Gestión de Transacciones

Problema

La gestión de transacciones trata el problema de asegurar la bd en un estado consistente aún en casos de *acceso concurrente* y de *fallas*.

Aspectos críticos:

- Correctitud de ejecuciones concurrentes
- Recuperación de fallas

Concepto Clásico de Transacción

Transacción:

Ejecución de un programa que incluye operaciones de acceso a la BD que verifica las propiedades **ACID**.

Atomicity (Atomicidad): Una transacción es una unidad atómica de procesamiento, o bien se realiza por completo o no se realiza en absoluto.

Consistency (Consistencia): Preserva estados consistentes.

Isolation (Aislamiento): Los efectos de una transacción son invisibles a las transacciones concurrentes.

Durability (Permanencia): Los efectos de una transacción validada son permanentes.

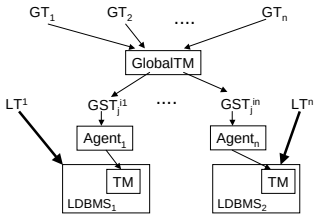
Modelo de Transacción

- Operaciones significativas de una transacción
  - begin_transaction
  - commit, abort
  - read, write
- Terminación de una transacción:
  - Commit:
    - Cambios durables sin posibilidad de deshacer
    - Cambios visibles a otras transacciones
  - Abort:
    - Cualquier cambio se debe cancelar

Concepto de Transacción en BD Heterogéneas

Existen dos tipos de transacciones en BD Heterogéneas:

- transacciones *locales*: acceden datos de una sola BD
- transacciones *globales*: acceden datos de varias BDs



Correctitud de ejecuciones concurrentes

La ejecución entremezclada de transacciones sin control puede resultar en una base de datos inconsistente.

Problemas de:

- **Pérdida de la actualización:**
  Diferentes ejecuciones entrelazadas pueden producir valores finales diferentes.
- **Lectura inconsistente**:
  Existen ejecuciones entrelazadas que violan RI.
- **Lectura irreproducible**:
  Existen ejecuciones entremezcladas donde el valor desplegado no es el mismo.

Serializabilidad

Intuitivamente:

Cualquier ejecución de un conjunto de transacciones es correcta si es libre de interferencias.

Asumiendo que transacciones individuales son correctas, una ejecución **serial** de transacciones es **correcta** desde que las operaciones que se ejecutan serialmente lo hagan en **aislamiento** y no pueden interferir entre ellas.

Serializabilidad:

Ejecuciones concurrentes que son equivalentes a ejecuciones seriales.

Serializabilidad por conflicto

- **Dos operaciones están en conflicto si**
  - Se aplican sobre el mismo objeto
  - Una de ellas es una operación de escritura
  - Provienen de dos transacciones diferentes
- Definición 1:

  Dos operaciones p y q están en conflicto en el estado s, sii
  (state(state(o,p),q) ≠ state(state(o,q),p)) or
  (return(o,q) ≠ return(state(o,p),q)) or
  (return(o,p) ≠ return(state(o,q),p))

Dos operaciones que no están en conflicto son compatibles o conmutan.

Serializabilidad por conflicto asegura que pares de operaciones en conflicto aparecen en el mismo orden en dos ejecuciones equivalentes.

Definición de Transacción

Definición 2

Una transacción T es un orden parcial donde:

1. Operaciones en T son operaciones sobre objetos más las primitivas : *begin, commit* y *abort.*
2. Todas las operaciones de una transacción siguen a su operación de begin.
3. Una transacción puede commit o abort pero no ambas.
4. Todas las operaciones de una transacción preceden asu commit o abort.
5. El orden de las operaciones en conflicto está definido.

Definición de Historia

Definición 3:

Sea T = { T1, T2, ..., Tn} un conjunto de transacciones. Una **historia (completa) H** de la ejecución concurrente de un conjunto de transacciones T contiene todas las operaciones y primitivas invocadas por las transacciones en T e indican el orden parcial en el que estas operaciones ocurren.

