Chapter 5: Transactions in Federated Databases

Transaction Management in Federated DBMSs (1)

Common assumption: all transactions under control of a global distributed transaction manager (TM).

Transaction Management in Federated DBMSs (2)

1. Now: local transactions unknown to the GTM in addition.
How to achieve global serializability?

Is the following sufficient?
- Same concurrency-control protocol, and
- each server guarantees local serializability?

Example (1)
1. $D_1=\{a\}$; $D_2=\{b, c\}$,
1. global transactions $T_1=r(a) \ w(b)$; $T_2=w(a) \ r(c)$,
1. local transactions $T_3=r(b) \ w(c)$.
1. Why are $T_1$ and $T_2$ global, and $T_3$ is local?
1. Global TM is 'conservative'; serial execution ($T_1; T_2$).
1. Possible local histories:
   - server 1: $s_1= r_1(a) \ w_2(a)$
   - server 2: $s_2= r_3(b) \ w_1(b) \ r_2(c) \ w_3(c)$
1. What are the local serialization orders?

Example (2)
1. Corresponding global history:
   $s= r_1(a) \ r_3(b) \ w_1(b) \ w_2(a) \ r_2(c) \ w_3(c)$
1. $s_1, s_2$ conflict-serializable: $s_1=_{c} T_1 T_2$; $s_2=_{c} T_2 T_3 T_1$

Direct vs. Indirect Conflict
1. Important distinction: 
   direct conflict vs. indirect conflict.
Global Serializability – Overview

1. Global serializability based on local guarantees.
   1. Strictness, and commit is delayed,
   2. commitment ordering.
2. Tickets (do not require additional local guarantees; ticket = additional local database object that global transactions must manipulate)

Strictness (1)

1. Undo typically works with before images.
2. Problem illustration – original value of x: 1
   1. \( w_1(x, 2); w_2(x, 3); \text{abort}_1 \)
   2. \( w_1(x, 2); w_2(x, 3); \text{abort}_1; \text{abort}_2 \)
3. Are these histories
   a. recoverable,
   b. cascadeless?

Strictness (2)

\( \text{strictness} :\) write(\( x, \text{val} \)) is delayed until all transactions that have written \( x \) before have committed or aborted, + cascadelessness.

Strictness (3)

1. \( T_1 = w_1[x] w_1[y] w_1[z] c_1 \)
2. \( T_2 = r_2[u] w_2[x] r_2[y] w_2[y] c_2 \)
3. \( H_9 = w_1[x] w_1[y] r_2[u] w_2[x] w_1[z] c_1 r_2[y] w_2[y] c_2 \)
4. \( H_{10} = w_1[x] w_1[y] r_2[u] w_1[z] c_1 w_2[x] r_2[y] w_2[y] c_2 \)
5. Which histories are strict?
Strictness (4)
Strictness alone does not suffice to ensure global serializability – example:
1. $D_1 = \{a,b\}; D_2 = \{c,d\}$,
2. global transactions: $T_1 = w(a) w(d), T_2 = w(c) w(b)$,
3. local transactions: $T_3 = r(a) r(b), T_4 = r(c) r(d)$,
4. local histories:
   - $s_1 = w_1(a) c_1 r_3(a) r_3(b) c_3 w_2(b) c_2$
   - $s_2 = w_2(c) c_2 r_4(c) r_4(d) c_4 w_1(d) c_1$
5. $s_1$ and $s_2$ are strict, but serialization orders differ.
6. Observe that order of commits of global transactions differs between nodes.

Strictness (5)
1. Do not issue commit right away.
2. Global transaction is commit-deferred
   - global TM sends commit to local sites only after the local sites have acknowledged execution of all operations.

Strictness (6)
1. Continuation of example:
   - $D_1 = \{a,b\}; D_2 = \{c,d\}$
   - global transactions: $T_1 = w(a) w(d), T_2 = w(c) w(b)$
   - local transactions: $T_3 = r(a) r(b), T_4 = r(c) r(d)$
   - local histories:
     - $s_1 = w_1(a) c_1 r_3(a) r_3(b) c_3 w_2(b) c_2$
     - $s_2 = w_2(c) c_2 r_4(c) r_4(d) c_4 w_1(d) c_1$
   - Not allowed. Deadlock situation.

Strictness (7)
1. Continuation of example:
   - $D_1 = \{a,b\}; D_2 = \{c,d\}$
   - global transactions: $T_1 = w(a) w(d), T_2 = w(c) w(b)$
   - local transactions: $T_3 = r(a) r(b), T_4 = r(c) r(d)$
   - local histories:
     - $s_1 = w_1(a) c_1 r_3(a) r_3(b) c_3 w_2(b) c_2$
     - $s_2 = w_2(c) c_2 r_4(c) r_4(d) c_4 w_1(d) c_1$
   - $c_2$ takes place strictly after $c_1$ in $s_1$:
     - $r_3(a)$ after $c_1$ because of strictness.
     - $w_2(b)$ after $r_3(b)$ according to serialization order.
     - $c_2$ after $w_2(b)$ because of commit-deferredness.
   - Analogously, $c_1$ takes place strictly after $c_2$.
Strictness (8)

Theorem: let \( s \) be a global history for \( s_1, \ldots, s_n \), where \( s_i \) is strict for \( 1 \leq i \leq n \), and all global transactions are commit-deferred. \( \Rightarrow s \) is globally serializable.

Strictness (9)

Why is commit-deferredness, together with local serializability, not sufficient? Why do we need strictness as well?

