Introducción a las Bases de Datos en Tiempo Real

Agustín Gurvich

FCEIA, UNR

19/04/2022

Bases de Datos en Tiempo Real

Saud A. Aldarmi

"Real-time systems can be defined as those computing systems that are designed to operate in a timely manner"

Krithi Ramamritham

Traditional databases deal with persistent data. Transactions access this data while maintaining its consistency.

Bases de Datos en Tiempo Real

En el último tiempo, la demanda por sistemas que reaccionen a los flujos constantes de información creció cada vez más.

Bases de Datos en Tiempo Real

En el último tiempo, la demanda por sistemas que reaccionen a los flujos constantes de información creció cada vez más.

Las bases de datos tradicionales comienzan a flaquear en ciertos aspectos.

Bases de Datos en Tiempo Real

Ventajas de las Bases de Datos tradicionales:

- Descripción de los datos

Bases de Datos en Tiempo Real

Ventajas de las Bases de Datos tradicionales:

- Descripción de los datos
- Correctitud e integridad de la información

Bases de Datos en Tiempo Real

Ventajas de las Bases de Datos tradicionales:

- Descripción de los datos
- Correctitud e integridad de la información
- Acceso eficiente a los datos

Bases de Datos en Tiempo Real

Ventajas de las Bases de Datos tradicionales:

- Descripción de los datos
- Correctitud e integridad de la información
- Acceso eficiente a los datos
- Propiedades ACID para las transacciones

Bases de Datos en Tiempo Real

Desventajas de las Bases de Datos tradicionales:

- La información es volátil

Bases de Datos en Tiempo Real

Desventajas de las Bases de Datos tradicionales:

- La información es volátil
- Es importante el *cuándo* de los datos

Bases de Datos en Tiempo Real

Desventajas de las Bases de Datos tradicionales:

- La información es volátil
- Es importante el *cuándo* de los datos
- No es necesario información 100% correcta

Bases de Datos en Tiempo Real

Desventajas de las Bases de Datos tradicionales:

- La información es volátil
- Es importante el *cuándo* de los datos
- No es necesario información 100% correcta
- Podemos ser laxos con la atomicidad de las transacciones

Bases de Datos en Tiempo Real

Por todos estos problemas, las bases de datos tradicionales necesitan ciertas mejoras.

Bases de Datos en Tiempo Real

Por todos estos problemas, las bases de datos tradicionales necesitan ciertas mejoras.

Por eso tomamos inspiración de los sistemas en tiempo real.

Sistemas en Tiempo Real

Definición

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.
Además, tienen que ejecutarse dentro de un *intervalo de tiempo* determinado

Bases de Datos en Tiempo Real

No existe una definición formal de bases de datos en tiempo real (BDTR).

Podemos pensar en una BDTR como una Base de Datos que, además de cumplir restricciones sobre la información que almacena, debe cumplir con los pedidos (transacciones) dentro de un rango de tiempo estipulado.

Bases de Datos en Tiempo Real

Una definición interesante es la siguiente:

Definición (Stankovic y Hansson)

A real-time database system is a database system where timing constraints are associated with transactions and data have specific time intervals for which the data is valid.
The transaction timing constraints can be deadlines to complete by, times by which they must start, periodic invocations, deadlines, etc.
It is not necessary that every transaction has a timing constraint, only that some transactions do.
In addition to transaction timing requirements, data has time semantics as well.

Conceptos clave

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

Qué NO es una BDTR

No implica velocidad.

Qué NO es 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.

Qué NO es 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.

Qué NO es una BDTR

Podríamos pensar que los problemas se solucionan llevando toda la base de datos a memoria. No es suficiente, pues existen otras fuentes de retrasos (locks, scheduler, poder de procesamiento distribuido) que no pueden ser solucionados simplemente llevando a memoria principal toda la información.

BDTR vs Bases de Datos Temporales

Una Base de Datos Temporal (BDT) se encarga de soportar el tiempo asociado a la información (validez y transacción).

BDTR vs Bases de Datos Temporales

Una Base de Datos Temporal (BDT) se encarga de soportar el tiempo asociado a la información (validez y transacción).

Una BDTR se ve afectada por esos tiempos, generando nuevas restricciones sobre la información.

BDTR vs Bases de Datos Temporales

Una Base de Datos Temporal (BDT) se encarga de soportar el tiempo asociado a la información (validez y transacción).

