera clase de scheluders, los híbridos, que permiten la apropiación en ciertos puntos de la ejecución de la tarea. En un sistema de tiempo real, los schedulers apropiativos son los más adecuados ya que permiten tener un control mucho más fuerte sobre los recursos.

Se mide la complejidad computacional de un scheduler considerando dos factores: computabilidad (¿Es posible que este conjunto de transacciones se resuelva en el tiempo propuesto?) y decidibilidad (¿Es posible crear un programa que sea factible para ejecutar estas transacciones?). El primer problema puede ser resuelto en tiempo polinomial, pero el segundo es un problema NP-Completo.

A continuación se presentan distintos algoritmos de scheduling propuestos en [1] y se describen sus comportamientos.

2.5.1. Priority-Based scheduling

En los sistemas de tiempo real, las tareas deben ser planeadas de acuerdo a su criticidad. Por esto es que los mejores algoritmos de planeamiento son aquellos que lo toman en cuenta. El objetivo de los schedulers basados en prioridad es darle tratamiento preferencial a las tareas críticas por sobre las menos importantes. De esta forma, se aseguran que el sistema no le dé importancia a tareas menores.

2.5.2. Rate-Monotomic Policy

Rate-Monotomic Policy es una política apropiativa que asigna prioridades de acuerdo a la periodicidad de la transacción. Mientras más corto sea el período de la transacción, más prioridad se le asigna. Es un scheduler con prioridades estáticas, lo que lo hace muy sencillo de implementar y es óptimo en sistemas con tareas periódicas.

2.5.3. Most-Critical-First Policy

La política MCF divide todo el universo posible de tareas en niveles de prioridad, basado en su importancia en el sistema. De esta forma es posible generar planeamientos confiables que intenten ejecutar las tareas más críticas sin importar la carga del sistema. Como contra parte, agregar o modificar una tarea en el sistema es muy complicado pues debemos revisar todas las ya definidas y actualizar sus prioridades de ser necesario.

2.5.4. Earliest-Deadline-First Policy

EDF es una política apropiativa que utiliza las deadlines de las tareas como referencia de prioridad. De esta forma, las tareas que tengan una deadline más cercana se les asigna mayor prioridad. De esta forma podemos obtener la siguiente propiedad:
La política Earliest-Deadline-First es factible sii $\frac{C_1}{T_1} + \frac{C_2}{T_2} + ... + \frac{C_n}{T_n} \leq 1$
Donde C_i y T_i son el tiempo de computación y el período de la tarea i

Un problema importante de esta política es que bajo condiciones de sobrecarga de tareas le asigna una alta prioridad a aquellas que estén a punto de perder su deadline o que ya la hayan perdido. Una mejora posible a este esquema es tomar en cuenta la deadline cercana más factible. Una deadline se dice factible si el tiempo que le resta a su ejecución es menor que el tiempo que falta para incumplirla.

2.5.5. Value-Density Policy

En una sección anterior se presentó a las funciones de valor como una forma de interpretar la importancia de cumplir las deadlines. Utilizando esta idea es posible definir la política Value-Density. Una función VD se define como $\frac{Valor}{Cmputo}$, y hace que sea preferible darle prioridad a las tareas que otorgan mayor densidad de valor al sistema generando más valor por unidad de tiempo. Este es un algoritmo voraz que consume el primer valor alto que encuentre, por lo que puede dejar pasar oportunidades para ejecutar otras tareas y luego maximizar la densidad previamente obtenida.

2.5.6. Criticalness-Deadline-First Policy

Si bien en las técnicas presentadas anteriormente se intentó trabajar sobre la criticidad y las deadlines de forma separada, algunos autores plantean que ese tipo de algoritmos no son adecuados. Por esto se realizaron varios intentos para combinar ambas nociones en un esquema mucho más efectivo.

