DATABASE SYSTEMS

CS – 355/CE – 373

Instructor: Maria N. Samad

November 11th, 2024

DEADLOCKS VS INCONSISTENCY

- If we do not use locking, or if we unlock data items too soon after reading or writing them, we may get inconsistent states.
- On the other hand, if we do not unlock a data item before requesting a lock on another data item, deadlocks may occur.
- Both are undesirable, however - ***deadlocks are definitely preferable to inconsistent states***, since they can be handled by rolling back transactions,
- Inconsistent states may lead to real-world problems that cannot be handled by the database system.

STARVATION

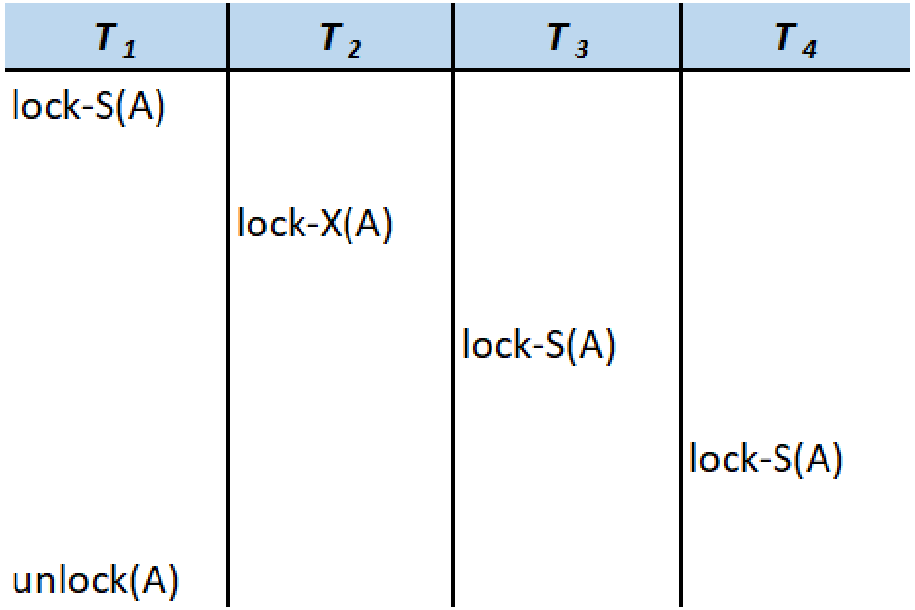
- Suppose a transaction T_2 has a shared-mode lock on a data item, and another transaction T_1 requests an exclusive-mode lock on the data item.
- T_1 has to wait for T_2 to release the shared mode lock.
- Meanwhile, a transaction T_3 may request a shared-mode lock on the same data item.
- The lock request is compatible with the lock granted to T_2, so T_3 may be granted the shared-mode lock.
- At this point T_2 may release the lock, but still T_1 has to wait for T_3 to finish.

STARVATION

- In fact, it is possible that there is a sequence of transactions that each requests a shared mode lock on the data item,
- And each transaction releases the lock a short while after it is granted, but T_1 never gets the exclusive-mode lock on the data item.
- The transaction T_1 may never make progress, and is said to be ***starved*.**

STARVATION

- Example:



T_1	T_2	T_3	T_4
lock-S(A)			
	lock-X(A)		
		lock-S(A)	
			lock-S(A)
unlock(A)			

- Here, T_2 may end up starving

AVOIDING STARVATION

- We can avoid starvation of transactions by granting locks in the following manner:
- When a transaction T_i requests a lock on a data item Q in a particular mode M, the concurrency-control manager grants the lock provided that:
  1. ***There is no other transaction holding a lock on Q in a mode that conflicts with M***
  2. ***There is no other transaction that is waiting for a lock on Q and that made its lock request before*** T_i
- Thus, a lock request will never get blocked by a lock request that is made later.

TWO-PHASE LOCKING PROTOCOL

- One protocol that ensures serializability is the two-phase locking protocol.
  - Also called 2PL
- This protocol requires that each transaction issue lock and unlock requests in two phases:
  - ***Growing phase:*** A transaction may obtain locks, but may not release any lock.
  - ***Shrinking phase:*** A transaction may release locks, but may not obtain any new locks.
- Initially, a transaction is in the growing phase.
- The transaction acquires locks as needed.
- Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests

TWO-PHASE LOCKING PROTOCOL

- It is **not** mandatory that all the unlocks happen together at the end of transaction.
- As long as there is not going to be any further acquiring of the locks, i.e. growing phase, the unlocks can happen even before the partially committed state
- **PLEASE NOTE**:
  - The locks are directly acquired in exclusive mode if there is write operation in the future instructions as well. This is done to avoid deadlocks
  - The locks must be released in the same order as they were acquired in Two-phase protocol

TWO-PHASE LOCKING PROTOCOL

- Example 1:
  - For the given schedule, get the 2PL equivalent schedule if possible:
    - S: r1(A), w1(A), r2(A), r3(A), w2(A), w1(B), w3(A), w2(B), C1, r2(B), C2, r3(B), C3
  - Is the resultant schedule consistent?

TWO-PHASE LOCKING PROTOCOL

- Solution 1:
  - On board

T1	**T2**	**T3**
r(A)		
w(A)		
	r(A)	
		r(A)
	w(A)	
w(B)		
		w(A)
	w(B)	
Commit		
	r(B)	
	Commit	
		r(B)
		Commit

TWO-PHASE LOCKING PROTOCOL

- Example 2:
  - For the given schedule, get the 2PL equivalent schedule if possible:
    - S: w1(A), w2(A), w1(B), w2(B), C2, C1
  - Is the resultant schedule consistent?

TWO-PHASE LOCKING PROTOCOL

- Solution 2:
  - On board

T1	**T2**
w(A)	
	w(A)
w(B)	
	w(B)
	Commit
Commit	

TWO-PHASE LOCKING PROTOCOL

- Example 3:
  - For the given schedule, get the 2PL equivalent schedule if possible:
    - S: w1(A), w3(A), C3, w2(B), C2, w1(B), C1
  - Is the resultant schedule consistent?

TWO-PHASE LOCKING PROTOCOL

- Solution 3:
  - On board

T1	**T2**	**T3**
w(A)		
		w(A)
		Commit
	w(B)	
	Commit	
w(B)		
Commit		

TWO-PHASE LOCKING PROTOCOL

- Activity Sheet

TWO-PHASE LOCKING PROTOCOL

- Activity Sheet Solution:
  - 2PL Solution