Una BDTR se ve afectada por esos tiempos, generando nuevas restricciones sobre la información.

Si un sensor toma datos cada T unidades de tiempo, y tiene una duración t, se deberá volver a sensar dentro de min(T, t) unidades de tiempo para que la información sea relevante en una BDTR.

Nuevas nociones de consistencia

Este nuevo modelo de bases de datos impone nuevas restricciones.

Un dato ya no debe ser solo lógicamente consistente (satisfacer las restricciones de integridad de la información) sino que también debe serlo temporalmente (satisfacer restricciones de tiempo).

Consistencia temporal

La consistencia de los datos en el tiempo:

- Consistencia absoluta: El estado del entorno con respecto al estado de la información en la BD
- Consistencia relativa: El tiempo en el que un dato es considerado válido

Definición formal de consistencia (Ramamritham)

Definimos un item 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, el tiempo que es válido el dato luego de que se ingresa en la base de datos
- $d_{timestamp}$: El tiempo en el que se ingresó el dato

Definición formal de consistencia (Ramamritham)

Definimos un conjunto de consistencia relativa R como el conjunto de datos que tienen una validez relativa R_{rvi}

Definición formal de consistencia (Ramamritham)

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:
   1. Consistencia absoluta: $(tiempo\_actual - d_{timestamp}) \leq d_{avi}$
   2. Consistencia relativa: $\forall d' \in R, |d_{timestamp} - d'_{timestamp}| \leq R_{rvi}$

Definición formal de consistencia (Ramamritham)

En otras palabras:

- La consistencia absoluta de d implica que el dato siga siendo válido (que no haya expirado)
- La consistencia relativa de d implica que si hay otro dato dentro del conjunto consistente, entonces ese dato ingresó dentro en un tiempo prudencial a la base de datos

Definición formal de consistencia: Ejemplo

Supongamos que nuestra BD 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:
$R = \{Tipo\_De\_Objeto\}$, $Tipo\_De\_Objeto_{avi} = 5$, $R_{rvi} = 5$ Esto implica:

- Solo clasificaremos el tipo del objeto que pasa frente a la cámara
- Como cada 5 segundos cambia el objeto, el dato es válido (*avi*) por 5 segundos
- De la misma forma podemos deducir el *rvi* del conjunto

Definición formal de consistencia: Ejemplo

Supongamos ahora que nuestra BD se utiliza para guardar mediciones del clima en un campo. Cada 5 minutos debe muestrearse la temperatura y cada 10 la presión atmosférica. El intervalo de validez relativa para el conjunto R es de 2 minutos,

En este caso podemos pensar que:
$R = \{temperatura, presion\}, temperatura_{avi} = 5, presion_{avi} = 10, R_{rvi} = 2$

Definición formal de consistencia: Ejemplo

Supongamos que $tiempo\_actual = 90$ (pasaron 90 minutos desde el inicio del sistema). y sean dos mediciones $temperatura = (30, 5, 81)$ y $presion = (1007, 10, 84)$.

Estos datos son *absolutamente consistentes*, pero no son *relativamente consistentes*!

Frecuencia de muestreo

Mantener la consistencia absoluta es sencillo, basta con ingresar datos frescos a la BD de manera frecuente.

Por otro lado, mantener la consistencia relativa es más complicado. Requiere que los datos ingresados sean observados en intervalos cercanos entre ellos.

Frecuencia de muestreo

Mantener la consistencia absoluta es sencillo, basta con ingresar datos frescos a la BD de manera frecuente.

Por otro lado, mantener la consistencia relativa es más complicado. Requiere que los datos ingresados sean observados en intervalos cercanos entre ellos.

La consistencia relativa es importante solamente cuando los datos se usan para derivar información adicional.

Ejemplos

Vamos a presentar distintos dominios donde se pueden aplicar las BDTR. Consideramos:

Ejemplos

Vamos a presentar distintos dominios donde se pueden aplicar las BDTR. Consideramos:

- Tamaño de la BD
- Tiempo de respuesta de lectura/escritura
- Consistencia absoluta
- Consistencia relativa
- Durabilidad de la BD

Aviones y naves espaciales

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

Aviones y naves espaciales

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

Industria

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

ACID

En una base de datos tradicional, las propiedades ACID (atomicidad, consistencia, aislamiento, durabilidad) previenen las inconsistencias entre transacciones.