Uno de los primeros intentos propuestos por Biayabani en [4] fueron los algoritmos ALG1 y ALG2 que se comportan de la siguiente manera: primero intentan planear el orden de las tareas según sus deadlines, ignorando la criticidad. Si es factible, entonces el planeamiento es exitoso. Caso contrario, se intenta colocar la nueva tarea en el planeamiento eliminando tareas menos críticas. La diferencia entre ambos algoritmos radica en que ALG1 elimina tareas considerando solo la criticidad (de baja a alta) mientras que ALG2 tiene en cuenta además la lejanía de su deadline. Podemos observar que ambos algoritmos se comportan como EDF bajo condiciones normales, y en sobrecarga se comportan como MCF o una combinación entre EDF y MCF respectivamente.

Por otro lado, en [10] Huang propuso la política Criticalness-Deadline-First. En ella las prioridades se asignan según la fórmula $\frac{deadline-llegada}{criticidad}$. De esta forma es posible obtener una mejora en el rendimiento por sobre técnicas que cuentan ambos factores por separado y demostró ser efectivo para ordenar tareas críticas perdiendo la menor cantidad de tareas poco críticas.

2.6. Sobrecarga de tareas

El problema de la sobrecarga de tareas está íntimamente relacionado con el scheduler. En condiciones normales, un planeamiento del scheduler podrá ordenar cada tarea de manera tal que cumpla su deadline (siempre y cuando esto sea factible). Cunado no es posible, se dice que el sistema está sobrecargado. Algunas causas de la sobrecarga de tareas pueden ser: una serie de situaciones de emergencia, errores al estimar la complejidad de los casos, fallas

en el hardware, etc.

Para enfrentar esto, es posible optar por dos actitudes. Por un lado, rechazar las tareas entrantes. A esto se le llama enfoque garantizado porque efectivamente detiene la sobrecarga a costo de no ejecutar nuevas tareas. No es un enfoque que debería usarse a menudo, pues se corre el riesgo de perder información. Como segunda opción se pueden remover las tareas de menor valor hasta que se regularice el sistema, lo que se llama enfoque robusto. De esta forma solo se eliminan aquellas tareas que es sabido que no van a ser críticas para el sistema, y es posible mejorarlo implementando una cola de removidos que almacene dichas tareas y las ejecute si siguen siendo válidas al normalizarse el sistema.

Para cerrar esta sección, queremos hacer hincapié en un concepto importante. Cuando el sistema está sobrecargado es imposible cumplir con todas las tareas. Por lo tanto, muchos investigadores llegaron a la conclusión de que se deberían programar al menos las tareas más importantes. Lo que no está cubierto por esta idea es la definición de tarea importante. Al no existir una definición formal, esto resulta ser un problema puramente objetivo.

2.7. Concurrencia

La concurrencia es un componente natural en las bases de datos, y más aún en los sistemas distribuidos. El trabajo del control de concurrencia es que las transacciones sean serializables. Si bien en las bases de datos tradicionales las propiedades ACID se encargan de asegurar esto, las BDTR deben sacrificar estas a cambio de mayores velocidades de procesamiento. Por esto fue necesario desarrollar nuevas estrategias para el control de concurrencia.

2.7.1. Problemas de concurrencia

En esta sección se desarrollaran brevemente dos problemas de concurrencia muy importantes.

En primer lugar, se presenta el problema de la inversión de prioridades. Supongamos dos transacciones T_i, T_j con prioridades k, k' tales que $k < k'$. Si T_i tiene en su poder un recurso que T_j necesita para poder ejecutar (un lock de escritura por ejemplo), es muy probable que T_i no se ejecute a menudo por tener menor prioridad. Pero en este caso, T_j no podrá avanzar pues no tiene en su poder el recurso necesario. Una forma posible para resolver este problema es modificar T_i de manera tal que momentáneamente tenga una prioridad más adecuada para que complete su ejecución y libere el recurso, técnica conocida como heredamiento de prioridades.

