Concurrency Control

R&G - Chapter 17

Review

- ACID transaction semantics.
- Today: focus on isolation property
  - Serial schedules safe but slow
  - Try to find schedules equivalent to serial ...

Smile, it is the key that fits the lock of everybody’s heart.

Anthony J. D’Angelo,
The College Blue Book

Conflicting Operations

- Need a tool to decide if 2 schedules are equivalent
- Use notion of “conflicting” operations

Definition: Two operations conflict if:
  - They are by different transactions,
  - they are on the same object,
  - and at least one of them is a write.

Conflict Serializable Schedules

Definition: Two schedules are conflict equivalent iff:
  - They involve the same actions of the same transactions, and
  - every pair of conflicting actions is ordered the same way

Definition: Schedule S is conflict serializable if:
  - S is conflict equivalent to some serial schedule.

Note, some “serializable” schedules are NOT conflict serializable
  - A price we pay to achieve efficient enforcement.

Conflict Serializability – Intuition

- A schedule S is conflict serializable if:
  - You are able to transform S into a serial schedule by swapping consecutive non-conflicting operations of different transactions.

Example:

R(A) W(A)       R(A) W(A)       R(B) W(B)
R(A) W(A)       R(B) W(B)

R(A) W(A) R(B) W(B)  \[=\]  R(A) W(A) R(B) W(B)

Conflict Serializability (Continued)

Here’s another example:

R(A) W(A)
R(A) W(A)
R(B) W(B)

Serializable or not?????

NOT!
Dependency Graph

- **Dependency graph:**
  - One node per Xact
  - Edge from Ti to Tj if:
    - An operation Oi of Ti conflicts with an operation Oj of Tj
    - Oi appears earlier in the schedule than Oj.

- **Theorem:** Schedule is conflict serializable if and only if its dependency graph is acyclic.

Example

- A schedule that is not conflict serializable:

<table>
<thead>
<tr>
<th></th>
<th>T1: R(A), W(A), R(B), W(B)</th>
<th>T2: R(A), W(A), R(B), W(B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The cycle in the graph reveals the problem. The output of T1 depends on T2, and vice-versa.

An Aside: View Serializability

- Alternative (weaker) notion of serializability.
- Schedules S1 and S2 are view equivalent if:
  1. If Ti reads initial value of A in S1, then Ti also reads initial value of A in S2
  2. If Ti reads value of A written by Tj in S1, then Ti also reads value of A written by Tj in S2
  3. If Ti writes final value of A in S1, then Ti also writes final value of A in S2
- Basically, allows all conflict serializable schedules + “blind writes”

<table>
<thead>
<tr>
<th>T1: R(A)</th>
<th>W(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2: W(A)</td>
<td></td>
</tr>
<tr>
<td>T3: W(A)</td>
<td></td>
</tr>
</tbody>
</table>

Notes on Serializability Definitions

- View Serializability allows (slightly) more schedules than Conflict Serializability does.
  - Problem is that it is difficult to enforce efficiently.
- Neither definition allows all schedules that you would consider “serializable”.
  - This is because they don’t understand the meanings of the operations or the data.
- In practice, Conflict Serializability is what gets used, because it can be enforced efficiently.
  - To allow more concurrency, some special cases do get handled separately, such as for travel reservations, etc.

Two-Phase Locking (2PL)

<table>
<thead>
<tr>
<th>Lock</th>
<th>S</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>√</td>
<td>~</td>
</tr>
<tr>
<td>X</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

**rules:**
- Xact must obtain a S (shared) lock before reading, and an X (exclusive) lock before writing.
- Xact cannot get new locks after releasing any locks.

2PL guarantees conflict serializability

But, does not prevent Cascading Abort.
Strict 2PL

- **Problem:** Cascading Aborts
- **Example:** rollback of T1 requires rollback of T2!

\[
\begin{array}{c|c|c}
\text{T1:} & \text{R(A), W(A), R(B), W(B), Abort} \\
\text{T2:} & \text{R(A), W(A)} \\
\end{array}
\]

- Strict Two-phase Locking (Strict 2PL) protocol:
  
  Same as 2PL, except:
  
  - **Locks released only when transaction completes**
  - \(a\) transaction has committed (commit record on disk),
  - \(b\) transaction has aborted and rollback is complete.

Next ...

