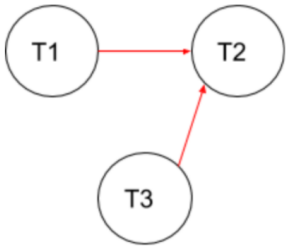
CS 186 Fall 2024 | Introduction to Database Systems Alvin Cheung | DIS 8

1 Conflict Serializability

T1		R(A)	W(A)	R(B)					
T2					W(B)	R(C)	W(C)	W(A)	
T3	R(C)								W(D)

(a) Draw the dependency graph (precedence graph) for the schedule. Assume all transactions commit at the end of the schedule.

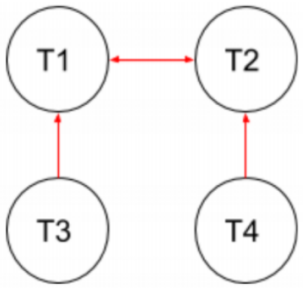


(b) Is this schedule conflict serializable? If so, what are all the conflict equivalent serial schedules? If not, why not?

Yes. T3, T1, T2 and T1, T3, T2. Topologically sorting the above graph gives these schedules.

T1	R(A)		R(B)				W(A)	
T2		R(A)		R(B)				W(B)
T3					R(A)			
T4						R(B)		

(c) Draw the dependency graph (precedence graph) for the schedule.



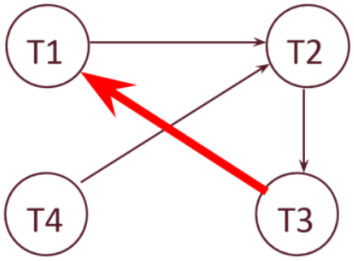
(d) Is this schedule conflict serializable? If so, what are all the conflict equivalent serial schedules? If not, why not?

No, there's a cycle between T1 and T2: T1 must come before T2 and T2, before T1.

2 Deadlocks

T1	S(A)	S(D)		S(B)					
T2			X(B)				X(C)		
T3					S(D)	S(C)			X(A)
T4								X(B)	

(a) Draw a "waits-for" graph at the end of the schedule and state whether or not there is a deadlock. Assume all transactions attempt to commit at the end of the schedule.



Yes, there is a deadlock. There is a cycle between T1, T2, and T3.

(b) If we try to avoid deadlock by using wait-die deadlock avoidance policy, would any transactions be aborted? Assume T1 priority > T2 > T3 > T4.

Yes, T3 and T4 are aborted. When T4 attempts to acquire a lock on B, which is held by T2, T4 will abort since it is attempting to acquire a lock held by a transaction with higher priority. The same thing happens when T3 attempts to acquire a lock on A, which is held by T1.

3 Locking

T1	T2
Lock_X(B)	
Read(B)	
B := B * 10	
Write(B)	
Lock_X(F)	
Unlock(B)	
	Lock_S(F)
F := B * 100	
Write(F)	
Commit	
Unlock(F)	
	Read(F)
	Unlock(F)
	Lock_S(B)
	Read(B)
	Print(F + B)
	Commit
	Unlock(B)

(a) What is printed, assuming we initially have B = 3 and F = 300?
3030

(b) Does the execution use 2PL or strict 2PL?
Neither - S(F) unlocked before T2 acquires S(B)

(c) Would moving Unlock(F) in the second transaction to any point after Lock_S(B) change this (or keep it) in 2PL?
Yes - all locks would be acquired (for T2) before any are released.

(d) Would moving Unlock(F) in the first transaction and Unlock(F) in the second transaction to the end of their respective transactions change this (or keep it) in strict 2PL?
No - T1 still unlocks B before the end of the transaction

(e) Would moving Unlock(B) in the first transaction and Unlock(F) in the second transaction to the end of their respective transactions change this (or keep it) in strict 2PL?
Yes - all unlocks would only happen when the respective transactions end

4 Multigranularity Locking

(a) Suppose a transaction T1 wants to scan a table R and update a few of its tuples. What kinds of locks should T1 have on R, the pages of R, and the updated tuples?

1. Obtain SIX on R
2. Obtain IX on Page [We don't obtain a SIX because there is already an S lock on R (from the SIX). Obtaining another S on the Page is redundant.]
3. Obtain X on Tuples being modified

(b) Is an S lock compatible with an IX lock?

Suppose T1 wants an S lock on an object, O, and T2 wants an IX lock on the same object O. An S lock implies that T1 will read the entire object (all of its sub-objects). An IX lock implies that T2 will write some of the sub-objects of the object. This means that there is some sub-object of O that T1 will read and T2 will write. This is not valid, so the S and IX locks must be incompatible.

(c) Consider a table which contains two pages with three tuples each, with Page 1 containing Tuples 1, 2, and 3, and Page 2 containing Tuples 4, 5, and 6.

i. Given that a transaction T1 has an IX lock on the table, an IX lock on Page 1, and an X lock on Tuple 1, which locks could be granted to a second transaction T2 for Tuple 2?

X or S locks

ii. Given that a transaction T1 has an IS lock on the table and an S lock on Page 1, what locks could be granted to a second transaction T2 for Page 1?

S or IS locks