Por otro lado, existe el problema de los deadlocks. Estos son muy problemáticos ya que no siempre es posible generar algoritmos para evitarlos. La metodología usual dentro de las bases de datos consiste en utilizar algoritmos para detectarlos y luego aplicar una estrategia de resolución de deadlocks. Algunas de ellas son:

1. Abortar la transacción que genera el deadlock
2. Hallar las transacciones involucradas en el deadlock y eliminar la primer transacción que haya fallado su deadline. Si ninguna cumple esa característica, abortar la que tenga la deadline más lejana pues podría volver a iniciarse en el tiempo restante
3. Hallar las transacciones involucradas en el deadlock y eliminar la primer transacción que haya fallado su deadline. Si ninguna cumple esa característica, abortar la que tenga la deadline más cercana pues es probable que no pueda completarse
4. Hallar las transacciones involucradas en el deadlock y eliminar la primer transacción que haya fallado su deadline. Si ninguna cumple esa característica, abortar la que tenga menor criticidad
5. Abortar la transacción no factible con menor criticidad. Si todas son factibles, abortar la de menor criticidad

2.7.2. Control de concurrencia

Para mantener la consistencia de los datos cuando se están ejecutando múltiples transacciones es necesario introducir alguna forma de control de concurrencia. De manera general es posible diferenciar dos tipos de estrategias importantes. Por un lado tenemos las estrategias pesimistas, en las cuales debemos obtener permiso antes de realizar operaciones sobre la información. Estas suelen ser más consistentes, pero con ejecuciones más lentas. Por el otro lado existen las estrategias optimistas. En este estilo de control de concurrencia no hay que pedir permisos, sino que se ejecuta toda la transacción y se valida al momento de realizar el commit en la base de datos para revisar si mantiene la serialización.

En las próximas secciones se discuten distintas estrategias de control de concurrencia planteadas en [1].

2.7.3. Locking

Las estrategias de locking son las más utilizadas en las bases de datos convencionales. La idea detrás de este concepto es bloquear el acceso a un recurso para que no más de un actor pueda utilizarlo. Es importante tener en cuenta

que el uso indiscriminado de locks no asegura el mantenimiento de la consistencia. A continuación se desarrollan algunas estrategias de locking.

Two-Phase-Locking (2PL) es una técnica de locking pesimista que garantiza la serialización de las transacciones controlando los momentos en los que se bloquean y se desbloquean los recursos. La transacción se divide en dos fases. Durante la primera fase, la fase de crecimiento, una transacción adquiere todos los locks que sean necesarios pero no puede liberar ninguno. Luego ingresa en la segunda fase, la fase de decrecimiento, donde comienza a liberar todos los locks adquiridos y no puede adquirir ninguno nuevo. Una transacción se bloqueará hasta que no obtenga el permiso (lock) correspondiente sobre el recurso. Si bien esta técnica es adecuada para las bases de datos convencionales, las restricciones de tiempo de los sistemas en tiempo real hacen que las estrategias de locking no sean eficientes debido a los largos tiempos de espera impuestos sobre los recursos.

2PL Wait Promote (2PL-WP) es un protocolo con un funcionamiento idéntico a 2PL, pero agrega un mecanismo de heredamiento de prioridades. Al detectar que una transacción bloquea a otra de mayor prioridad por acaparar un recurso, la primera obtiene la prioridad de la segunda. De esta forma la transacción se ejecutará y eventualmente liberará el recurso. Si bien los tiempos de bloque siguen siendo inciertos, este esquema mantiene la granularidad de los recursos planteada por 2PL y reduce los tiempos de espera por uso de locks. De todas formas, es posible que el sistema se degrade a una base de datos convencional si todas las transacciones ejecutan con la misma prioridad.

2PL High-Priority (2PL-HP) es otro tipo de modificación de 2PL que tiene en cuenta la prioridad de la transacción para resolver conflictos, haciéndolo siempre a favor de la que posea mayor prioridad. Si la transacción T_i intenta adquirir un recurso actualmente utilizado por T_j se comparan las prioridades. Si la prioridad de T_j es menor entonces T_j es abortada o reiniciada, caso contrario T_i esperará a que se libere el recurso. Una ventaja muy importante de 2PL-HP es que no presenta problemas de inversión de prioridades o deadlocks, pero es capaz de abortar transacciones en favor de otras que podrían no cumplir su deadline.

