CS186 Discussion 8

(Transactions, Concurrency Control)

Matthew Deng

Transactions

Transactions

- DBMS abstract view of program or activity
- Sequence of reads and writes
- All actions must commit or abort entirely
- Has ACID properties

ACID

- Atomicity
  - All or none
- Consistency
  - Stay consistent
- Isolation
  - Isolated from other transactions
- Durability
  - Commit effects persist

Atomicity and Durability

- Transactions are committed or aborted
- Committed transactions are permanent

Consistency

- A transaction will bring one consistent state to another
- Transactions must satisfy integrity constraints

Isolation (Concurrency)

- Actions of different transactions do not interfere
- Each transaction executes as if ran by itself

Serializability

- Serial schedule
  - Transactions are run one at a time, with no intervention
- Equivalence
  - Same transactions with same actions
  - Leave DB in same state
- Serializable
  - Equivalent to a serial schedule

Conflict Serializability

- Conflict
  - Same object
  - Different transactions
  - At least one is a write

- Conflict Equivalent
  - Same transactions with same actions
  - Conflicts are in the same order

- Conflict Serializable
  - Conflict equivalent to a serial schedule

{conflict serializable schedules} $\subseteq$ {serializable schedules}

Dependency Graph

- Node
  - Transaction

- Directed edge
  - O_i of T_i conflicts with O_j of T_j, and O_i appears earlier than O_j
  - (T_i, T_j) if T_j depends on T_i

Schedule is conflict serializable if and only if its dependency graph is acyclic.

Transactions Exercises Worksheet #1, 2

Transaction Exercises

1. For each letter in ACID, what property does it stand for and what does it mean? What are the methods we employ to ensure that each of these properties are held?

   A:

   C:

   I:

   D:

Transaction Exercises

1. For each letter in ACID, what property does it stand for and what does it mean? What are the methods we employ to ensure that each of these properties are held?

   A: Atomicity - either all Xacts happen, or none. Enforced through logging.

   C: Consistency - if a DB begins as consistent, it is still consistent after a Xact. Enforced through integrity constraints.

   I: Isolation - Execution of a Xact is isolated from other Xacts. Enforced through serializability and locks.

   D: Durability - If a Xact commits, its effects persist. Enforced through logging.

Transaction Exercises

2. Consider the following schedules:

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.

Transaction Exercises

2. Consider the following schedules:

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.

T3 -> T2 [(R(C), W(C)];

T1 -> T2 [R(A), W(A) and W(A),W(A) and R(B),W(B)]

Transaction Exercises

2. Consider the following schedules:

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

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

Transaction Exercises

2. Consider the following schedules:

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

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

Yes. Serial schedules: T3, T1, T2; T1, T3, T2.

Transaction Exercises

2. Consider the following 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.

Transaction Exercises

2. Consider the following 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.

T3 -> T1 [R(A), W(A)];

T2 -> T1 [R(A), W(A)];

T1 -> T2 [R(B), W(B)];

T4 -> T2 [R(B), W(B)]

Transaction Exercises

2. Consider the following schedules:

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

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

Transaction Exercises

2. Consider the following schedules:

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

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

No. Why not: cycle in the precedence graph (T1 must precede T2, T2 must precede T1)