1. Las operaciones en H son exactamente aquellas en T1, ... Tn.

2. El orden de las operaciones dentro de cada transacción es preservado.

3. El orden de cada par de operaciones en conflicto está definido.

Historias equivalentes (por Conflicto) y Seriales

Definición 3:

Dos historias son equivalentes (por conflicto) si

1. Son definidas sobre el mismo conjunto de transacciones y tienen las mismas operaciones.

2. Todo par de operaciones en conflicto p, q de transacciones validadas T1 y T2 respectivamente están ordenadas en la misma forma en ambas historias.

Definicion 4:

Una historia es serial si para todo par de transacciones T1, T2, todas las operaciones de T1 aparecen antes que todas las operaciones de T2 o viceversa.

Historia Serializable

Definicion 5:

Una historia es serializable (por conflicto) si su proyección por commit es equivalente a una historia serial.

Proyección por commit de una historia H es la historia que contiene solamente todas las operaciones de transacciones validadas (terminadas por commit).

Operaciones en conflicto inducen requerimientos de ordenes entre las transacciones, esta dependencia de ordenes entre transacciones la denotamos por *C* relacion binaria entre transacciones.

Orden de Serializabilidad

Serializabilidad demanda que el orden de serializabilidad debe ser aciclico.

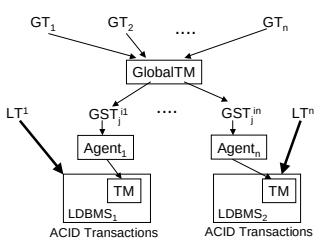
Definicion 6:

__ H_{comm} *es serializable (por conflicto) sii* $\forall t \in T_{comm} \neg (t\ C^*\ t)$.

Dada una historia se puede determinar si es serializable si la proyeccion de sus operaciones de transacciones terminadas por commit no induce ciclos en las depencdencias de orden.

Gestión de Transacciones en Bases de Datos Distribuídas Heterogéneas

Transacciones en BDD Heterogéneas



Técnicas tradicionales para asegurar atomicidad y consistencia en BDD no son apropiadas por:

- Autonomía de diseño
- Autonomía de ejecución
- Autonomía de comunicación

Transacciones en BDD Heterogéneas

Técnicas tradicionales para asegurar atomicidad y consistencia en BDD no son apropiadas por:

- Autonomía de diseño:
  No se pueden hacer cambios a los manejadores de transacciones locales para acomodarlos al manejador de transacciones global.
  Además se podrían dejar aplicaiones pre-existentes inoperativas.
- Autonomía de ejecución:
  Cada manejador de transacciones local tiene control completo sobre todas las transacciones (GST y LT) que se realizan en su sitio, lo que incluye la posibilidad de abortar en cualquier punto cualquiera de las transacciones.
- Autonomía de comunicación:
  Los manejadores de transacciones locales no comparten sus controles con el manejador de transacciones global.

Caracterización del nivel de Autonomía

Definir la **interface** con la cual el sitio local interactúa con el GTM.
La interface se caracteriza por las **operaciones** que ofrece:
Operaciones de *Transacciones y* Operaciones de *Información de Status*

Operaciones de *Transacciones:*

begin, end, read, write, abort, commit,

prepare to commit: transacciones esperan el commit o abort del GTM.

service request: es equivalente a someter todas las acciones de una transacción a la vez, desde el begin al commit.

Caracterización del nivel de Autonomía (II)

Operaciones de *Información de Status*:

Get-wait-for-graph: devuelve el local-wait-for-graph para ser usado en la detección de deadlocks globales.

Get-serialization-order: devuelve información sobre el orden de los commit.

Inquire: Encuentra el status de una transacción.

Disable transaction class: indica que algunas clases de transacciones no son posibles de commit en los manejadores de transacciones locales.