La última de las técnicas de locking propuestas es Conditional Priority Inheritance (CPI). La idea de este esquema es la siguiente: al detectar inversión de prioridades puede suceder que la transacción de menor nivel está a punto de ser completada, o puede tener que ejecutar muchos más pasos antes de terminar. Si sucede lo primero, entonces hereda la prioridad de la transacción de mayor nivel. Si sucede lo segundo, la transacción es abortada. El aspecto más complicado de este esquema es definir el concepto de transacción a punto de completar. Una estrategia posible (planteada por los autores de CPI en

[11]) es utilizar la cantidad de pasos restantes. El problema que surge de este enfoque es no tener en cuenta la semántica de los pasos, que podrían demorar demasiado y hacer que la transacción no cumpla su deadline. Para lograr una correcta implementación de CPI es necesario tener un perfecto entendimiento de los tiempos de ejecución de los procesos en el CPU y considerar límites dinámicos para la definición de una transacción a punto de terminar.

2.7.4. Control de Concurrencia Optimista

El control de concurrencia optimista (OCC) se basa en el concepto de validación de las transacciones al final de su ejecución. La idea de OCC es realizar todos los chequeos al mismo tiempo (al terminar la ejecución) para minimizar los retrasos por bloqueos u otros problemas de validación. Si las transacciones no interfieren entre sí los problemas son mínimos, pero es inevitable estos que aumenten a medida que el sistema crece. Como todas las escrituras en la base de datos se realizan al momento del commit, es equivalente a realizar una escritura atómica y como consecuencia tenemos un orden de serialización definido.

En el OCC clásico las transacciones realizan operaciones de lectura y escritura libremente, guardando las operaciones en un espacio de memoria privado. Antes de realizar un commit la transacción debe superar una serie de validaciones que comparan las demás transacciones realizadas en el tiempo desde que la primera comenzó su ejecución. Si falla estas pruebas, la transacción se reinicia.

El esquema OCC-Broadcast Commit (OCC-BC) utiliza las ideas de OCC pero agrega una función de broadcast para adecuarse a los sistemas de tiempo real. En esta estrategia es posible abortar una serie de transacciones conflictivas de manera tal que no se desperdicie tiempo de ejecución. Si una transacción T pasa las validaciones, se envía una señal al resto de las transacciones activas que puedan tener conflictos con T con el objetivo de notificar un conflicto y obligarlas a reiniciarse. Bajo este esquema no es necesario realizar validaciones con transacciones ya completadas, pues si hubiera conflicto alguno ya se habría reiniciado mediante una señal. En este caso, validar una transacción es equivalente a completarla.

OCC-Sacrifice es una extensión del esquema OCC-BC que tiene en cuenta el valor de las prioridades de las transacciones. OCC-Sacrifice compara las prioridades de la transacción completada T con todas aquellas que tiene conflictos. Si existe una transacción T' con mayor prioridad que T, entonces T es reiniciada. Caso contrario se realiza el commit. Si bien este esquema agrega un concepto tan importante a los sistemas de tiempo real como lo son las prioridades, tiene la desventaja de que es capaz de cancelar transacciones terminadas

en favor de otras que quizás no puedan alcanzar su deadline.

Otra extensión de OCC-BC es OCC-Wait. En este caso se añade una cola de espera para las transacciones que se encuentren en estado conflictivo. La transacción se mantiene esperando que las otras de mayor prioridad en la cola realicen su commit. Si lo hacen, es posible que haya generado conflictos y deba reiniciarse la transacción. Caso contrario, se realiza el commit y las transacciones conflictivas son abortadas.

Finalmente, se puede definir el esquema Wait 50, una extensión de OCC-Wait donde se define un conjunto conflictivo como un conjunto de transacciones que tienen conflictos con la transacción que se quiere validar. Si al menos 50 % de dichas transacciones tienen mayor prioridad que la validante, entonces se reinicia. Caso contrario se realiza el commit.

Una consideración más general de estos últimos esquemas es Wait k, donde k es el porcentaje de transacciones conflictivas que superan la prioridad de la validante. En este caso, OCC-Wait equivale a Wait 100 (pues espera que todas las transacciones conflictivas se completen), OCC-BC equivale a Wait 0 (pues no espera a ninguna transacción conflictiva) y Wait 50 se comporta de la forma previamente explicada.