Para las BDTR, esto deja de ser la principal preocupación.
Intercambiamos consistencia por velocidad de procesamiento, agregando nuevas técnicas para preservar la integridad de los datos.

Concurrencia

La concurrencia es natural en las bases de datos, y más aún en los sistemas distribuidos.

Concurrencia

La concurrencia es natural en las bases de datos, y más aún en los sistemas distribuidos.

Los algoritmos clásicos de eliminación de concurrencia en bases de datos no suelen ser útiles en esta área (al menos no de la forma en la que se plantean).

Concurrencia

La concurrencia es natural en las bases de datos, y más aún en los sistemas distribuidos.

Los algoritmos clásicos de eliminación de concurrencia en bases de datos no suelen ser útiles en esta área (al menos no de la forma en la que se plantean).

Abortar una transacción que bloquea a otra puede causar que ambas pierdan su deadline.

Deadline

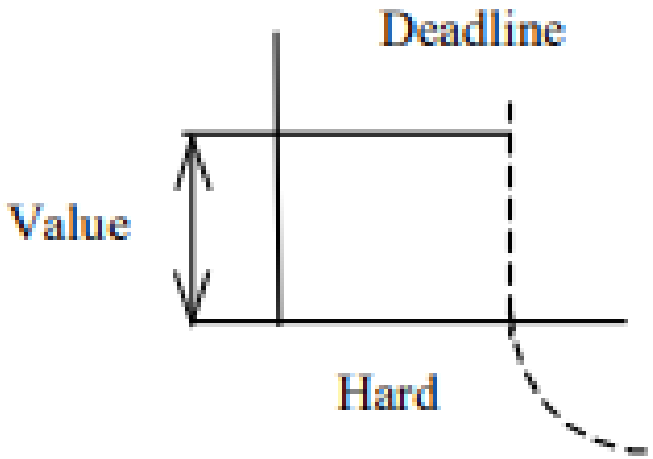
Una *deadline* es un límite de tiempo en el cual debe cumplirse la transacción.

Se corresponden con situaciones de la vida real y pueden ser representadas con funciones de valor (value functions).

Hard-Deadline

Una *deadline dura* es una transacción cuyo incumplimiento tiene consecuencias serias (inclusive catastróficas). Estas transacciones son críticas para la seguridad del sistema, y son la razón por la que se evalúa perder ciertas certezas de las bases de datos convencionales.

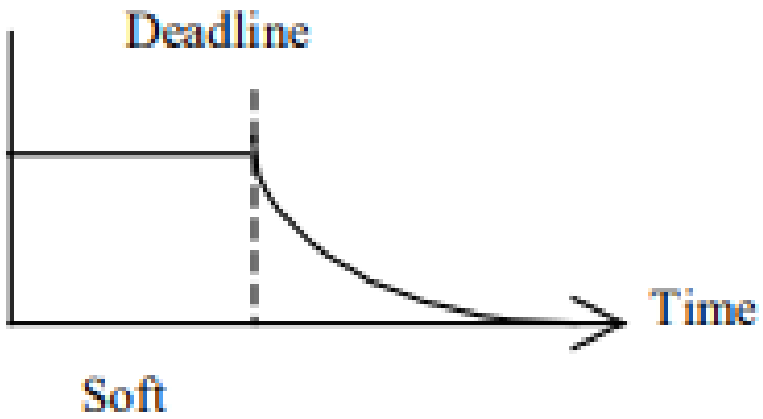
Su función de valor es la siguiente:



Soft-Deadline

Una *deadline blanda* es una transacción cuyo cumplimiento es deseado, pero no indispensable. En general, no es un problema su incumplimiento, sino que la deadline es muy blanda pues es posible ganar cierto valor de la transacción aunque ya haya pasado su tiempo de máxima ganancia.

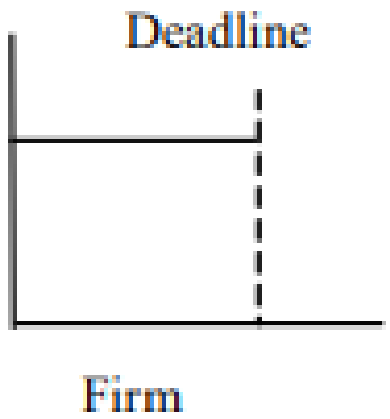
Su función de valor es la siguiente:



Firm-Deadline

Una *deadline firme* es una deadline blanda cuyo incumplimiento no suma ni resta valor al sistema. Es deseable que dicha transacción se cumpla a tiempo, pero no es un inconveniente si no lo hace.

