EECS 484

Homework 5 Solutions

Winter 2015

Part 1: Serializability

Question 1:

- Consider the following (incomplete) schedule involving transactions T1, T2, T3 and T4. Is the schedule recoverable? Explain.

T1	**T2**	**T3**	T4
R(X)			
W(X)			
			R(Z)
	R(Y)		
			W(Z)
	W(Y)		
		R(Z)	
		COMMIT	
			COMMIT

Solution:

Consider the following (incomplete) schedule involving transactions T1, T2, T3 and T4. Is the schedule recoverable? Explain.

T1	T2	T3	T4
R(X)			
W(X)			
			R(Z)
	R(Y)		
			W(Z)
	W(Y)		
		R(Z)	
		COMMIT	
			COMMIT

Solution: No, the schedule is not recoverable. T3 reads T4's changes, but commits before T4. A schedule is recoverable if transactions commit only after all transactions whose changes they read commit

Solution:

- Consider the following (incomplete) schedule involving transactions T1, T2, T3 and T4. Does it avoid cascading aborts. Explain.

T1	**T2**	**T3**	T4
R(X)			
W(X)			
			R(Z)
	R(Y)		
			W(Z)
	W(Y)		
		R(Z)	
		COMMIT	
			COMMIT

Solution:

- Consider the following (incomplete) schedule involving transactions T1, T2, T3 and T4. Does it avoid cascading aborts. Explain.

T1	T2	T3	T4
R(X)			
W(X)			
			R(Z)
	R(Y)		
			W(Z)
	W(Y)		
		R(Z)	
		COMMIT	
			COMMIT

Solution: No, the schedule does not avoid cascading aborts. T3 reads T4's changes, but commits before T4. The schedule avoids cascading aborts, if the transactions only read the changes of committed transactions

Solution:

- Consider the following (incomplete) schedule involving transactions T1, T2, T3 and T4. Could it be generated by non-strict 2PL?

T1	**T2**	**T3**	T4
R(X)			
W(X)			
			R(Z)
	R(Y)		
			W(Z)
	W(Y)		
		R(Z)	
		COMMIT	
			COMMIT

Solution:

- Consider the following (incomplete) schedule involving transactions T1, T2, T3 and T4. Could it be generated by non-strict 2PL?

T1	T2	T3	T4
R(X)			
W(X)			
			R(Z)
	R(Y)		
			W(Z)
	W(Y)		
		R(Z)	
		COMMIT	
			COMMIT

- **Solution**:Yes. The only conflicting operation is T3 R(Z) following T4's W(Z) operation. Since, T4 is not modifying Z after this, it will release the lock in non-strict 2 PL.

Solution:

- Consider the following (incomplete) schedule involving transactions T1, T2, T3 and T4. Could it be generated by strict 2PL?

T1	**T2**	**T3**	T4
R(X)			
W(X)			
			R(Z)
	R(Y)		
			W(Z)
	W(Y)		
		R(Z)	
		COMMIT	
			COMMIT

Solution:

- Consider the following (incomplete) schedule involving transactions T1, T2, T3 and T4. Could it be generated by strict 2PL?

T1	T2	T3	T4
R(X)			
W(X)			
			R(Z)
	R(Y)		
			W(Z)
	W(Y)		
		R(Z)	
		COMMIT	
			COMMIT

- **Solution**:No.The only conflicting operation is T3 R(Z) following T4's W(Z) operation. However, in Strict 2PL, T4 will not release the lock on Z until T4 commits.

Solution:

- Consider the following (incomplete) schedule involving transactions T1, T2, and T3, T4. Is it conflict-serilizable?

T1	**T2**	**T3**	T4
R(X)			
W(X)			
			R(Z)
	R(Y)		
			W(Z)
	W(Y)		
		R(Z)	
		COMMIT	
			COMMIT

Solution:

- Consider the following (incomplete) schedule involving transactions T1, T2, and T3, T4. Is it conflict-seriliazable?

