Serializability

• Fallacy: being serializable is the same as being serial
• Being serializable implies that the schedule is a correct schedule
  – It will leave the database in a consistent state
  – The interleaving is appropriate and will result in a state equivalent of at least one schedule where the transactions are serially-executed.
  – It will achieve efficiency due to concurrent execution

Checking for Serializability

This schedule suffers from the lost update problem. It is not serializable as the final state of $X = 7$ which is not possible with any sequential execution (either T1:T2 or T2:T1) of these transactions.
Checking for Serializability (contd)

- **Serializability is hard to guarantee**
- Interleaving of operations happens at runtime through some scheduler
- Difficult to determine/predict how the operations of a schedule will be interleaved
- Practical Approach
  - Protocols that ensure serializability
    - Tradeoff (constrain how transactions are written, and their performance for serializability)
  - Reduce the problem of checking the whole schedule to checking only a committed projection of the schedule.

Locks and Transactions

- **Shared Locks** (multiple outstanding locks)
  - Read Locks (many transactions can read at same time)
- **Exclusive Locks** (one transaction at a time)
  - Write Locks
- **Rules** (for transaction T)
  - T must obtain a read or write lock on X before reading X.
  - T must obtain a write lock before writing to X.
  - T cannot obtain locks on items it already has locked
  - T must issue the unlock(X) operation after all read and write operations on X within T are finished.

Using locks to guarantee serializability

- **Initial values**
  - \(X=20, Y=30\).
- **Result of \(T_1 \rightarrow T_2\)**
  - \(X=50, X=80\)
- **Result of \(T_2 \rightarrow T_1\)**
  - \(X=70, Y=50\)
- **Does this work?**
  - Is locking sufficient
• Locking rules hold in this schedule
• Result of this schedule
  – X=50, Y=50
  – It does not match T1→T2, or T2→T1
• So just locking does not suffice

2-phase locking
• Basic criteria (2 separable phases)
  – Locking Phase: Obtain all locks needed in the transaction [growing/expanding phase]
    • New locks may be obtained, none may be released.
  – Unlocking Phase: Release existing locks [shrinking phase]
    • Locks may be released, none may be obtained.
All problems solved?

• Deadlock Problem
  – T1 acquires Y, T2 acquires X. Both want the others lock and are unwilling to release their locks.

2-phase hierarchical locking

• Basic criteria (2 separable phases)
  – Locking Phase: Obtain all locks needed in the transaction [growing/expanding phase]
    • New locks may be obtained, none may be released
  – Order locks, and obtain locks in predefined order.
    • Trade off deadlock avoidance for performance (less concurrency)
  – Unlocking Phase: Release existing lock [shrinking phase]
    • Locks may be released, none may be obtained

Solved? Yes!

With the above transactions, schedule on right can never happen.
Unlock can happen in any order.
Conflict Equivalence

• Two schedules are said to be conflict equivalent if the order of any two **conflicting operations** is the same in both schedules
• Conflicting Operations (different transactions)
  – Read after Write (RAW)
    • W1(X), R2(X)
  – Write after Read (WAR)
    • R2(Y), W1(Y)
  – Write after Write (WAW)
    • W1(Y), W2(Y)

Conflict Serializability

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁</td>
<td>T₂</td>
</tr>
<tr>
<td>read_int(X);</td>
<td>read_int(Y);</td>
</tr>
<tr>
<td>X:=M;</td>
<td>X:=N;</td>
</tr>
<tr>
<td>write_int(X);</td>
<td>read_int(Y);</td>
</tr>
<tr>
<td>Y:=N;</td>
<td>write_int(Y);</td>
</tr>
</tbody>
</table>

LHS is Conflict Equivalent to RHS.
RHS is a serial schedule.
Therefore, LHS is conflict serializable.

Testing Conflict Serializability

• Create a precedence graph.
  – Create a node for each transaction.
  – Create a dependency (line from one node to another) for every conflict.
• Test for cycles in precedence graph.
• No cycles => conflict serializable
  • Cycles => no serial schedule exists that is conflict equivalent to original schedule.
View Serializable

- Slightly weaker notion of serializability when compared to conflict-serializability.
- Premise
  - Each read operation of a transaction reads the result of the same write operation in both schedules
  - The write operation of each transaction must produce same results
Relationship between View and Conflict Serializability

- The two are the same under the “constrained write assumption” which assumes any transaction T that writes a value X (in other words no blind writes)
  - Reads OLD VALUE OF X
  - New X = f(OLD VALUE OF X)
- Example
  - T1: r1(X), w1(X)
  - T2: w2(X)
  - T3: w3(X)
  - The schedule r1(X); w2(X); w1(X); w3(X); is view equivalent but not conflict serializable