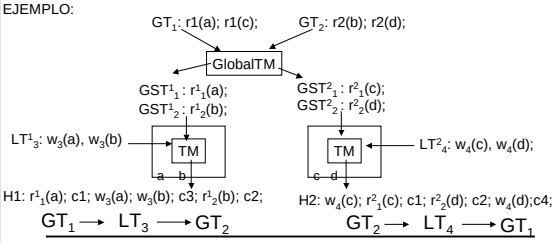
Funciones del GTM

- Garantizar las propiedades ACID de las transacciones globales aun en la presencia de transacciones locales que no conoce.
- Garantizar ejecuciones libres de dealocks de las transacciones globales.
- Proveer formas de recuperación sobre distintos tipos de fallas.

Problema de Serializabilidad Global

- Cada LTM usa distintos protocolos de control de concurrencia:
  Two Phase Locking (2PL), Timestamp ordering (TO), Serialization Graph Testing (SGT), etc.
- Control de información encapsulada en los LTM

EJEMPLO:



Soluciones para Serializabilidad Global

Para evitar ciclos el GTM dbe tomar acciones.
Las acciones dependen del grado de conocimiento que el GTM tenga respecto a los mecanismos de control de concurrencia locales.

Distintos escenarios:

- Escenario Base: El GTM solo sabe que cada LTM genera historias locales serializables y libres de deadlocks.
  Cada LTM es considerado como un “unlabeled black box”
- El GTM conoce propiedades adicionales (labels) de las fuentes locales (black box).

Tipos de acciones tomas por el GTM

- Pesimista: Las GT son demoradas para evitar ciclos.
- Optimista: Potenciales ciclos son detectados y eliminados abortando a las GT.

Pesimista vs. Optimista

- Pesimista: No genera abortos de GT pero resulta en un grado de concurrencia menor.
- Optimista: Incrementa la concurrencia pero resulta en un gran número de de abortos de GT.

Integración con *unlabeled DBMS*

En el ejemplo, si usa el **enfoque optimista** debería:

- Retardar la ejecución de GT_2 hasta que GTM esté seguro que puede ocurrir un ciclo.
- GTM no tiene control sobre DB_2 $\Rightarrow$ **No aplicable**

(GT_2 *tiene que ejecutarse después que* LT_4 *para evitar la dependencia* $GT_2 \rightarrow LT_4$ *No hay como saber cuando* LT_4 *se completa,)*

Solución: El GTM se asegura que el arco $GT_1 \rightarrow GT_2$ se coloque en el grafo de serializacion local de DB2.

Asi el arco asegura que $GT_2 \rightarrow LT_4 \rightarrow GT_1$ no puede ser generado en BD2 pues generaría ciclo local.

Método del ticket [Georgakopoulos *et al.* 1991]

Idea: Se fuerza a que GT_1 escriba algun dato en cada sitio que accesa datos y se fuerza a GT_2 a leer esos datos.

Método del Ticket

Caso GT_1 y GT_2 son concurrentes se debe asegurar que en un sitio no se genere el arco $GT_1 \rightarrow GT_2$ mientras que en el otro se genere el $GT_2 \rightarrow GT_1$ en sus grafos de serialización locales respectivamente.

$\Rightarrow$ Se logra con el uso de un ítem de dato especial: ticket
El valor del ticket indica el orden de serialización de la GT en el sitio.

Características:

- Un solo ticket por sitio local
- Solo GT pueden acceder los tickets
- Cada GT ejecutandose en el sitio local debe:
  leer el valor del ticket,
  incrementarlo y
  escribir su valor incrementado.

Método del Ticket

El GTM asigna consistencia en el orden de ejecución global de las subtransacciones en los sitios locales usando los valores de los tickets usados por cada subtransación.

