Correctness Issues at the intersection of Failures and Concurrency

- Cascading rollback, recoverable schedule
- Deadlocks
  - Prevention
  - Detection

Concurrency control & recovery

Example: T_j T_i

\[
\begin{align*}
W_j(A) & \rightarrow W_i(A) \\
& \leftarrow \text{Commit } T_i \\
& \leftarrow \text{Abort } T_j
\end{align*}
\]

- Cascading rollback (Bad!)
• Schedule is conflict serializable
• T_j_\rightarrow_ T_i
• But not recoverable

• Need to make “final’ decision for each transaction:
  – commit decision - system guarantees transaction will or has completed, no matter what
  – abort decision - system guarantees transaction will or has been rolled back (has no effect)

To model this, two new actions:
• C_i - transaction T_i commits
• A_i - transaction T_i aborts
**Back to example:**

<table>
<thead>
<tr>
<th>Ti</th>
<th>Ti</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>Wj(A)</td>
<td>r(A)</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>Ci → can we commit here?</td>
<td></td>
</tr>
</tbody>
</table>

**Definition**

Ti reads from Tj in S (Tj ⇒S Ti) if

1. \( w_j(A) <_S r(A) \)
2. \( a_j <_S r(A) \) (\(<_S : \) does not precede)
3. If \( w_j(A) <_S w_k(A) <_S r(A) \) then \( a_k <_S r(A) \)

**Definition**

Schedule S is **recoverable** if whenever \( T_j \Rightarrow_S T_i \) and \( j \neq i \) and \( C_i \in S \) then \( C_j <_S C_i \)
Note: in transactions, reads and writes precede commit or abort

\[ \text{\textbullet} \quad \begin{align*}
\text{If } C_i \in T_i, \text{ then } r_i(A) &< C_i \\
&w_i(A) < C_i \\
\text{If } A_i \in T_i, \text{ then } r_i(A) &< A_i \\
&w_i(A) < A_i
\end{align*} \]

\[ \\text{\textbullet} \quad \text{Also, one of } C_i, A_i \text{ per transaction} \]

How to achieve recoverable schedules?

\[ \Rightarrow \text{With 2PL, hold write locks to commit (strict 2PL)} \]

\[
\begin{array}{c|c}
T_j & T_i \\
\hline
\vdots & \vdots \\
W_j(A) & \vdots \\
\vdots & \vdots \\
C_j & \vdots \\
u_j(A) & r_i(A)
\end{array}
\]
\[\Rightarrow\] With validation, no change!

- S is recoverable if each transaction commits only after all transactions from which it read have committed.

- S avoids cascading rollback if each transaction may read only those values written by committed transactions.

- S is strict if each transaction may read and write only items previously written by committed transactions.
Where are serializable schedules?

Examples

- Recoverable:
  - $w_1(A) w_1(B) w_2(A) r_2(B) c_1 c_2$
- Avoids Cascading Rollback:
  - $w_1(A) w_1(B) w_2(A) c_1 r_2(B) c_2$
- Strict:
  - $w_1(A) w_1(B) c_1 w_2(A) r_2(B) c_2$

Deadlocks

- Detection
  - Wait-for graph
- Prevention
  - Resource ordering
  - Timeout
  - Wait-die
  - Wound-wait
Deadlock Detection

- Build Wait-For graph
- Use lock table structures
- Build incrementally or periodically
- When cycle found, rollback victim

Resource Ordering

- Order all elements A₁, A₂, ..., Aₙ
- A transaction T can lock Aᵢ after Aⱼ only if i > j

Problem: Ordered lock requests not realistic in most cases

Timeout

- If transaction waits more than L sec., roll it back!
- Simple scheme
- Hard to select L
Wait-die

- Transactions given a timestamp when they arrive \( \ldots ts(T_i) \)
- \( T_i \) can only wait for \( T_j \) if \( ts(T_i)<ts(T_j) \)
  \( \ldots \) else die

Example:

\[ \begin{array}{c}
T_1 \\
(ts = 10)
\end{array} \quad \begin{array}{c}
\text{wait} \\
\text{wait}
\end{array} \quad \begin{array}{c}
T_2 \\
(ts = 20)
\end{array} \quad \begin{array}{c}
\text{wait} \\
\text{wait}
\end{array} \quad \begin{array}{c}
T_3 \\
(ts = 25)
\end{array} \]

Wound-wait

- Transactions given a timestamp when they arrive \( \ldots ts(T_i) \)
- \( T_i \) wounds \( T_j \) if \( ts(T_i)<ts(T_j) \)
  else \( T_i \) waits

"Wound": \( T_j \) rolls back and gives lock to \( T_i \)