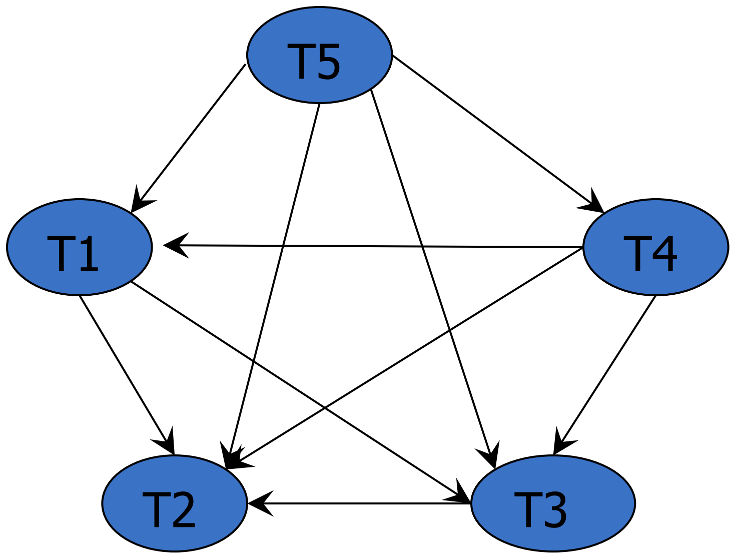
T1	**T2**	**T3**	T4
R(X)			
W(X)			
			R(Z)
	R(Y)		
			W(Z)
	W(Y)		
		R(Z)	
		COMMIT	
			COMMIT

Solution: Yes. The precedence graph has only one edge from T4 to T3. Since, there is no cycle in the precedence graph, the schedule is conflict-serializable.

Solution – Question 2

Time	T1	T2	T3	T4	T5
0					W(Z)
1				W(Z)	
2	R(Z)				
3			W(Z)		
4			COMMIT		R(Y)
5					COMMIT
6				R(Y)	
7				COMMIT	
8		R(Y)			
9	R(Y)				
10	R(X)				
11	COMMIT				
12		R(X)			
13		W(Y)			
14		R(Z)			
15		COMMIT			

Solution – Question 2



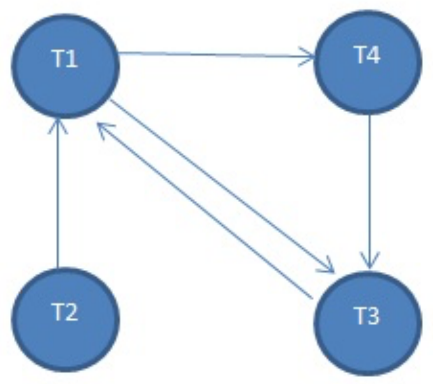
10 edges in the precedence graph
No cycle
It is conflict seriliazible

Part 1- Question 3

- Given the following schedules of read and write by different transactions, draw the precedence graph and determine whether the schedule is conflict-serializable.
- What are the nodes involved in a cycle, if there is one.?

Time	T1	T2	T3	T4
0	W(X)			
1	R(X)			
2			R(X)	
3		R(Z)		
4	R(X)			
5				R(X)
6		W(Z)		
7		COMMIT		
8			W(X)	
9			COMMIT	
10	R(X)			
11				W(Y)
12				COMMIT
13	R(Z)			
14	COMMIT			

Solution – Question 3



The graph has 5 edge
A cycle exists between T1 and T3, so this schedule is not serializable.

Part 2 - Locks

- Give a schedule that **strict 2PL** would generate for Q1(b), assuming that locks on a resource are granted in order of arrival. If you run into a deadlock, show the final state as a graph.
- Assume the following:
  - All transactions arrive at time 0, but in the order T1, T3, T2, and T4.
  - Any transactions that are eligible to run (not waiting for locks) run in round-robin fashion. In other words, each transaction does one step (either R or W operation) goes to the end of the runnable queue, and then cedes to the next transaction that is eligible to run. If a transaction blocks on a lock request, it goes to the end of the corresponding lock queue.

Question 2

- More assumptions
  - Getting a lock and using the resource is done in one operation. COMMIT is a separate operation and releases all locks.
  - A lock is released immediately on COMMIT or ABORT and immediately transferred to the first waiting transaction. If a write lock is released and a reader is the first one in the queue, all waiting readers are granted the lock, irrespective of their positions in the queue. They become runnable and put at the end of the runnable queue.
  - The system rotates among waiting transactions

