Concurrency Control 2

Instructor: Beom Heyn Kim

beomheynkim@hanyang.ac.kr

Department of Computer Science

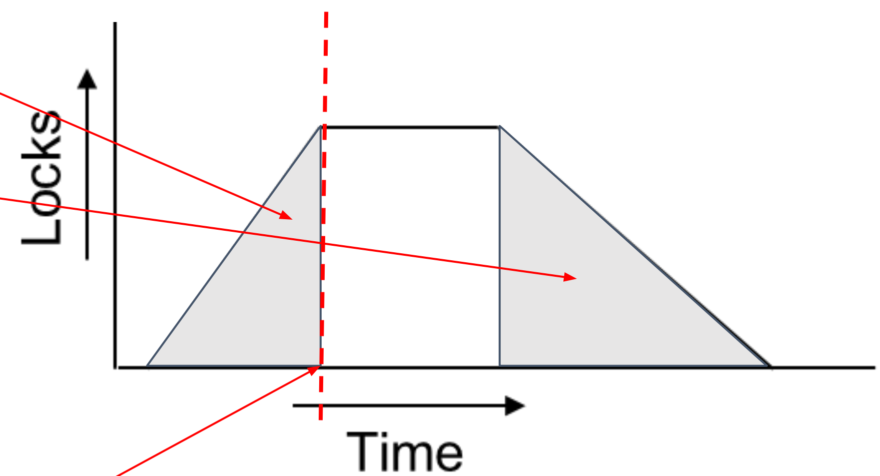
Overview

- Lock-based protocols 2
- Assignments

Two-Phase Locking Protocol

Two-phase locking protocol has two phases:

- Phase 1: **Growing Phase**
  - Transaction may obtain locks
  - Transaction may not release locks
- Phase 2: **Shrinking Phase**
  - Transaction may release locks
  - Transaction may not obtain any new locks
- Two-phase locking protocol ensures conflict serializability.
  - It can be proved that the transactions can be serialized in the order of their **lock points** (i.e., the point where a transaction acquired its final lock).



Two-Phase Locking Protocol (Cont.)

- T1 and T2 are NOT two phase

T_1:	T_2:
lock-X(B);	lock-S(A);
read(B);	read(A);
$B := B - 50$;	unlock(A);
write(B);	lock-S(B);
unlock(B);	read(B);
lock-X(A);	unlock(B);
read(A);	display($A + B$).
$A := A + 50$;	
write(A);	
unlock(A).	

Two-Phase Locking Protocol (Cont.)

- T3 and T4 are two phase

T_3:
- Growing phase:
  - lock-X(B);
  - read(B);
  - $B := B - 50$;
  - write(B);
  - lock-X(A);
- read(A);
- $A := A + 50$;
- write(A);
- Shrinking phase:
  - unlock(B);
  - unlock(A).

T_4:
- Growing phase:
  - lock-S(A);
  - read(A);
  - lock-S(B);
- read(B);
- display($A + B$);
- Shrinking phase:
  - unlock(A);
  - unlock(B).

Lock Points

The two -phase locking protocol ensures conflict-serializability

Two-Phase Locking Protocol (Cont.)

- Two-phase locking protocol is not free from deadlock.
  - Although T3 and T4 in the previous slide are two-phase, deadlock can occur. Recall:

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

Two-Phase Locking Protocol (Cont.)

- The two-phase locking is not free from cascading rollback.

T_5	T_6	T_7
lock-X(A)		
read(A)		
lock-S(B)		
read(B)		
write(A)		
unlock(A)		
	lock-X(A)	
	read(A)	
	write(A)	
	unlock(A)	
		lock-S(A)
		read(A)

Two-Phase Locking Protocol (Cont.)

- Extensions to basic two-phase locking to avoid cascading rollbacks:
  - **Strict two-phase locking**: a transaction must hold all its exclusive locks until it commits/aborts.
    - Ensures recoverability and avoids cascading rollbacks
  - **Rigorous two-phase locking**: a transaction must hold *all* locks until it commits/aborts.
    - Transactions can be serialized in the order in which they commit.
- Most databases implement rigorous two-phase locking, but refer to it as simply *two-phase locking*

Lock Conversion

- Consider the following transactions:

T_8: read(a_1);
read(a_2);
. . .
read(a_n);
write(a_1).

T_9: read(a_1);
read(a_2);
display($a_1 + a_2$).

- What types of locks needed to be acquired by T_8 and T_9?
  - T_8 has write(a_1), so it requests a lock-X
  - T_9 has only reads, so it requests a lock-S

lock-X(a_1)
T_8: read(a_1);
read(a_2);
. . .
read(a_n);
write(a_1).
unlock(a_1)

T_9 cannot be executed between.
Only serial execution is possible.
Note: X-lock is only needed right before write(a_1)

Lock Conversion (Cont.)

