Deadlock Control

Prevention and Detection

Deadlock

Transaction A
Read X
Read Y
Write X

Readlock Y

Cannot get lock due to B’s writelock -- A sleeps

Each sleeps, waiting for the other

Transaction B
Read Y
Read X
Write Y

Writelock Y

Readlock X

Cannot get lock due to A’s writelock -- B sleeps
Deadlock Prevention Strategy

• Change some rule about locks
• Normally: Can’t get lock ⇒ Sleep on Queue
• New rule: Can’t get lock ⇒ Some transaction may die
• Strategy: find something (X) that always happens when there is going to be a deadlock:
  If deadlock about to happen Then X is true
• Turn it around to make the rule:
  If X is true Then deadlock is about to happen
  And someone must die to prevent it.

Prevention Solutions

• Conservative 2PL
• Wait-Die Protocol
• Wound-Wait Protocol
• No Waiting
• Cautious Waiting
Conservative 2PL

- Saw this one earlier
- Get all needed locks at beginning
- If you cannot get them, die and retry

Wait-Die Protocol

- If you cannot get a lock, sometimes die instead of waiting
- Rule: Compare age of transaction who holds lock and transaction who wants it
  - Older Transaction will wait
  - Younger Transaction must die
Wait-Die Example

Transaction A
- Read X
- Write X
- Read Y
- Waitlock X
- Readlock Y
- Cannot get lock due to B's Writelock -- A is older: waits
- No Deadlock

Transaction B
- Read Y
- Writelock Y
- Read X
- Writelock X
- Readlock Y
- Cannot get lock due to A's Writelock -- B is younger: dies

Unnecessary Deaths

Transaction A
- Read X
- Readlock Y
- Write X
- Cannot get lock due to B's Writelock -- A is younger: dies
- No Deadlock would have occurred if A waited

Transaction B
- Read Y
- Readlock W
- Write Y
- Unlock Y
- Unlock W
Wound-Wait Protocol

• If you cannot get a lock, sometimes kill instead of waiting
• Rules:
  – Older Transaction kills younger
  – Younger Transaction will wait
• Death comes to younger lock holder like bolt from the blue
  – Not when younger does anything
  – When older tries for lock

Wound-Wait Example

<table>
<thead>
<tr>
<th>YOUNGER</th>
<th>OLDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction A</td>
<td>Transaction B</td>
</tr>
<tr>
<td>Writelock X</td>
<td>Writelock Y</td>
</tr>
<tr>
<td>Read X</td>
<td>Read Y</td>
</tr>
<tr>
<td>Readlock Y</td>
<td>Readlock X</td>
</tr>
<tr>
<td>Read Y</td>
<td>Cannot get lock due to B’s Writelock -- A is younger: waits</td>
</tr>
<tr>
<td>Write X</td>
<td>Cannot get lock due to A’s Writelock -- B is older: kills A</td>
</tr>
</tbody>
</table>

No Deadlock
No Waiting

• If you cannot get a lock, assume deadlock
  – Die
  – Wait
  – Restart
• Never a Queue for a lock

Cautious Waiting

• Not so rash as ‘No Waiting’
• If you cannot get a lock, check lock holder
  – If lock holder is already waiting, die and restart
  – If lock holder is active, wait
## Cautious Waiting Example

<table>
<thead>
<tr>
<th>Transaction A</th>
<th>Transaction B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read X</td>
<td>Writelock Y</td>
</tr>
<tr>
<td>Writelock X</td>
<td>Read Y</td>
</tr>
<tr>
<td>Readlock Y</td>
<td>Readlock X</td>
</tr>
<tr>
<td>Read Y</td>
<td>Write Y</td>
</tr>
<tr>
<td>Write X</td>
<td>Cannot get lock due to B’s Writelock -- B is active: A waits</td>
</tr>
<tr>
<td>No Deadlock</td>
<td>Cannot get lock due to A’s Writelock -- A is sleeping: B dies</td>
</tr>
</tbody>
</table>

## Deadlock Detection

- Use locks with no deadlock prevention
- Every so often, check for deadlock
  - use Wait-for Graph
- If found, kill some transaction to break deadlock
No deadlock

Cost-Benefit of Detection

• No unnecessary slaughter
• Transactions sit in deadlock for a while instead of restarting
• Time taken to calculate wait-for graph every so often