Deadlock Prevention

84

- When there is a high level of lock contention and an increased likelihood of deadlocks
- Prevent deadlocks by giving each Xact a priority
- Assign priorities based on timestamps.
  - Lower timestamp indicates higher priority
  - i.e., oldest transaction has the highest priority
  - Higher priority Xacts cannot wait for lower priority Xacts (or vice versa)

84

Deadlock Prevention (cont'd)

85

- Assume Ti wants a lock that Tj holds. Two policies are possible:
  - **Wait-Die:** If Ti has higher priority, Ti waits for Tj; otherwise Ti aborts
  - **Wound-wait:** If Ti has higher priority, Tj aborts; otherwise Ti waits
- Both schemes will cause aborts even though deadlock may not have occurred.
- If a transaction re-starts, make sure it has its original timestamp – WHY?

85

Wait-Die

86

- Suppose Ti tries to lock an item already locked by Tj.
- If Ti is the older transaction then Ti will wait
- Otherwise Ti is aborted and re-starts later with the same timestamp.
- Lower priority transactions never wait for higher priority transactions.

86

Example: Wait-Die

87

T1 starts before T2, so T1 is higher priority

T1	**T2**
S(A)	
R(A)	
	S(B)
	R(B)
X(B)	
	X(A)
	Abort
Resume	

T1 is older and so it is allowed to wait.

T2 is younger, and is aborted, which results in its locks being released. This allows T1 to proceed.

87

Wound-Wait

- Suppose Ti tries to lock an item locked by Tj.
- If Ti is higher priority (older transaction)
  - then Tj is aborted and restarts later with the same timestamp;
- Otherwise, Ti waits.

Example: Wound-Wait

T1 starts before T2, so T1 is higher priority

T1	**T2**
S(A)	
R(A)	
	S(B)
	R(B)
X(B)	
	Abort

T1 is older, so T2 is aborted, and that allows T1 to proceed.

Wait-Die: Let's consider…

T1 (ts =22) requests A: wait for T_2 or T_3 or both?

Note: ts between 20 and 25.

T2 (ts =20)

wait(A)

T3 (ts =25)

Wait-Die: Option 1

One option: T_1 waits just for T_3, transaction holding lock.

But when T_2 gets lock, T_1 will have to die!

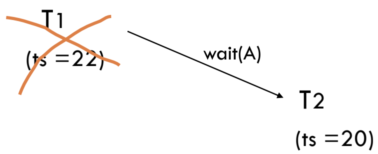
T2

(ts =20)

Wait-Die: Option 2

92

Another option: T_1 waits for both T_2, T_3.
T_1 allowed to wait iff there is at least one younger Xact waiting for A.
But again, when T2 gets lock, T_1 must die!

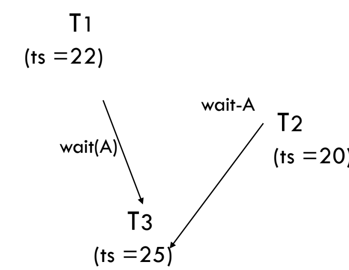


92

Wait-Die: Option 3

93

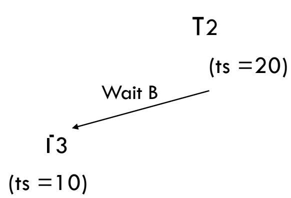
- Yet another option: T_1 preempts T_2 (T_2 is just waiting idle anyway),
- So T_1 only waits for T_3; T_2 then waits for T_1
- And lots of WFG work for Deadlock Manager



93

Wound-Wait

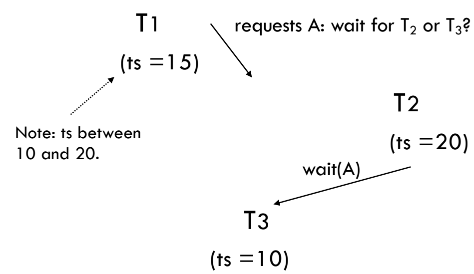
94



94

Wound-Wait: Let's Consider …

95



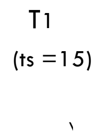
95

Wound-Wait: Option 1

96

One option: T_1 waits just for T_3, transaction holding lock.

But when T_2 gets lock, T_1 waits for T_2 and wounds T_2.



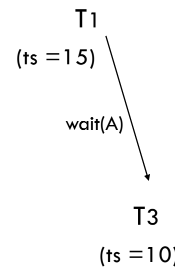
96

Wound-Wait: Option 2

97

Another option:

T_1 waits for both T_2, T_3 $\Longrightarrow$ T_2 wounded right away!



97

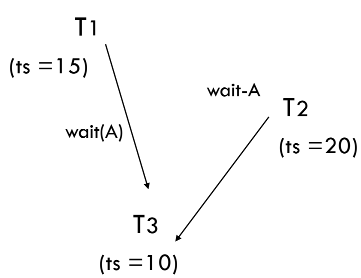
Wound-Wait: Option 3

98

Yet another option: T_1 preempts T_2, so T_1 only waits for T_3;

T_2 then waits for T_3 and T_1... $\Longrightarrow$ T_2 is spared!

Lots of WFG work for Deadlock Mgr (shifting edges)



98

Comparing Deadlock Management Schemes

99

- Wait-die and Wound-wait ensure no starvation (unlike detection)
- Waits-for graph technique only aborts transactions if there really is a deadlock

99

Summary

100

- There are several lock-based concurrency control schemes (Strict 2PL, 2PL).
- SQL-92 provides different isolation levels that control the degree of concurrency
- The lock manager keeps track of the locks issued. Deadlocks can either be prevented or detected.

100