- Lock conversion:
  - Exclusive locks can be **downgraded** to shared locks
  - Shared locks can be **upgraded** to exclusive locks
  - Upgrading can take place in only the growing phase
  - Downgrading can take place in only the shrinking phase
- More concurrency becomes possible:

T_8	T_9
lock-S(a_1)	
read(a_1);	lock-S(a_1)
lock-S(a_2)	read(a_1);
read(a_2);	lock-S(a_2)
lock-S(a_3)	read(a_2);
lock-S(a_4)	display($a_1 + a_2$);
read(a_n);	unlock(a_1)
	unlock(a_2)
lock-S(a_n)	
upgrade(a_1)	
write(a_1);	

- 2 phase locking with lock conversion ensures conflict-serializability

Locking Protocols

- Given a locking protocol (such as 2PL – i.e. two-phase locking)
  - A schedule S is **legal** under a locking protocol if it can be generated by a set of transactions that follow the protocol
  - A protocol **ensures** serializability if all legal schedules under that protocol are serializable
    - e.g. 2PL protocol ensures serializability, so:
      - all legal schedules under a 2PL protocol are serializable
- However, 2PL protocol is not a necessary condition for serializability
  - There are conflict-serializable schedules that cannot be obtained if the two-phase locking protocol is used
    - There are other non-two-phase locking protocols ensuring conflict-serializability with additional requirements (e.g. graph-based protocols in 18.1.5)

Automatic Acquisition of Locks

- A transaction T_i issues the standard read/write instruction, without explicit locking calls.
- The operation **read**(D) is processed as:

  if T_i has a lock on D

  then

  read(D)

  else begin

  if necessary wait until no other transaction has a **lock-X** on D

  grant T_i a **lock-S** on D;

  read(D)

  end

Automatic Acquisition of Locks (Cont.)

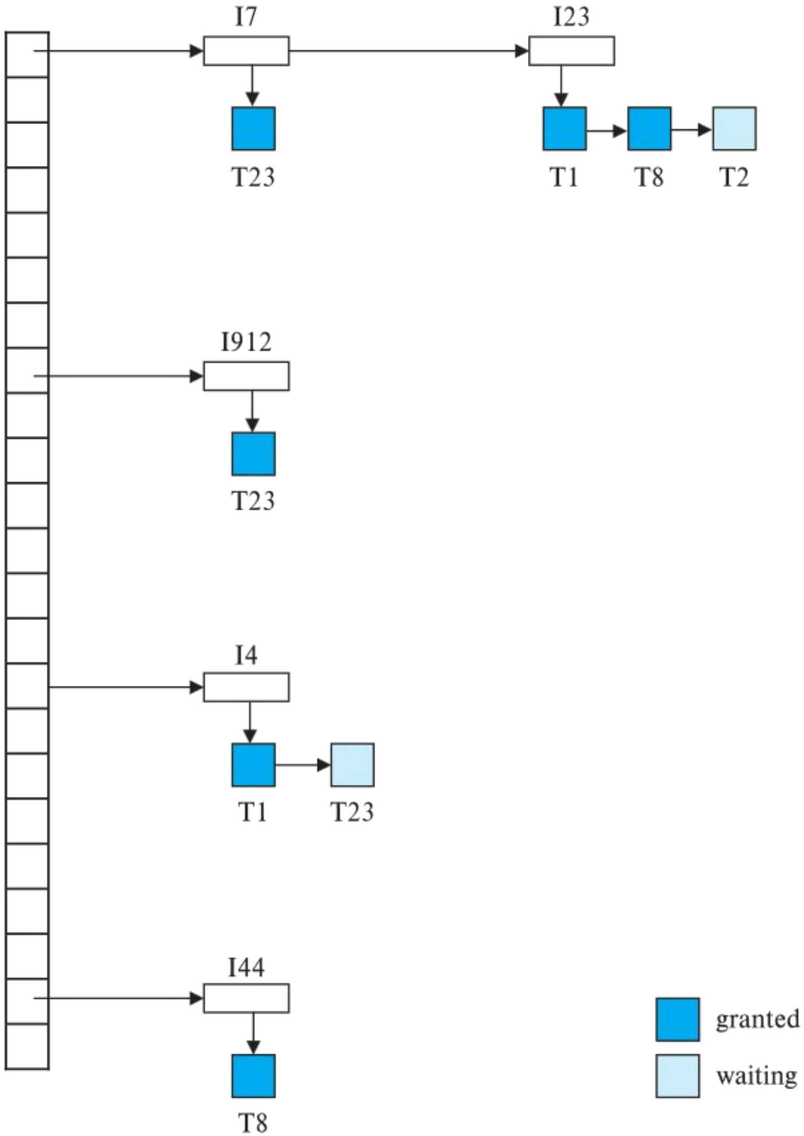
- The operation **write**(*D*) is processed as:

  if T_i has a **lock-X** on *D*
  then
  write(*D*)
  else begin
  if necessary, wait until no other transaction has any lock on *D*
  if T_i has a **lock-S** on *D*
  then
  upgrade lock on *D* to **lock-X**
  else
  grant T_i a **lock-X** on *D*
  write(*D*)
  end;
- **All locks are released after commit or abort**