En su trabajo, Aldarmi plantea una situación conflictiva encontrada a la hora de comparar los esquemas OCC contra los esquemas de locking. Dos investigadores, Haritsa y Huang, trabajaron sobre las ventajas de cada estilo de control de concurrencia. Los estudios de Haritsa lo llevaron a concluir que OCC era superador a 2PL, mientras que los estudios de Huang lo llevaron a concluir exactamente lo contrario. Los resultados contradictorios se explicaron teniendo en cuenta la hipótesis de los trabajos: Haritsa utilizó una base de datos pequeña y Huang una base de datos grande. En [8] Haritsa concluye que existe un punto de cruce a la hora de evaluar la efectividad de las estrategias con respecto al volumen de datos.

2.7.5. Control de Concurrencia Especulativo

Una desventaja de los controles de concurrencia optimistas en el ámbito de las BDTR es que los conflictos entre las transacciones no son detectados hasta la fase de validación, momento en el cual suele ser muy tarde para reiniciar la transacción. Basada en esta observación, Bestavros planteó en [3] una nueva clase de protocolos para control de concurrencia llamado control de concurrencia especulativo (SCC). Estos protocolos utilizan computaciones redundantes para mantener la serialización.

El funcionamiento detrás de los SCC se basa en combinar las ideas de OCC y locking. Una transacción se ejecuta normalmente bajo un protocolo OCC hasta que encuentra un conflicto. En este momento se produce una copia de la transacción llamada sombra, la cual ejecuta bajo un esquema de locking y funciona a modo de referencia limpia de la original. Si la transacción T finaliza sin inconvenientes, se realiza el commit. Si encuentra un inconveniente, esta se elimina y su sombra es promovida a la versión primaria pasando a ejecutar bajo un protocolo OCC. De esta forma, cancelar una transacción no implica perder todo el progreso realizado.

Existen diversos protocolos basados en SCC, pero aquí solo se comentan algunos. SCC-Ordered-Based genera $n!$ sombras por cada transacción. Por el costo computacional de crear sombras, se planteó SCC-Conflict-Based que genera un máximo de n^2 sombras por cada transacción. Finalmente se propuso SCC k-Shadow (SCC-kS), que permite un máximo de k sombras. Una de las versiones más comunes es SCC-2S, que permite un máximo de dos sombras por transacción (una primaria y una de espera).

2.7.6. Control de Concurrencia Multiversion

Estas estrategias de control de concurrencia se basan en sacar provecho de la redundancia de datos, permitiendo que cada actualización sobre valores genere una nueva versión de los datos. De esta forma es posible que las transacciones accedan a los datos generados en distintos momentos de tiempo, logrando que las operaciones no generen conflictos entre sí. Este esquema no resulta muy factible por tres motivos importantes. Primero, almacenar las copias de la información requiere de una gran cantidad de espacio de almacenamiento. Segundo, las operaciones de lectura y escritura se complejizan de forma significativa al requerir especificar la versión de los datos sobre la que se quiere trabajar. Tercero, comprobar la serialización de los datos se vuelve una tarea considerablemente más compleja.

2.7.7. Control de Concurrencia Basado en Similaridad

La idea de la similaridad no es nueva en la práctica, ya que se ha usado en diversas aplicaciones con gran éxito. Por ejemplo, en el área de agricultura se puede considerar que el cambio entre dos muestras de temperatura tomadas consecutivamente no sea significativo. En general, es sensato asumir que un conjunto de muestras medidas en intervalos de tiempo cercanos son idénticas. En [12], se intenta formalizar este concepto: es posible no considerar las muestras más nuevas y utilizar medidas viejas, siempre y cuando se logre establecer un período razonable y justificable de cuándo es despreciable la diferencia.

2.8. Seguridad en Bases de Datos

Debido al auge de los sistemas en tiempo real en el marco de áreas con información sensible (bancos, criptomonedas, bolsa, etc.) en [9] se estudia el concepto de bases de datos en tiempo real seguras (BDTRS). El objetivo es proteger la confidencialidad de la información en BDTR, agregando capas de seguridad para mantener la información inaccesible a los usuarios no autorizados.

