Transaction Management - Concurrency Control – Part 1

From Chapters 16, 17
Motivation

- Concurrent execution
  - Why is this desirable?
- Crash recovery
  - Crashes not desirable but unavoidable!
- Transactions
Transactions: The ACID properties

- **A** tomicity:
- **C** onsistency:
- **I** solation:
- **D** urability:
Transactions API

Client Application

Begin transaction
SQL Query 1
if (...) then SQL Update 2
else SQL Update 3
...
End transaction

Database System

ODBC/JDBC Connection
Outline – Concurrency Control

- Examples
- Formal definition of serializability
- Possible solutions to concurrent execution anomalies
Goal of Concurrency Control

- Transactions should be executed so that it is as though they executed in some serial order
- Weaker variants also possible
Example

User 1: Buy 10 Snicker bars
User 2: Buy 2 Gatorade bottles

Possible order of processing at DB server:
Anomalies (Lost Update)

User 1: Buy 10 Snicker bars
User 2: Buy 2 Snicker bars

Order of processing at DB server:
DBMS’s View

U1: Read nb Snickers
U2: Read nb Snickers

U1: Reduce count Snickers by 10
U1: Write new nb Snickers back

U2: Reduce count Snickers by 2
U2: Write new nb Snickers back

T1: R(Snickers)
T1: W(Snickers)
T1: COMMIT

T2: R(Snickers)
T2: W(Snickers)
T2: COMMIT
Inconsistent-Read Anomalies

- **Dirty reads** – read uncommitted data
  - T1: R(A), W(A), R(B), W(B), Abort
  - T2: R(A), W(A), Commit

- **Unrepeatable reads**
  - T1: R(A), R(A), R(A), W(A), Commit
  - T2: R(A), W(A), Commit
Class Exercise

- Transaction Steps
- Possible Schedule
- Possible Problems

- T1: Transfer money from savings to checking
- T2: Add interest for savings account
Outline

- Examples
- Formal definition of serializability
- Possible solutions to concurrent execution anomalies
Scheduling Transactions

- **Serial schedule:**

- **Equivalent schedules:**

- **Serializable schedule:**
Conflict Serializable Schedules

- Two schedules are conflict equivalent if:

- Schedule S is conflict serializable if S is conflict equivalent to some serial schedule
Example

- A schedule that is not conflict serializable:

| T1:  | R(A), W(A), R(B), W(B) |
| T2:  | R(A), W(A), R(B), W(B) |

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

- **Dependency graph**: One node per Xact; edge from \( T_i \) to \( T_j \) if \( T_i \) precedes \( T_j \) and \( T_i \) conflicts with \( T_j \)
- 2 actions **conflict** if at least one is a write
- **Theorem**: Schedule is conflict serializable if its dependency graph is acyclic
- Certain serializable executions are not conflict serializable!
Example

\[
\begin{array}{c}
\text{T1: } R(A) & W(A) \\
\text{T2: } & W(A) \\
\text{T3: } & W(A)
\end{array}
\]

\[
\begin{array}{c}
\text{T1: } R(A), W(A) \\
\text{T2: } & W(A) \\
\text{T3: } & W(A)
\end{array}
\]
Outline

- Examples
- Formal definition of serializability
- Possible solutions to concurrent execution anomalies
Resource Locking

- **Locking**: prevents multiple applications from obtaining copies of the same resource when the resource is about to be changed

- **Lock granularity** - size of a locked resource

- Types of lock
  - Exclusive lock (X)
  - Shared lock (S)
Explicit Locks

User 1: Buy 10 Snicker bars
User 2: Buy 2 Snicker bars

User 1:
Lock Snickers
Read nb Snickers (ns=500)
Reduce count Snickers by 10 (ns=490)
Write new nb Snickers back (ns=490)

User 2:
Lock Snickers
Read nb Snickers (ns2=500)
Reduce count Snickers by 2 (ns2=498)
Write new nb Snickers back (ns2=498)

Order of processing at DB server:
Class Exercise – Place Locks

- T1: R(Sa), \textcolor{red}{W(Sa)}, \quad \text{R(Ch), W(Ch), Abort}
- T2: \quad \textcolor{red}{R(Sa)}, W(Sa), C
Strict Two-Phase Locking

- Strict two-phase locking
  - Locks are obtained throughout the transaction

- All locks are released at the end of transaction (COMMIT or ROLLBACK)
Strict 2PL Example

- Strict 2PL
  - X(A)
  - R(A)
  - W(A)
  - X(B)
  - R(B)
  - W(B)
  - Rel(B,A)

- Not 2PL
  - X(A)
  - R(A)
  - W(A)
  - Rel(A)
  - X(B)
  - R(B)
  - W(B)
  - Rel(B)
Lock Management

- Lock and unlock requests are handled by the lock manager
- Lock table entry:
  - Number of transactions currently holding a lock
  - Type of lock held (shared or exclusive)
  - Pointer to queue of lock requests
- Locking and unlocking have to be atomic operations
- Lock upgrade: transaction that holds a shared lock can be upgraded to hold an exclusive lock
Next Time

- Deadlock prevention and detection
- Advanced locking techniques
- Lower degrees of isolation
- Concurrency control for index structures
Solution 1

1) Get exclusive lock on entire database
2) Execute transaction
3) Release exclusive lock

- Similar to “critical sections” in operating systems
- Serializability guaranteed because execution is serial!

- Problems?

Database Management Systems, R. Ramakrishnan and Johannes Gehrke
Solution 2

1) Get exclusive locks on accessed data items
2) Execute transaction
3) Release exclusive locks

- Greater concurrency

- Problems?
Solution 3

1) Get exclusive locks on data items that are modified; get shared locks on data items that are only read

2) Execute transaction

3) Release all locks

- Greater concurrency
- Conservative Strict Two Phase Locking (2PL)
- Problems?

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>X</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Solution 4

1) Get exclusive locks on data items that are modified and get shared locks on data items that are read

2) Execute transaction and release locks on objects no longer needed *during execution*

- Greater concurrency
- **Conservative Two Phase Locking (2PL)**

- Problems?
Solution 5

1) Get exclusive locks on data items that are modified and get shared locks on data items that are read, but do this during execution of transaction (as needed).

2) Release all locks

- Greater concurrency
- Strict Two Phase Locking (2PL)

- Problems?
Solution 6

1) Get exclusive locks on data items that are modified and get shared locks on data items that are read, but do this during execution of transaction (as needed)

2) Release locks on objects no longer needed during execution of transaction

3) *Cannot acquire locks once any lock has been released*
   - *Hence two-phase (acquiring phase and releasing phase)*
   - Greater concurrency
   - Two Phase Locking (2PL)

Problems?
Summary of Alternatives

- **Conservative Strict 2PL**
  - No deadlocks, no cascading aborts
  - *But* need to know objects a priori, when to release locks

- **Conservative 2PL**
  - No deadlocks, more concurrency than Conservative Strict 2PL
  - *But* need to know objects a priori, when to release locks, cascading aborts

- **Strict 2PL**
  - No cascading aborts, no need to know objects a priori or when to release locks, more concurrency than Conservative Strict 2PL
  - *But* deadlocks
Method of Choice

- **Strict 2PL**
  - No cascading aborts, no need to know objects a priori or when to release locks, more concurrency than Conservative Strict 2PL
  - **But** deadlocks

- **Reason for choice**
  - Cannot know objects a priori, so no Conservative options
  - Thus only 2PL and Strict 2PL left
  - 2PL needs to know when to release locks (main problem)
    - Also has cascading aborts
    - Hence Strict 2PL

- **Implication**
  - Need to deal with deadlocks!