Implementation of Locking

- A **lock manager** can be implemented as a separate process
- Transactions can send lock and unlock requests as messages
- The lock manager replies to a lock request by sending a lock grant messages (or a message asking the transaction to roll back, in case of a deadlock)
  - The requesting transaction waits until its request is answered
- The lock manager maintains an in-memory data-structure called a **lock table** to record granted locks and pending requests

Lock Table



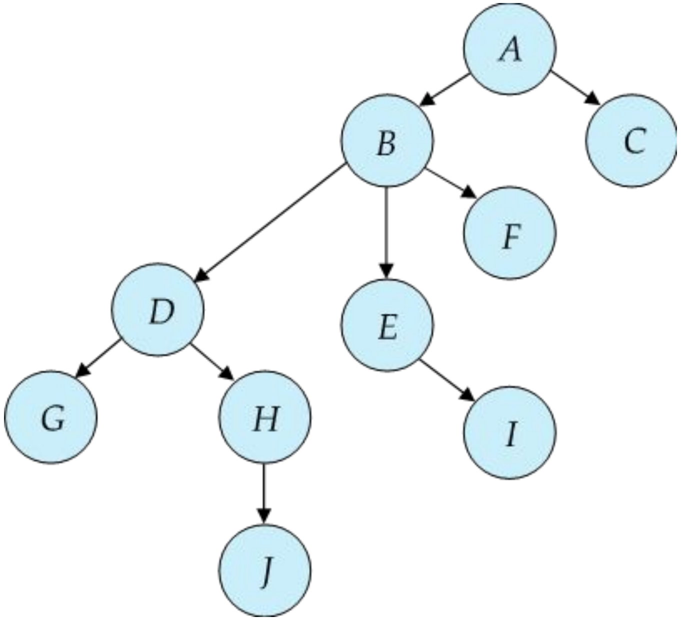
- Dark rectangles indicate granted locks, light colored ones indicate waiting requests
- Lock table also records the type of lock granted or requested
- New request is added to the end of the queue of requests for the data item, and granted if it is compatible with all earlier locks
- Unlock requests result in the request being deleted, and later requests are checked to see if they can now be granted
- If transaction aborts, all waiting or granted requests of the transaction are deleted
    - lock manager may keep a list of locks held by each transaction, to implement this efficiently

Graph-Based Protocols

- Graph-based protocols are an alternative to 2PL protocol
- Impose a partial ordering $\rightarrow$ on the set **D** = $\{d_1, d_2, \ldots, d_h\}$ of all data items.
  - If $d_i \rightarrow d_j$ then any transaction accessing both d_i and d_j must access d_i before accessing d_j.
  - Implies that the set **D** may now be viewed as a directed acyclic graph, called a *database graph*.
- The *tree-protocol* is a simple kind of graph protocol.

Tree Protocol

- Only exclusive locks are allowed.
- The first lock by T_i may be on any data item. Subsequently, a data Q can be locked by T_i only if the parent of Q is currently locked by T_i.
- Data items may be unlocked at any time.
- A data item that has been locked and unlocked by T_i cannot subsequently be relocked by T_i



Graph-Based Protocols (Cont.)

- The tree protocol ensures conflict serializability as well as freedom from deadlock.
- Unlocking may occur earlier in the tree-locking protocol than in the two-phase locking protocol.
  - Shorter waiting times, and increase in concurrency
  - Protocol is deadlock-free, no rollbacks are required
- Drawbacks
  - Protocol does not guarantee recoverability or cascade freedom
    - Need to introduce commit dependencies to ensure recoverability
  - Transactions may have to lock data items that they do not access.
    - increased locking overhead, and additional waiting time
    - potential decrease in concurrency
- Schedules not possible under two-phase locking are possible under the tree protocol, and vice versa.

Overview

- Lock-based protocols 2
- Assignments

Assignments

- Reading: Ch 18.1.3, 18.1.4, 18.1.5
- Practice Excercises: 18.1, 18.2, 18.5

Solutions to the Practice Excercises:
https://www.db-book.com/Practice-Exercises/index-solu.html

The End