Las subtransacciones de dos GT se ejecutan en una forma consistente en cada uno de los sitios locales si:

los valores de los tickets de todas las subtransacciones de una GT son menores que los valores de los tickets de todas las subtransacciones de la otra GT.

Integración con *labeled DBMS*

Mayor concurrencia - Menor autonomía
Conocimiento de propiedades de las historias locales.

- **DBMS Fuertemente Serializables:**
- **DBMS Basados en Serialización Puntual (*SP-schedules*)**
- **DBMS Fuertemente Recuperables**
- **DBMS Rigurosos**

DBMS Serializables

Suponemos que los LTM usan el mecanismo de control de concurrencia Timestamp Ordering (TO) (*ordenamiento por marca de tiempo)*

Del Ejemplo: H1: $r^1_1(a)$; c1; $w_3(a)$; $w_3(b)$; c3; $r^1_2(b)$; c2;
H2: $w_4(c)$; $r^2_1(c)$; c1; $r^2_2(d)$; c2; $w_4(d)$;c4;

Sea *t2* el TO de GT2, y *t1* el TO de GT1.
Como GT1 y GT2 no se solapan $\Rightarrow$ *t2 > t1*

El mecanismo de TO asegura que las GT son serializables entonces no puede haber dependencias GT2 $\rightarrow$ .... $\rightarrow$ GT1 en el grafo de serializacion local.
Si se deja hacer $w_4(d)$ *se genera esa dependencia entonces el LTM en DB2 aborta a* LT_4.

Idea: Correr GT serialmente. Si no se solapan le seran asignados TO incrementales serializados en el orden correcto en cada sitio y la HG sera serializable.

DBMS Fuertemente Serializables

DBMS Fuertemente Serializables:

Ti y Tj en S son *Fuertemente Serializables* si la última operación deTi (commit o abort) precede a la primera operación de Tj. Entonces hay alguna historia serial equivalente a S donde Ti precede a Tj.

Algoritmos de:

- Site-graph protocolo (basado en lock-per-site) [Breitbart and Silberschatz, 1988]
  - El GTM mantiene un grafo bipartito,
    - Nodos: GT y sitios,
    - Arcos: Conectan GT con los sitios donde se ejecutan.
  - Si no hay ciclo para GTi, GTi se ejecuta sino GTi es demorada hasta que desaparezca el ciclo.
- Altruist locking protocolo [Salem *et al.,* 1989]
  - Si hay un LDBMS que GT1 y GT2 deben acceder , GT2 no lo puede hacer antes que GT1 haya finalizado su ejecucion alli.

Serialización Puntual (*SP-schedules*)

DBMS Basados en Serialización Puntual (*SP-schedules*)

Sea S: T1, T2,...., Tn. S es sp sii existe un mapeo de transacciones a acciones tal que:

1) sp(Ti) = op_k donde op_k pertenece a Ti
2) Si sp(Ti) ocurre antes que sp(Tj) en S, entonces no son permitidas dependencias de la forma Tj $\rightarrow$ .... $\rightarrow$ Ti en un grafo de serializabilidad local para S.

Historias locales-sp son fuertemente serializables $\Rightarrow$ se pueden usar el criterio previo. Pero si cada LTM notifica al GTM de antemano que accion constituye el punto de serializacion entonces se puede obtener serializacion global mas eficiente.

IDEA: Puntos de serializacion de transacciones que lleven a ciclos son ejecutadas en el mismo orden en todos los sitios [Mehrotra *et al*. 1992]

DBMS Fuertemente Recuperables:

DBMS Fuertemente Recuperables:

Sk es *Fuertemente Recuperable* si para todo para de transacciones Ti, Tj si Ti esta en conflicto directo con Tj en Sk y Tj commits en Sk entonces Tj no puede ejecutar su commit hasta despues que Ti commits.