- A few examples

### Non-2PL, A = 1000, B = 2000, Output = ?

<table>
<thead>
<tr>
<th>Lock X(A)</th>
<th>Lock S(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(A)</td>
<td></td>
</tr>
<tr>
<td>A := A - 50</td>
<td></td>
</tr>
<tr>
<td>Write(A)</td>
<td></td>
</tr>
<tr>
<td>Unlock(A)</td>
<td></td>
</tr>
<tr>
<td>Read(A)</td>
<td></td>
</tr>
<tr>
<td>Unlock(A)</td>
<td></td>
</tr>
<tr>
<td>Lock X(B)</td>
<td></td>
</tr>
<tr>
<td>Read(B)</td>
<td></td>
</tr>
<tr>
<td>B := B + 50</td>
<td></td>
</tr>
<tr>
<td>Write(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(B)</td>
<td>Unlock(A)</td>
</tr>
<tr>
<td>Unlock(B)</td>
<td></td>
</tr>
<tr>
<td>PRINT(A+B)</td>
<td></td>
</tr>
</tbody>
</table>

### 2PL, A = 1000, B = 2000, Output = ?

<table>
<thead>
<tr>
<th>Lock X(A)</th>
<th>Lock S(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(A)</td>
<td></td>
</tr>
<tr>
<td>A := A - 50</td>
<td></td>
</tr>
<tr>
<td>Write(A)</td>
<td></td>
</tr>
<tr>
<td>Lock X(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(A)</td>
<td></td>
</tr>
<tr>
<td>Read(A)</td>
<td></td>
</tr>
<tr>
<td>Lock S(B)</td>
<td></td>
</tr>
<tr>
<td>Read(B)</td>
<td></td>
</tr>
<tr>
<td>B := B + 50</td>
<td></td>
</tr>
<tr>
<td>Write(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(B)</td>
<td>Unlock(A)</td>
</tr>
<tr>
<td>Unlock(B)</td>
<td></td>
</tr>
<tr>
<td>PRINT(A+B)</td>
<td></td>
</tr>
</tbody>
</table>

### Strict 2PL, A = 1000, B = 2000, Output = ?

<table>
<thead>
<tr>
<th>Lock X(A)</th>
<th>Lock S(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(A)</td>
<td></td>
</tr>
<tr>
<td>A := A - 50</td>
<td></td>
</tr>
<tr>
<td>Write(A)</td>
<td></td>
</tr>
<tr>
<td>Lock X(B)</td>
<td></td>
</tr>
<tr>
<td>Read(B)</td>
<td></td>
</tr>
<tr>
<td>B := B + 50</td>
<td></td>
</tr>
<tr>
<td>Write(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(B)</td>
<td>Unlock(A)</td>
</tr>
<tr>
<td>Unlock(A)</td>
<td></td>
</tr>
<tr>
<td>Unlock(B)</td>
<td></td>
</tr>
<tr>
<td>PRINT(A+B)</td>
<td></td>
</tr>
</tbody>
</table>
Lock Management

- Lock and unlock requests handled by **Lock Manager**
- LM keeps an entry for each currently held lock.
- Entry contains:
  - List of xacts currently holding lock
  - Type of lock held (shared or exclusive)
  - Queue of lock requests

Lock Management, cont.

- When lock request arrives:
  - Does any other xact hold a conflicting lock?
    - If no, grant the lock.
    - If yes, put requestor into wait queue.
- Lock upgrade:
  - xact with shared lock can request to upgrade to exclusive

Example

<table>
<thead>
<tr>
<th>Lock_X(A)</th>
<th>Lock_S(B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(A)</td>
<td>Lock_S(A)</td>
</tr>
<tr>
<td>Write(A)</td>
<td></td>
</tr>
</tbody>
</table>

Deadlocks

- **Deadlock**: Cycle of transactions waiting for locks to be released by each other.
- Two ways of dealing with deadlocks:
  - prevention
  - detection
- Many systems just punt and use Timeouts
  - What are the dangers with this approach?
Deadlock Detection

- Create and maintain a “waits-for” graph
- Periodically check for cycles in graph

Deadlock Detection (Continued)

Example:

T1: S(A), S(D), S(B)
T2: X(B) S(D), S(C), X(A)
T3: X(C) X(B)
T4: X(B)

Deadlock Prevention

- Assign priorities based on timestamps.
- Say Ti wants a lock that Tj holds
  Two policies are possible:
  
  **Wait-Die**: If Ti has higher priority, Ti waits for Tj; otherwise Ti aborts
  
  **Wound-wait**: If Ti has higher priority, Tj aborts; otherwise Ti waits

- Why do these schemes guarantee no deadlocks?
- Important detail: If a transaction re-starts, make sure it gets its original timestamp. -- Why?

Summary

- Correctness criterion for isolation is “serializability”.
  - In practice, we use “conflict serializability,” which is somewhat more restrictive but easy to enforce.
- Two Phase Locking and Strict 2PL: Locks implement the notions of conflict directly.
  - The lock manager keeps track of the locks issued.
  - **Deadlocks** may arise; can either be prevented or detected.