Su función de valor es la siguiente:



Comparación de deadline: blanda vs firme

Cuando hay sobrecarga de tareas en la base de datos, el incumplimiento de deadlines blandas aumenta *exponencialmente*. En cambio, con las 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.

Tareas en el sistema

En un sistema no es posible prever todas las situaciones que pueden darse, pero sí podemos modelar con cierta estimación las acciones a realizar. Se clasifican en 3 tipos:

- Periódicas: Las tareas que realiza constantemente el sistema

Hay que tener en cuenta la cantidad de operaciones de cada tipo a la hora de elegir un *scheduler* para la BDTR

Tareas en el sistema

En un sistema no es posible prever todas las situaciones que pueden darse, pero sí podemos modelar con cierta estimación las acciones a realizar. Se clasifican en 3 tipos:

- Periódicas: Las tareas que realiza constantemente el sistema
- Aperiódicas: Tareas conocidas pero que no se pueden prever

Hay que tener en cuenta la cantidad de operaciones de cada tipo a la hora de elegir un *scheduler* para la BDTR

Tareas en el sistema

En un sistema no es posible prever todas las situaciones que pueden darse, pero sí podemos modelar con cierta estimación las acciones a realizar. Se clasifican en 3 tipos:

- Periódicas: Las tareas que realiza constantemente el sistema
- Aperiódicas: Tareas conocidas pero que no se pueden prever
- Esporádicas: Tareas aperiódicas que surgen en emergencias

Hay que tener en cuenta la cantidad de operaciones de cada tipo a la hora de elegir un *scheduler* para la BDTR

Scheduler

Uno de los puntos críticos para cumplir con las deadlines es la elección de un scheduler adecuado.

Para eso, es necesario estimar el tiempo de ejecución de las tareas en el sistema (determinismo, peor caso, probabilidades).

Un scheduler debe saber si una transacción es **factible**, y se dice que es **óptimo** si puede programar todas las tareas que otros schedulers pueden.

Scheduler

Uno de los puntos críticos para cumplir con las deadlines es la elección de un scheduler adecuado.

Para eso, es necesario estimar el tiempo de ejecución de las tareas en el sistema (determinismo, peor caso, probabilidades).

Un scheduler debe saber si una transacción es **factible**, y se dice que es **óptimo** si puede programar todas las tareas que otros schedulers pueden. Esto no involucra que deba poder programar TODAS las tareas.

Tipos de scheduler

Un scheduler pede ser:

- Apropiativo, si puede cancelar otras tareas
- No apropiativo, si una vez iniciada una tarea esta no puede cancelarse
- Híbrido, si solamente soporta la apropiación en ciertos puntos de la ejecución

Tipos de scheduler

Un scheduler pede ser:

- Apropiativo, si puede cancelar otras tareas
- No apropiativo, si una vez iniciada una tarea esta no puede cancelarse
- Híbrido, si solamente soporta la apropiación en ciertos puntos de la ejecución

Los schedulers apropiativos son adecaudos para trabajar en sistemas de tiempo real.

Complejidad de un scheluder

La complejidad computacional de un algoritmo de scheduler se mide con 2 factores:

1. Computabilidad: ¿Es posible que este conjunto de transacciones se complete bajo los límites de tiempo que propone?
2. Decidibilidad: ¿Es posible crear un programa que sea factible para ejecutar estas transacciones?

Schedulers con prioridad

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 (Priority-Based Scheduling).

Estas propiedades pueden asignarse estáticamente (al momento de iniciarse) o dinámicamente (variar durante la ejecución).

Schedulers con prioridad

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 (Priority-Based Scheduling).

Estas propiedades pueden asignarse estáticamente (al momento de iniciarse) o dinámicamente (variar durante la ejecución).

El concepto de prioridad no es **único**. Algunas medidas de prioridad posibles son: criticidad, deadline, tiempo de espera, tiempo requerido de ejecución, edad, etc.

Rate-Monotonic policy

Política apropiativa que asigna prioridades de acuerdo a la *periodicidad* de la transacción. Mientras más corto el período de la transacción, más prioridad se le asigna.