IDEA:

Transacciones que se procesan serialmente deben hacer el commit de sus operaciones serialmente.
[Breitbart and Silberschatz, 1991]

DBMS Rigurosos

DBMS Rigurosos:

Sk es *Riguroso* si para todo par de transacciones Ti y Tj;
si Ti esta en conflicto directo con Tj en Sk y Tj commits en Sk
entonces Tj no puede ejecutar sus operaciones con conflicto hasta
que Ti commits.

IDEA:

[Breitbart et al, 1991] muestra que si las historias locales son rigurosas y el GTM no ordena los commit de una transaccion tal que todas las transacciones previas hayan completado su ejecucion, entonces la HG es serializable.

El 2PL protocolo genera HG serializables.

Resumen de Integración con *labeled DBMS*

	DBMS	GTM
Más grales.	Fuertemente Serializables	Coordina las ejecuciones de todas las operaciones de GT tal que GTs se ejecuten serialmente.
	Basados en Serialización Puntual (*SP-schedules*)	Coordina los puntos de serialización de las operaciones de las transacciones.
	Fuertemente Recuperables	Coordina los commit de las operaciones para ejecutar las transacciones serialmente.
Más restrictivas	Rigurosos	No hay coordinación del GTM entre GT. Difiere el commit de las GST hasta el end de la GT.

Nociones Débiles de Serializabilidad

Se considera Serializabilidad a dos niveles:

- Considerando dos tipos de datos (locales y globales)
- Considerando tres tipos de restricciones
  - *Local* : restricciones envolviendo solo un sitio local.
  - *Global:* mas de un sitio, envuelve items de datos globales.
  - *Local /Global:* restricciones solamente en sitios locales, pero envuelve items locales y globales.

En lugar de usar el concepto de serializabilidad se definen nociones de consistencia alternativa:

- Dos Niveles Serializable
- Ordenamiento Cooperativo de Serialización
- Multi-Serializabilidad
- Quasi- Serializable

Nociones de consistencia alternativas

Dos Niveles Serializable [Mehrotra *et al.* 1991]

- Una historia global es 2LS si es localmente serializable y sus proyecciones al conjunto de GT es serializable.

- Ordenamiento Cooperativo de Serialización [Ramamritham and Chrysanthis, 1994]

  Un conjunto de transacciones cooperativas puede estar formado por transacciones que cooperan sobre el mismo objeto para mantener su consistencia. Consistencia puede ser mantenida si:

  - Otras transacciones que no pertenecen al conjunto pueden ser serializables respecto a todas las transacciones del conjunto.

  - Cuando una transaccion establece una dependencia con otra transaccion la misma correspondencia debe ser establecida entre todas las transacciones de sus conjuntos de transacciones cooperativas.

Nociones de consistencia alternativas (II)

- Multidatabase- Serializabilidad [Mehrotra et al. 1992]

  Emula protocolos de 2PC usando transacciones "redo".

  - El commit de una GT puede ser decidido usando el 2PC del agente que hace la interface con el LTM (sin la participaacion del LTM)

  - Una GST no necesita entrar al estado de *prepare-to-commit* durante la fase de decisión.

Consecuencias:

- El estado de la BD sobre la cual se ejecuta el redo de la transaccion es el mismo que que el visto por la transaccion abortada.

- El redo de la transacción no debe invalidar ninguna otra transacción activa o commited.

Nociones de consistencia alternativas (II)

- Quasi-Serializabilidad [Du and Elmagarmid, 1989]

Basa la correctitud de la ejecución de un conjunto de transacciones globales y locales en la noción de historia quasi-serial, que especifica que solo transacciones globales son ejecutadas serialmente.

Una historia es quasi-serial si:

- Todas las historias locales son serializables por conflicto.

- Existe un orden total de todas las GT, g_m y g_n donde g_m precede a g_n y todas las operaciones de g_m preceden a todas las opercaiones de g_n en todas las historias locales donde ellas aparecen.