Strictness (10)

Running example:
- \( D_1 = \{a, b\} \); \( D_2 = \{c, d\} \)
- global transactions:
  - \( T_1 = \text{w}(a) \ \text{w}(d) \)
  - \( T_2 = \text{w}(c) \ \text{w}(b) \)
- local transactions: \( T_3 = \text{r}(a) \ \text{r}(b) \)
  - \( T_4 = \text{r}(c) \ \text{r}(d) \)
Global TM issues \( \text{w1}(a) \) and \( \text{w1}(d) \).
  Then \( \text{w2}(c) \) and \( \text{w2}(b) \).
- No guarantee that operations take place in that order. (At least when global TM does without handshaking.)
- Thus, local histories so far could be:
  - \( s_1 = \text{w1}(a) \ \text{r3}(a) \ \text{r3}(b) \ \text{c3} \ \text{w2}(b) \)
  - \( s_2 = \text{w2}(c) \ \text{r4}(c) \ \text{r4}(d) \ \text{c4} \ \text{w1}(d) \)
- Global TM now issues \( \text{c1} \), on both nodes.

Commitment Ordering (1)

Global serializability based on local guarantees.
- Approach 2: commitment ordering (COCSR).
- Commit order conflict serializability := Two transactions conflict. \( \Rightarrow \) Commit operations in conflict order.
Commitment Ordering (2)

A DBMS that implements commitment ordering works as follows – example:

1. applications issue \( w_1(x) \), \( w_2(x) \), \( c_2 \), strictly after each other.
2. Now \( c_1 \) is submitted to the DBMS – DBMS must reject it, must abort \( T_1 \).

Execution order \( w_1(x) \) \( w_2(x) \) \( c_2 \) \( c_1 \) does not have commit operations in conflict order.

Commitment Ordering (3)

- COCSR, and commits of global transactions are executed one after the other ("handshaking").
  \[ \Rightarrow \text{History is globally serializable.} \]

Commitment Ordering (4)

- Example – two histories:
  - \( s_1 = r_1(a) c_1 w_3(a) w_3(b) c_3 r_2(b) c_2 \)
  - \( s_2 = w_4(c) r_1(c) r_2(d) r_4(e) c_1 c_2 \)

Observe that \( s_2 \) is not COCSR.

- Why is COCSR alone not sufficient?
  - Suppose \( c_1 \) and \( c_2 \) are issued in parallel; \( c_1 \) is delayed.
  - This may result in commit order \( c_2 c_4 c_1 \).
  - I.e., COCSR on both sites, but no local serializability.

Tickets (1)

- In what follows: nodes do not offer additional properties such as strictness or COCSR, only CSR locally.
- Example again:
  - \( D_1 = \{a, b\}; D_2 = \{c, d\} \),
  - global transactions \( T_1 = r(a) r(c) \); \( T_2 = r(b) r(d) \),
  - local transactions \( T_3 = w(a) w(b) \); \( T_4 = w(c) w(d) \)
  - The following local histories are not globally correct:
    - \( s_1 = r_1(a) c_1 w_3(a) w_3(b) c_3 r_2(b) c_2 \)
    - \( s_2 = w_4(c) r_1(c) r_2(d) c_2 w_4(d) c_4 \)
  - Objective must be:
    - execute \( T_2 \) only if it cannot be part of a cycle.
Tickets (2)

1. Convert indirect conflict into direct one that global TM can recognize.
2. Additional data object in each database ("ticket").
3. Comparable with logical timestamp.
4. Each global transaction
   - reads ticket,
   - writes back incremented value. "Take-a-ticket" operation.

Tickets – Example

1. \( D_1 = \{a\}, D_2 = \{b, c\} \),
2. global transactions \( T_1 = r(a) w(b); T_2 = w(a) r(c) \)
3. local transaction \( T_3 = r(b) w(c) \)
4. possible local histories (without tickets):
   - \( s_1 = r_1(a) c_1 w_2(a) c_2 \)
   - \( s_2 = r_3(b) w_1(b) c_1 r_2(c) c_2 w_3(c) c_3 \)
5. use of tickets:
   - \( s_1 = r_1(I_1) w_1(I_1+1) r_1(a) c_1 r_2(I_1) w_2(I_1+1) w_2(a) c_2 \)
   - \( s_2 = r_3(b) r_1(I_2) w_1(I_2+1) w_1(b) c_1 \)
   - \( r_2(I_2) w_2(I_2+1) r_2(c) c_2 w_3(c) c_3 \)
6. \( s_2 \) not conflict-serializable any more.
7. The following history would be OK:
   - \( s_2 = r_1(I_2) w_1(I_2+1) w_1(b) c_1 r_2(I_2) w_2(I_2+1) r_2(c) c_2 r_3(b) w_3(c) c_3 \)

Optimistic vs. Conservative Ticket Approach

1. Conservative ticket approach:
   - global TM fixes order of transactions taking tickets.
2. Also possible:
   - optimistic approach to ensure the same serialization order on all components.
   - "Ticket order" must be the same everywhere.
   - Ticket-order graph:
     - global TM administers it.
     - Edge \( T_i \rightarrow T_j \): a subtransaction of \( T_i \) has read ticket before \( T_j \).
     - Cycle \( \Rightarrow \) abort.

Discussion

1. "Lightweight" approach.
2. Except for the fact that nodes must 'tolerate' additional data objects, no further interference with node autonomy.