Most-Critical-First policy

Divide todo el universo posible de tareas en niveles de prioridad, en base a su importancia en el sistema. Ayuda a generar planeamientos confiables al intentar que se ejecuten las tareas más críticas sin importar la carga del sistema.
La desventaja es que se vuelve muy complicado agregar tareas al sistema.

Earliest-Deadline-First policy

Política apropiativa que utiliza las deadlines de las tareas como la referencia de prioridad. Esto permite la siguiente propiedad:

La política de 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. Es decir, es posible realizar todos los cálculos necesarios y cumplir las deadlines dado que se cumpla dicha cota superior.

Earliest-Deadline-First policy

Política apropiativa que utiliza las deadlines de las tareas como la referencia de prioridad. Esto permite la siguiente propiedad:

La política de 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. Es decir, es posible realizar todos los cálculos necesarios y cumplir las deadlines dado que se cumpla dicha cota superior.

El principal problema de esta política es la sobrecarga de tareas.

Mejora: Tomar en cuenta la deadline factible más cercana

Políticas basadas en funciones de valor

Anteriormente presentamos funciones que representan la importancia del cumplimiento de una deadline para el sistema. Podemos considerar que el valor representa la criticidad de la tarea.

Existen una familia de algoritmos basados en utilizar estas funciones para calcular las prioridades.

Value-Density policy

Esta política selecciona las tareas de acuerdo a su densidad de valor ($\frac{valor}{tiempo\_de\_computo}$). Es decir, las acomoda de acuerdo a los beneficios que trae al sistema.

Es un algoritmo voraz, por lo que puede dejar pasar oportunidades para acomodar otras tareas.

Combinando deadlines y criticidad

Las prioridades de las tareas también pueden calcularse tomando en cuenta qué tan críticas son para el sistema y su deadline. Se propusieron varios esquemas para lograrlo.

ALG1 y ALG2

Estos esquemas planean el orden de las tareas según sus deadlines, sin importar la criticidad. Si este plan no es factible, entonces comienzan a eliminar tareas en el sistema. ALG1 quita tareas una a la vez, de menor a mayor criticidad. ALG2 lo hace con las de menor criticidad y deadline más lejana.

Aplican EDF en condiciones normales, pero en sobrecarga aplican MCF y una combinación de EDF y MCF respectivamente.

Criticalness-Deadline First

Las prioridades se asignan según la fórmula $\frac{deadline - llegada}{criticidad}$. CDF mejora la performance sobre técnicas que toman en cuenta ambos factores por separado, y demostró trabajar mejor al ordenar tareas críticas perdiendo menos tareas poco importantes.

Criticalness-Deadline First

Conclusiones de Stankovich y Ramamritham

First, in a CPU-bound system, the CPU scheduling algorithm has a significant impact on the performance of a real-time system, and dominates all of the other types of protocols.
Second, in order to obtain good CPU scheduling performance, both criticalness and deadlines of a task should be considered in priority assignment.

Sobrecarga

En condiciones normales, un planeamiento de scheduler podrá ordenar cada tarea de manera tal que cumpla su deadline. Cuando no es posible, se dice que el sistema está *sobrecargado*.

Los sistemas son susceptibles a la sobrecarga intermitente cuando:

- Se dan una serie de situaciones de emergencia
- Un scheduler efectivo necesita información completa de la ejecución de sus tareas, por lo que puede fallar si intenta hacer análisis de casos
- El análisis del peor caso fue muy optimista
- El hardware no se comporta como fue esperado

Sobrecarga

Se pueden optar por dos actitudes frente a la sobrecarga:

1. Rechazar las tareas entrantes (enfoque garantizado)
2. Remover las tareas de menor valor hasta que se regularice el sistema (enfoque robusto)

El enfoque robusto puede mejorarse con una *cola de removidos*, que puede guardar las tareas removidas y ejecutarlas si alcanza el tiempo.

Enfrentando la realidad

Cuando el sistema está sobrecargado, es imposible cumplir con todas las tareas.

Los investigadores llegaron a un consenso que *se deberían programar al menos las tareas más importantes*. Lo que entra en juego en cada sistema particular, es decidir *cuáles* lo son. Como no existe una definición formal, es imposible decidir objetivamente qué constituye una tarea importante.

Referencias

Ramamritham, Krithivasan. (1993). Real-time databases.

Aldarmi, Saud A. (1998). Real-Time Database Systems: Concepts and Design.

Kuo, Tei-Wei, Lam, Kam-Yiu. (2002). Real-Time Database Systems Architecture and Techniques