El modelo de seguridad utilizado para implementar el control de acceso a la información es el modelo Bell-LaPadula (BLP), basado en dos conceptos:

- Una transacción puede leer un objeto si su clasificación es mayor o igual que la clasificación del objeto
- Una transacción puede escribir en un objeto si su clasificación es menor o igual que la clasificación del objeto

Bajo esta idea es necesario asignar clasificaciones tanto a las transacciones como a los objetos. Para lograr dicho objetivo, a cada transacción T se le asigna un vector $P = (level, prio)$, donde $level$ es el nivel de clasificación y $prio$ es el nivel de prioridad de dicha transacción. De esta forma, si dos transacciones que quieren acceder a un recurso tienen el mismo nivel de clasificación, se utiliza el nivel de prioridad para decidir cuál debería tener acceso al objeto.

Capítulo 3

Estado Del Arte

En este capítulo se discuten brevemente algunas aplicaciones prácticas de las BDTR, así como diversas investigaciones realizadas en el marco del tema. Al ser un tema muy novedoso, la gran mayoría de los sistemas profesionales son desarrollos propietarios debido a la amplia variedad de requerimientos. Aún así, es posible hallar software comercial que simula el tiempo real mediante el uso de estrategias generales.

3.1. Software Comercial

Google Firebase es una plataforma de Google utilizada para el desarrollo de aplicaciones móviles. Entre los diversos servicios que provee se encuentra Realtime Database. Esta es una base de datos en tiempo real estructurada como un árbol JSON que provee una API para realizar operaciones sobre ella. Para realizar las actualizaciones en tiempo real, la base de datos local se convierte en una instancia de la base de datos general. De esta forma es posible aprovechar la velocidad de procesamiento local y simular el tiempo real mediante la alta velocidad. Para sincronizar cambios globales, la API se encarga de almacenar datos en la nube y notificar al resto de las instancias. RethinkDB es una base de datos en tiempo real que utiliza el formato JSON. Esta fue diseñada desde un principio utilizando C++ y se ofrece como un software de código abierto. Proveen su propio lenguaje de query (ReQL) y simula el tiempo real mediante la alta velocidad de su arquitectura. OrientDB es otra base de datos de código abierto que provee almacenamientos en forma de grafos, documentos clave-valor y objetos. La potencia de OrientDB a la hora de ofrecer tiempo real recae en el uso de referencias de memoria para recorrer la base de datos.

3.2. Investigaciones

[7] se presenta como una tesis de maestría cuyo objetivo fue decidir la mejor BDTR comercialmente disponible para una aplicación de viajes compartidos. Entre las alternativas estudiadas (Google Firebase, MongoDB, Cassandra, OrientDB), MongoDB y Firebase dieron los mejores resultados.

En [14] se desarrolla el refinamiento de los requerimientos temporales de un sistema denominado "DeeDS"luego de haber terminado su prototipado y obtener información relevante sobre la ejecución.

En [15] se plantea una comparación entre tres bases de datos NoSQL (Couchbase, MongoDB y RethinkDB) y su viabilidad como BDTR en contextos de un sistema multihilos y otro de un solo hilo. En esta ocasión, la conclusión obtenida señaló a Couchbase como la mejor base de datos para el problema.

[5] propone un nuevo algoritmo de scheduling llamado AEDF-TAL-DS, que utiliza ontologías y relaciones semánticas para mejorar los tiempos de ejecución de las consultas.

[17] plantea los problemas de la política EDF en el contexto de BDTR distribuidas para móviles, y propone un nuevo algoritmo de scheduling para solucionarlo. Obtienen pequeñas mejoras, pero significativas para su contexto.

En [2] se presenta una variante de MongoDB denomidada RT-MongoDB, que apunta a ofrecer una BDTR basada en el servicio estándar de MongoDB.

Finalmente, en [6] se desarrolla una implementación en JAVA de un TAD para representar consistencia temporal en una BDTR.

3.3. Temas Dictados en la Cátedra

Durante el dictado de la cátedra, se desarrollaron otros tres temas: bases de datos espacio-temporales, information retrieval y NoSQL. En esta sección se discute brevemente la relación de las BDTR con cada uno de ellos.