Concurrency Control

Locks

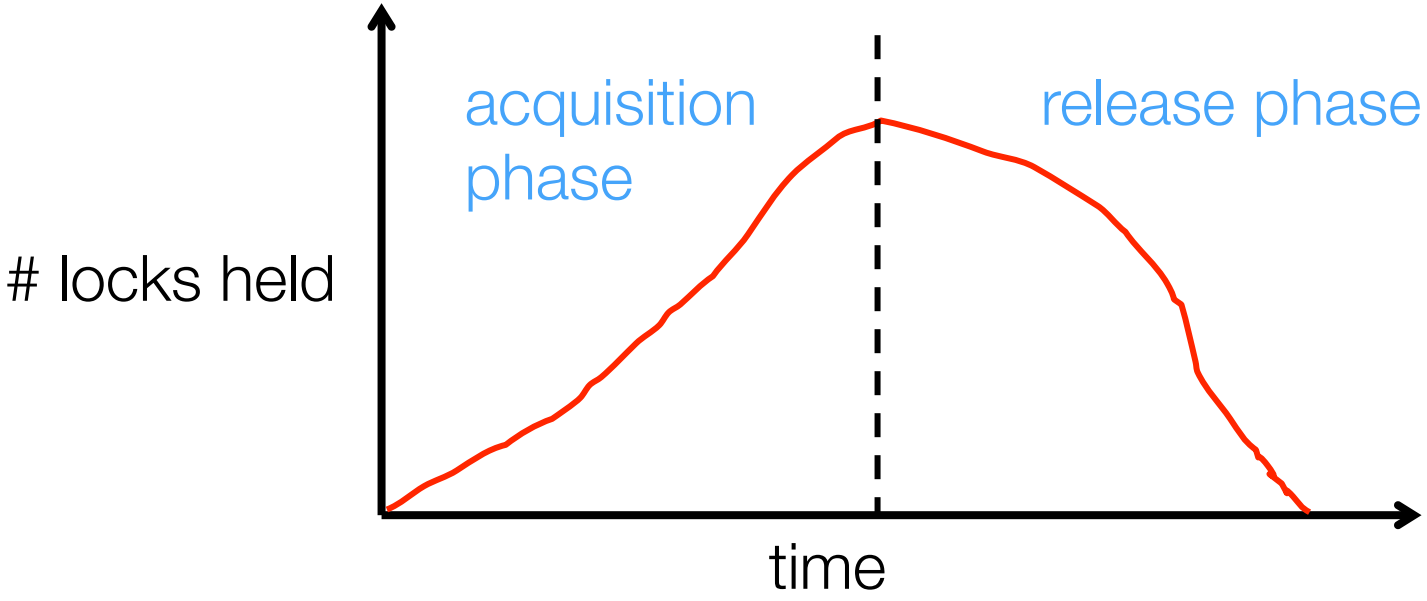
- S (shared lock) for reading
- X (exclusive lock) for writing

	S	X
S	√	–
X	–	–

Lock Compatibility Matrix

Two-Phase Locking (2PL)

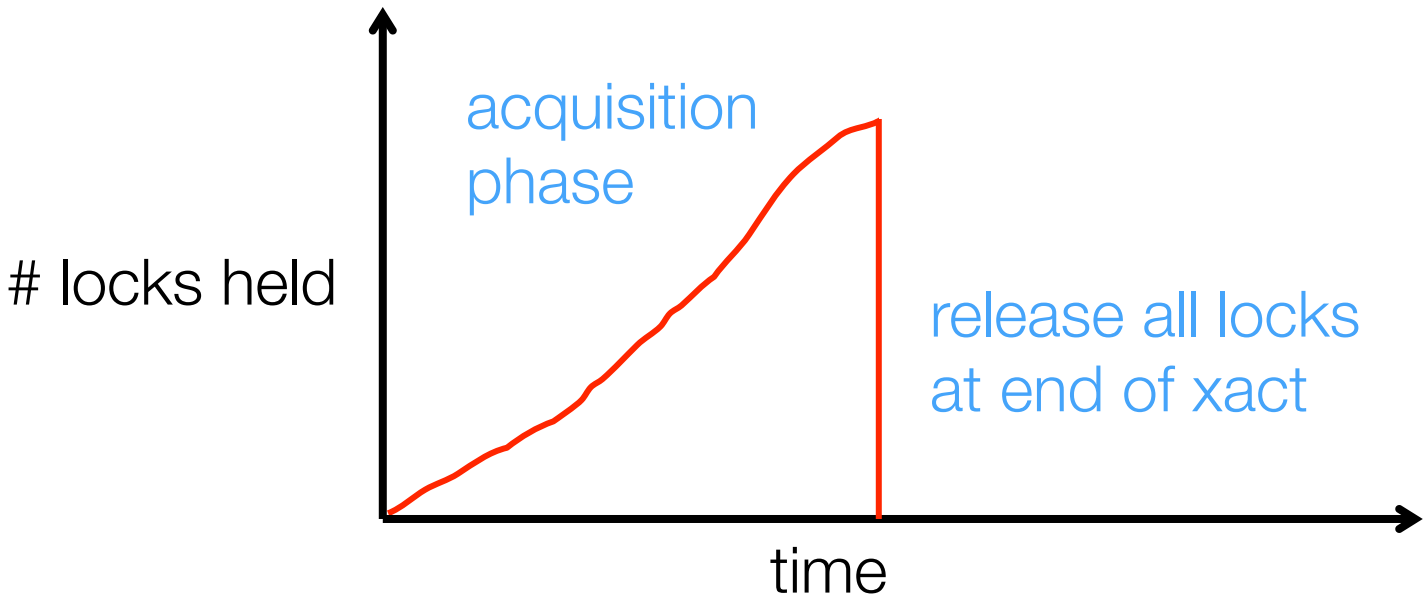
- Transaction cannot get new locks after releasing any locks
- Most common scheme for enforcing conflict serializability