Solution 2

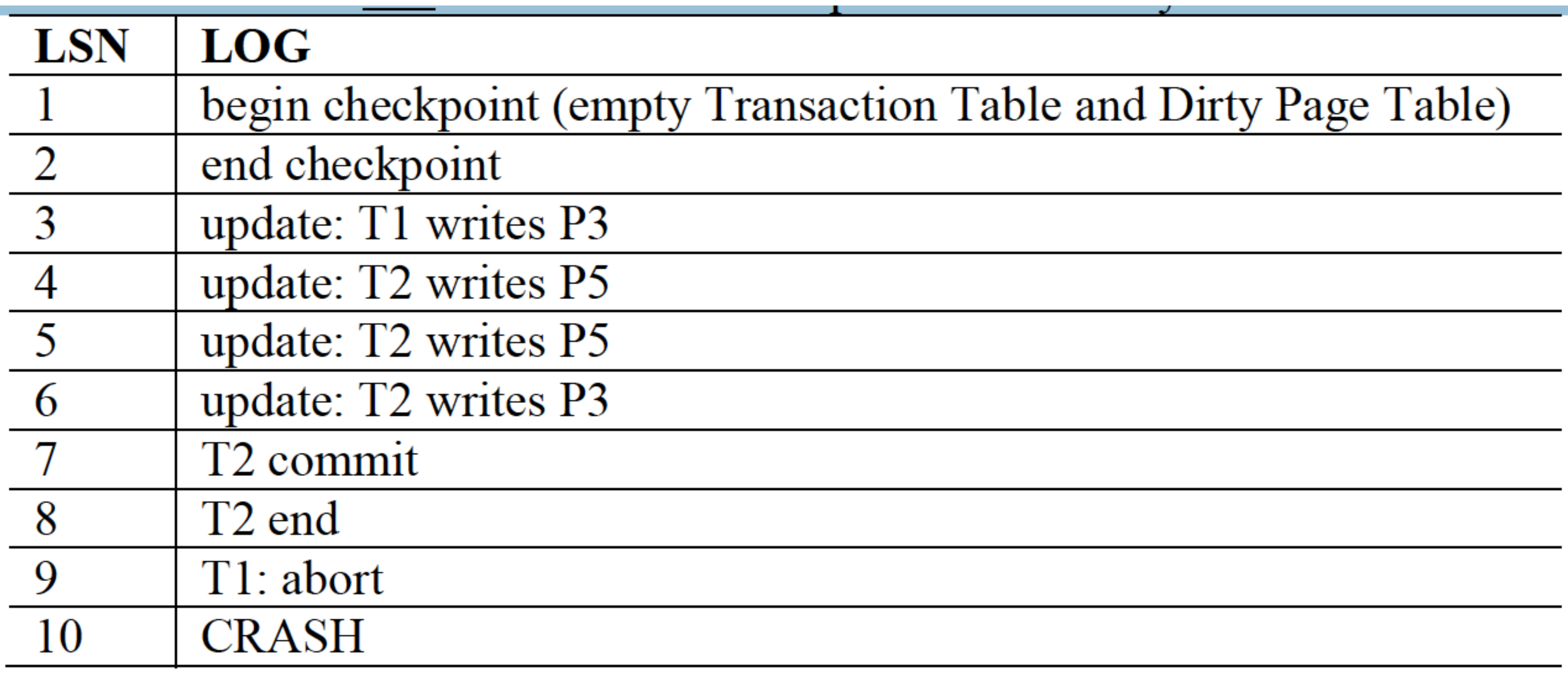
- Time 0: T1 does WL(X) and W(X)
- Time 1: T3 queues up. T2 does SL(Z) and R(Z)
- Time 2: T4 does SL(Z) and R(Z)
- Time 3: T1 does R(X)
- Time 4: T2 queues up. T4 does WL(Y) and W(Y).
- Time 5: T1 does R(X)
- Time 6: T4 commits and releases locks on Z and Y. T2 is released
- Time 7: T1 does R(X)
- Time 8: T2 does WL(Z) and W(Z)
- Time 9: T1 blocks on R(Z). T2 commits and releases lock on Z

Solution 2

- Time 10: T1 does SL(Z) and R(Z)
- Time 11: T1 does Commit. T1 releases locks on Z and X
- Time 12: T3 does SL(X) and R(X)
- Time 13: T3 does WL(X) and W(X)
- Time 14: T3 commits. T3 releases lock on X.

Part 3 - Recovery

- Consider the following log and the ARIES recovery protocol



LSN	LOG
1	begin checkpoint (empty Transaction Table and Dirty Page Table)
2	end checkpoint
3	update: T1 writes P3
4	update: T2 writes P5
5	update: T2 writes P5
6	update: T2 writes P3
7	T2 commit
8	T2 end
9	T1: abort
10	CRASH

Part 3 - Recovery

- Draw the Dirty Page table and the Transaction Table at the end of the ANALYSIS phase of recovery.

- Transaction table:

XID	Last LSN	Status
T1	9 (Alternate answer: 3)	Aborted

- T2 is not in transaction table because it is removed on end record
- D.P.T

Page ID	recLSN
P3	3
P5	4

Part 3 - Recovery

- Which transactions are loser transactions?

- Transaction table:

XID	Last LSN	Status
T1	9	Aborted

Answer: T1

Part 3 - Recovery

- Are any new compensation log records (CLRs) written during the REDO phase of recovery? Give reason.

- No, REDO is just reapply logged action, no additional logging.
- CLR is only added in UNDO phase if there is a need to undo an update

Part 3 - Recovery

- How many CLR records are added in UNDO phase? Briefly describe them.

- 1, CLR for undoing T1’s update on P3.