Problema de Atomicidad Global

Atomicidad Global: Todas las ST de una GT o bien commit o abort.
Autonomía ⇒ BDDH no exportan los "prepared state" como en BDD.

Problemas: Transacciones globales no atómicas, Historias globales incorrectas.

Ejemplo: DB1: a, DB2: c, GT_1: $r_1(a)$ $w_1(a)$ $w_1(c)$

1. Commit global
2. Terminación exitosa en DB2 pero abort en DB1 (autonomía local)
3. Transacción local LT_2 en DB1,
   LT_2: $r_2(a)$ $w_2(a)$
4. GTM re intenta el $w_1(a)$ ⇒ nueva transacción local en DB1 !
   LT_3: $w_3(a)$; c_3

⇒ vista de la historia global incorrecta!

H_1: $r_2(a)$; $w_2(a)$; c2; $w_3(a)$; c3; HG_1: $r_1(a)$; $r_2(a)$; $w_2(a)$; c2; $w_1(a)$; c1;

Solución a la Atomicidad Global

Según interface de los LTM: prepare-to-commit o no prepare-to-commit

- **Disponen de la operación de *prepare-to-commit***

Usan un protocolo de commit atomico (2PC)

Trabajo del protocolo 2PC :

- Cuando termina la ejecucion de una GT el GTM somente un *prepare-to-commit* a cada uno de los sitios donde la transaccion se ejecuta.
- El sitio como respuesta vota si commit o abort la GT
- El sitio cede sus derechos de abortar unilateralmente la transacción.
- El GTM dependiendo de los votos recibidos decide si abortar o continuar.

Solución a la Atomicidad Global (II)

NO disponen de la operación de *prepare-to-commit*

Es posible que una GT commit en un sitio local y aborte en otro.

Tres mecanismos para asegurar global atomicidad:

Redo: Todos las operaciones de write de la transaccion son instaladas ejecutando un redo de la transaccion.

Retry: La transaccion abortada es ejecutada otra vez en forma entera (no solo sus operaciones de write)

Compensable: En cada sitio donde una GST de una GT que aborto es commited, una transaccion compensada es ejecutada para deshacer los efectos semánticos del commit de la subtransacción.

Problema de Deadlock Global

Cada LTM usa un mecanismo para detectar y recuperarse de deadlocks locales, pero existe la posibilidad de un deadlock global que no sea detectado por el GTM.

Ejemplo: DB1: a, b DB2: c, d

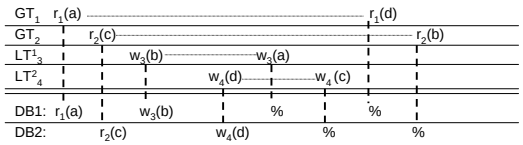
GT_1: $r_1(a)$ $r_1(d)$

GT_2: $r_2(c)$ $r_2(b)$

LT^1_3: $w_3(b)$ $w_3(a)$ LT^2_4: $w_4(d)$ $w_4(c)$

Situación Posible:

- LT_3 esperando por GT_1
- LT_4 esperando por GT_2
- GT_1 esperando por LT_4
- GT_2 esperando por LT_3



Conclusiones

Las propiedades de ACID son valiosas pero no absolutas:

- Atomic → Estructuradas pero flexible
- Consistent → Adecuadas en aplicaciones especificas al sentido semantico.
- Isolated → Cooperativas, Negociables
- Durable → Responsabilidad auditable, Explicables

Serializabilidad Global en BD Heterogeneas

Problema:

Ejecucion serial de transacciones globales no asegura serializabilidad global !!

Ejemplo:

BD1: a, b BD2:e,d

GT1: r1(a); r1(e); L1: w1(a), w1(b);
GT2: r2(b); r2(d); L2: w2(e), w2(d);

H1: r1(a); c1; w1(a); w1(b); c3; r2(b);c2;
H2: w2(e); r1(e); c1; r2(d); c2; w2(d);c2;