- Guarantees conflict serializability
- Does not prevent cascading aborts

Strict 2PL

- Transaction releases all locks when it commits or aborts



- Guarantees conflict serializability
- prevents cascading aborts

Deadlock Detection

- Waits-For Graph
- Node
  - Transaction
- Directed edge
  - O_i of T_i conflicts with O_j of T_j, and O_j appears earlier than O_i
  - (T_i, T_j) if T_i waits for T_j

Concurrency Control Worksheet #3, 4

Concurrency Control Exercises

3. (a) What will be printed in the following execution (B=3, F=300)?

Concurrency Control Exercises

3. (a) What will be printed in the following execution (B=3, F=300)?

330

Concurrency Control Exercises

3. (b) Does the execution use 2PL or Strict 2PL?

Concurrency Control Exercises

3. (b) Does the execution use 2PL or Strict 2PL?

Neither, because transaction 2 unlocks F before locking B. In 2PL, a transaction cannot get any new locks after another lock has been released.

Concurrency Control Exercises

3. (c) How would you change the above execution to use 2PL? How would you change it to use Strict 2PL? Does the output change?

Concurrency Control Exercises

3. (c) How would you change the above execution to use 2PL? How would you change it to use Strict 2PL? Does the output change?

2PL: You could switch Unlock(F) and Lock_S(B) in transaction 2. However, this will result in deadlock, since transaction 1 will wait for a lock on F, while transaction 2 will wait for a lock on B.

Strict 2PL: You could move transaction 2's Unlock(F) to the very end of the execution. If the locks are unlocked after the transaction has committed/aborted, it is Strict 2PL. With the current timing of executions, this would also result in a deadlock.

A solution that would not result in deadlock would be if transaction 2 did not try to get a lock on F until after transaction 1 got its exclusive lock on F. The output of this would be 3030.

Concurrency Control Exercises

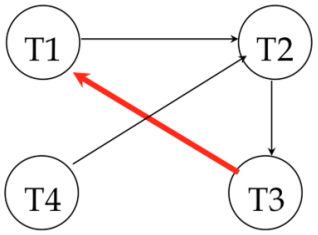
4. Consider the following schedule. Draw a “waits-for” graph and state whether or not there is a possibility of deadlock. Assume that no locks are released within the timeframe we are looking.

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

Concurrency Control Exercises

4. Consider the following schedule. Draw a “waits-for” graph and state whether or not there is a possibility of deadlock. Assume that no locks are released within the timeframe we are looking.

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



Yes, deadlock is possible since there is a cycle in the waits-for graph.