COSC 460 Lecture 17: Transactions 4: Deadlock

Professor Michael Hay

Fall 2018

Review: Two phase locking

- **2PL**: a transaction cannot acquire additional locks once it has released any lock
  - Growing phase (acquiring locks)
  - Shrinking phase (releasing locks)
- 2PL guarantees ***conflict serializability***

Deadlock

- T1 transfers money from B to A.
- T2 transfers money from A to B.

T1	T2
L(A)	
	L(B)
A=A+100	
	B=B+50
	L(A)
L(B)	A=A-50
B=B-100	U(A),U(B)
U(A), U(B)	

Deadlock! (Grayed out events never happen)

Strategies for deadlock

- **Timeout**: if T_i waiting for "long" time, abort.
  - Hard to tune. What is right amount to wait?
- **Prevention**: preemptively abort transactions in situations that *could* lead to deadlock
  - Conservative: more aborts than necessary.
- **Detection**: with each new lock request, check whether it creates deadlock.
  - Expensive: adds overhead to every request.

Prevention

- Each T_i assigned timestamp. *Older* transactions given *higher priority*.
- Suppose T_i requests lock held by T_j
- Options available to requestor: wait, abort, force holder to abort.
- Behavior depends on…
  - Who is older? Requestor or holder?
  - Policy: wait-die vs. wound-wait.

Prevention policies

- **Wait-die**: if requestor is older than holder, it wait; else it aborts.
- **Wound-wait**: if requestor is older than holder, it "wounds" (aborts) holder; else it waits.

Deadlock Prevention

- Assume txn number acts as timestamp (T1 is older)

- Under wait-die, what happens?

T1	T2
L(A)	
	L(B)
A=A+100	
	B=B+50
	L(A)
L(B)	A=A-50
B=B-100	U(A),U(B)
U(A), U(B)	

T2 is requestor for L(A) and T1 is holder. Under wait-die, T2 aborts.

Deadlock Prevention

Instructions: ~1 minute to think/answer on your own; then discuss with neighbors; then I will call on one of you

- Assume txn number acts as timestamp (T1 is older)
- Under wound-wait, what happens?

T1	T2
L(A)	
	L(B)
A=A+100	
	B=B+50
	L(A)
L(B)	A=A-50
B=B-100	U(A),U(B)
U(A), U(B)	

T2 waits on T1 for A. T1 is requestor for L(B) and T2 is holder. Under wound-wait, T1 kills T2.

Example

Instructions: ~1 minute to think/answer on your own; then discuss with neighbors; then I will call on one of you

- With deadlock prevention schemes, if txn is aborted and restarts, it does not get a new timestamp. Instead it keeps its old timestamp. Why is this important?

Detection

- "Waits for" graph
  - Nodes: running or waiting transactions T_1 , …, T_n
  - Edge: $T_i \rightarrow T_j$ if T_i is *waiting for* a lock held by T_j.
- If graph has a *cycle*, then there is deadlock.

Deadlock Detection

Instructions: ~1 minute to think/ answer on your own; then discuss with neighbors; then I will call on one of you

- Draw the waits-for graph for this schedule.
- Is there a deadlock?
- If so, which txn should be aborted?

T1	**T2**	**T3**
S(A)		
	X(B)	
S(B)		
|		S(C)
|	X(C)	
|	|	X(A)
		|

Yes. There is a cycle:
T3->T1->T2->T3.
Several criteria can be used to decide which to abort: youngest, fewest locks, least work, etc.

Aborts, atomicity and consistency

- When a transaction T_i aborts, we must undo any changes T_i made to database.
  - Goal: make it look as if transaction never happened (atomicity)
- How might an abort of T_i affect other transactions T_j? (Example on next slide.)

Aborts and Consistency

- Assume T_i number is timestamp (T1 is older)
- Under wound-wait, what happens when T1 requests lock on C?

T1	T2	T3
L(A)		
	L(B), L(C)	
A=A+100		
	B=B+50	
	U(B)	
		L(B)
		R(B)
L(C)		
C=C-100	C=C-50	
U(A)	U(C)	
U(C)		

T1 wounds T2, causing it to abort. **But T3 read data written by T2!!** (T3 should be aborted, too.)

Schedules

- **Recoverable schedule**: if T_j reads value written by T_i, then T_i commits before T_j *commits*.
- **Cascade-less schedule**: if T_j reads value written by T_i, then T_i commits before T_j *reads*.

Recoverability and Cascading Aborts

- Is this schedule recoverable?
- Is it cascade-less?

It is not **recoverable**. What if T1 aborts after reading B?

T1	T2
X(A), X(B)	
A=A+100	
U(A)	
	X(A)
	A=A*1.01
	U(A)
	Commit
R(B)	

Recoverability and Cascading Aborts

- Is this schedule recoverable?
- Is it cascade-less?

It is not **cascade-less**. What if T1 aborts after reading B?

T1	T2	T3
X(A), X(B)		
A=A+100		
U(A)		
	X(A)	
	A=A*1.01	
	U(A)	
R(B)		S(A)
Commit		R(A)
	Commit	
		Commit

Strict 2PL

- **Strict 2PL**: 2PL protocol with additional requirement that all locks are release when the transaction is completed.

- If all transactions follow Strict 2PL, then all schedules will be…

    - *Conflict-serializable*, and

    - *Cascade-less*