3.3.1. Bases de Datos Espacio-Temporales

Las bases de datos espacio-temporales presentan un contexto cargado con oportunidades para los sistemas de tiempo real. Las aplicaciones que utilizan geolocalización tienen una creciente demanda por datos constantemente actualizados. Se puede pensar, por ejemplo, en servicios como Google Maps y Waze

que requieren información de último momento para planificar rutas de movilidad. Otro caso es el de los robots autónomos, que en base a datos del entorno y sus propias directivas deben trazar trayectorias eficientes. Una última área a destacar es la de las telecomunicaciones: para los sistemas de comunicación de dispositivos móviles no solo es importante tener en cuenta la posición del receptor en todo momento, sino que también deben considerar los paquetes de información como datos de tiempo real (pues pueden expirar).

3.3.2. Information Retrieval

Las BDTR son de mucha ayuda en el proceso de information retrieval. El campo donde más intersección tienen es el de information retrieval en tiempo real, que permite analizar de información de manera veloz y confiable. Un caso de uso posible es la recuperación de información compleja de un usuario luego del proceso de inicio de sesión. En esta situación es necesaria una buena velocidad de respuesta para no frustrar al usuario, lo cual se vuelve posible mediante la imposición de deadlines, el diseño de una base de datos que tenga en cuenta estas necesidades y modelos eficientes de information retrieval.

3.3.3. NoSQL

NoSQL es quizás el campo que provee la mayor cantidad de oportunidades para el desarrollo de las BDTR. Como se presentó en el Capítulo 2, una de las motivaciones para desarrollar las BDTR son las limitaciones de las bases de datos convencionales. Esto incluye al SQL.

Actualmente la mayoría de los desarrollos de BDTR se dan en el contexto de NoSQL ya que su estructura complementa perfectamente las necesidades de tiempo real y sus principios están en línea con las características que estamos dispuestos a eliminar de la base de datos.

En la sección anterior se mencionaron distintos software comercialmente disponibles, todos ellos trabajan con NoSQL. De igual manera, fueron presentados algunos experimentos donde se utilizó esta tecnología con buenos resultados.

Capítulo 4

Conclusiones

Las BDTR son un área relativamente joven, con mucho desarrollo por delante. Se nutren de una combinación ideal entre bases de datos y sistemas en tiempo real, dos áreas que presentan constantes avances e ideas aplicables para muchos otros problemas. Si bien a lo largo de este trabajo se presentaron diversos aspectos teóricos, estos no alcanzan para cubrir los conocimientos necesarios para el trabajo efectivo en una BDTR y la información aquí expuesta representa una pequeña fracción de todo el conocimiento disponible.

Se pudo observar cómo se relacionan los problemas de scheduling y concurrencia, que tan sencillamente se solucionan en las bases de datos convencionales, con las deadlines de la información generando así una dependencia casi imposible de eliminar entre las dos áreas mencionadas anteriormente.

Dando una opinión personal, el trabajo de investigación preparado durante el cuatrimestre me resultó sumamente interesante. Nunca había tenido la oportunidad de desarrollar un tema con tanta profundidad y descubrí que tengo mucho interés en el área de bases de datos. Comencé con la idea de investigar las aplicaciones de las BDTR a los videojuegos en línea, pero terminé interesándome en los problemas de scheduling y concurrencia que tan poco disfruté al cursar las materias de Sistemas Operativos.

Como trabajos de investigación futuros, podemos proponer algunas áreas de estudio:

- Desarrollar nuevos algoritmos de scheduling para sistemas de alta latencia. Es un área donde la distancia entre los actores que utilizan la información es tan grande que debe tenerse en cuenta al organizar transacciones, y nos juega en contra a la hora de cumplir las deadlines. Como pertenecientes a estos sistemas podemos llegar a considerar las telecomunicaciones de larga distancia, o la comunicación espacial.

