Review: Using Timestamps for Concurrency Control

Computer Science 460
Boston University
David G. Sullivan, Ph.D.

Timestamp-Based Concurrency Control

- Transactions are assigned unique timestamps based on when they start.
- The system ensures that all operations are consistent with a serial ordering based on the timestamps.
  - example: $\text{TS}(T2) < \text{TS}(T1)$, so must be consistent with $T2; T1$

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS = 102</td>
<td>TS = 100</td>
<td>TS = 100</td>
</tr>
<tr>
<td>$w(A)$</td>
<td>$r(C)$</td>
<td>$r(C)$</td>
</tr>
<tr>
<td>$r(A)$</td>
<td>denied</td>
<td>$r(A)$</td>
</tr>
</tbody>
</table>

T2’s read of $A$ is too late:
- if this read were allowed, T2 would read T1’s write of A
- in the equiv. serial schedule, it would read A’s old value
→ the DBMS denies the read, rolls back T2, and makes it start over
Timestamp-Based Concurrency Control (cont.)

- Transactions are assigned timestamps based on when they start.
- Each data item D has a:
  - RTS – the largest timestamp of any txn that has read D
  - WTS – the largest timestamp of any txn that has written D

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>T2</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>TS = 220</td>
<td>TS = 230</td>
<td>RTS = WTS = 0</td>
<td>RTS = WTS = 0</td>
</tr>
<tr>
<td>w(A) r(B)</td>
<td>w(A) r(B)</td>
<td>WTS = 220 RTS = 230</td>
<td>WTS = 220 RTS: no change</td>
</tr>
</tbody>
</table>

Timestamp Rules for Reads and Writes

- When T tries to read A:
  - if \( TS(T) < WTS(A) \), roll back T and restart it
    - T’s read is too late
  - else allow the read
    - set RTS(A) = max(\( TS(T) \), RTS(A))

- When T tries to write A:
  - if \( TS(T) < RTS(A) \), roll back T and restart it
    - T’s write is too late
  - else if \( TS(T) < WTS(A) \), ignore the write and let T continue
    - in the equiv serial sched, T’s write would be overwritten
  - else allow the write
    - set WTS(A) = TS(T)
Ensuring Recoverability and Cascadelessness

• Transactions are assigned timestamps based on when they start.

• Each data item D has a:
  • RTS – the largest timestamp of any txn that has read D
  • WTS – the largest timestamp of any txn that has written D
  • commit bit – used to prevent dirty reads
    • true if the writer of the current value has committed
    • false otherwise

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS = 220 w(A)</td>
<td>TS = 230 w(A)</td>
<td>RTS = WTS = 0; c = true</td>
<td>RTS = WTS = 0; c = true</td>
</tr>
<tr>
<td>r(B) commit</td>
<td>r(B) commit</td>
<td>WTS = 220; c = false</td>
<td>WTS = 230; c = false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c: no change</td>
<td>c = true</td>
</tr>
</tbody>
</table>

Timestamp Rules for Reads and Writes

• When T tries to read A: when using commit bits
  • if TS(T) < WTS(A), roll back T and restart it
    • T’s read is too late (see our earlier example)
  • else allow the read (but if c(A) == false, make T wait)
    • set RTS(A) = max(TS(T), RTS(A))

• When T tries to write A:
  • if TS(T) < RTS(A), roll back T and restart it
    • T’s write is too late (see example 2 from last lecture)
  • else if TS(T) < WTS(A), ignore the write and let T continue
    (but if c(A) == false, make T wait)
    • in the equiv serial sched, T’s write would be overwritten
  • else allow the write
    • set WTS(A) = TS(T)  (and set c(A) to false)
Other Details

- When the writer of the current value of data item A commits, we:
  - set A's commit bit to true
  - allow waiting txns try again

- When a txn T is rolled back, we process:
  - all data elements A for which WTS(A) == TS(T)
    - restore their prior state (value and timestamps)
    - set their commit bits based on whether the writer of the prior value has committed
    - make waiting txns try again
  - all data elements A for which RTS(A) == TS(T)
    - restore their prior RTS

Example of Using Timestamps and Commit Bits

- The balance-transfer example would now proceed differently.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>bal1</th>
<th>bal2</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS = 350</td>
<td>r(bal1)</td>
<td>RTS = WTS = 0</td>
<td>RTS = WTS = 0</td>
</tr>
<tr>
<td></td>
<td>w(bal1)</td>
<td>c = true</td>
<td>c = true</td>
</tr>
<tr>
<td></td>
<td>r(bal2)</td>
<td>TS = 375</td>
<td>RTS = 350</td>
</tr>
<tr>
<td></td>
<td>w(bal2)</td>
<td>WTS = 350; c = false</td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>commit</td>
<td>c = true</td>
<td>WTS = 350; c = false</td>
</tr>
<tr>
<td></td>
<td>r(bal1)</td>
<td>RTS = 375</td>
<td>c = true</td>
</tr>
<tr>
<td></td>
<td>and completes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>