- El diseño de una BDTR general para un dominio específico. Si bien esto parece ser complicado, creo que debería ser posible identificar un dominio donde todos los sistemas tengan requerimientos de tiempo similares. Quizás se podría trabajar sobre alguna rama de la industria pesada, como la automotor o la del acero.
- El diseño de una BDTR para videojuegos en línea. Los MMOG (siglas en inglés de massively multiplayer online game) son videojuegos donde un gran número de personas interactúan en simultáneo con el entorno o entre sí, presentando así nuevos desafíos a la hora de sincronizar la información entre los actores. Los equipos de desarrollo de los MMOG deben trabajar constantemente con la consideración de la latencia entre los jugadores, y para ello se debieron desarrollar sistemas que respondan en tiempo real a los cambios de información. Ejemplos de estos juegos incluyen World Of Warcraft y RuneScape.

Bibliografía

[1] S. A. Aldarmi. *Real-Time Database Systems: Concepts and Design*. 1998.

[2] R. Andreoli, T. Cucinotta y D. Pedreschi. «RT-MongoDB: A NoSQL Database with Differentiated Performance». En: *CLOSER* (2021).

[3] A. Bestavros. «Speculative Algorithms for Concurrency Control in Responsive Databases». En: *Responsive Computer Systems: Steps Toward Fault-Tolerant Real-Time Systems*. 1995, págs. 143-165.

[4] S. Biyabani, J. Stankovic y K. Ramamritham. «The integration of deadline and criticalness in hard real-time scheduling». En: *Real-Time Systems Symposium* (1988).

[5] F. A. E. Bouaziz y W. Jaziri. «Improving the quality of service of real-timedatabase systems through a semantics-based scheduling strategy». En: *International Journal of Intelligent Information and Database Systems* (2021).

[6] C. E. Buckle, D. P. Barry y J. M. Urriza. «Implementación en Java de un Modelo de Consistencia Temporal para Datos de Tiempo Real». En: *CACIC 2012* (2012).

[7] G. N. Dissanayake. «A Study on Real-Time Database Technology and Its Applications». En: *Master Theses* (2020).

[8] J. Haritsa, M. Carey y M. Livny. «Data access scheduling in firm real-time database systems». En: *The Journal of Real-Time Systems 4* (1992), págs. 203-241.

[9] J. R. Haritsa y B. George. «Secure Real-Time Transaction Processing». En: *REAL-TIME DATABASE SYSTEMS: Architecture and Techniques* (1990), págs. 141-160.

[10] J. Huang, J. Stankovic, D. Towsley y K. Ramamritham. «Experimental evaluation of real-time transaction processing». En: *Real-Time Systems Symposium* (1989).

[11] J. Huang, J. Stankovic, D. Towsley y K. Ramamritham. «On Using Priority Inheritance in Real-Time Databases». En: *Real-Time Systems Symposium* (1991).

[12] T.-W. Kuo y A. K. Mok. «Similarity Semantics And Concurrency Control». En: *REAL-TIME DATABASE SYSTEMS: Issues and Applications* (1997), págs. 39-56.

[13] D. Locke. «Applications and System Characteristics». En: *REAL-TIME DATABASE SYSTEMS: Architecture and Techniques* (1990), págs. 17-26.

[14] J. Mellin, J. Hansson y S. F. Andler. «Refining Timing Constraints Of Applications In DeeDS». En: *REAL-TIME DATABASE SYSTEMS: Issues and Applications* (1997), págs. 325-339.

[15] D. A. Pereira, W. O. de Morais y E. P. de Freitas. «NoSQL real-time database performance comparison». En: *International Journal of Parallel Emergent and Distributed Systems* (2017).

[16] K. Ramamritham. «Real-Time Databases». En: *International Journal of Distributed and Parallel Databases* (feb. de 1996).

[17] P. K. Singh y U. Shanker. «A Priority Heuristic Policy in Mobile Distributed Real-Time Database System». En: *Advances in Data and Information Sciences. Lecture Notes in Networks and Systems, vol 38* (2018).

[18] J. A. Stankovic y S. H. Son. «Misconceptions About Real-Time Databases». En: *REAL-TIME DATABASE SYSTEMS: Architecture and Techniques* (1990), págs. 9-16.