

CS3743 Introduction to Database Systems Concurrency Control

Dr. Weining Zhang

Department of Computer Science
University of Texas at San Antonio

October 25, 2009

Outline

- 1 Concurrency Control
 - Overview
 - Two Phase Locking
 - Deadlock Management

Concurrency Control Techniques

- Concurrency control manager must guarantee serializable schedule. But, it is impractical to generate a schedule and then test for serializability.
 - It is too costly to fix a schedule that is not serializable
- In practice, CC manager always creates a serial schedule using a scheduler, which determines the ordering of transaction operations using one of three techniques
 - **Locking Protocol.** Determine ordering using locks on data items
 - **Timestamp Ordering.** Determine ordering based on timestamps of transactions
 - **Validation.** Validate an ordering when a transaction is ready to commit

Lock Compatibility

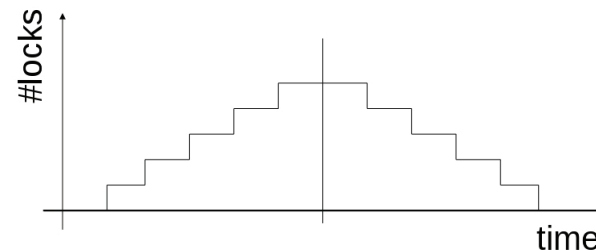
- Locking protocols are typically based on different types of locks
 - **Read lock** for read only access, **write lock** for both read and write accesses
 - Lock requests are denoted as $RL_i(X)$, $WL_i(X)$ and $UL_i(X)$
 - DBMS uses a lock compatibility table to determine if a lock request can be granted
- Suppose T_i holds a lock on data X and T_j also requests a lock on the same data. The following table determines if T_j 's request can be granted (Y means Yes, and N means No)

	RL_i	WL_i
RL_j	Y	N
WL_j	N	N

Basic Two Phase Locking (2PL)

- A transaction must request and be granted an appropriate lock before it can access a data. It must request
 - a shared lock for reads
 - an exclusive lock for writes
- CC scheduler uses a lock manager to grant lock requests according to lock compatibility table
- **A transaction may not request any more locks once it releases a lock**
- A transaction can release a lock at any time

The Phases in 2PL



Example of 2PL

Example

Following schedule S is 2PL.

$$S : WL_1(X)R_1(X)W_1(X)WL_1(Y)UL_1(X)WL_2(X)R_2(X)W_2(X) \\ R_1(Y)W_1(Y)UL_1(Y)WL_2(Y)UL_2(X)R_2(Y)W_2(Y)UL_2(Y)$$

Schedule S' is not 2PL

$$S' : WL_1(X)R_1(X)W_1(X)UL_1(X)WL_2(X)WL_1(Y)R_2(X)W_2(X) \\ R_1(Y)W_1(Y)UL_1(Y)WL_2(Y)UL_2(X)R_2(Y)W_2(Y)UL_2(Y)$$

Correctness of 2PL

Theorem

Every 2PL schedule is serializable

Intuitively, locks held by a transaction guarantee its operations are performed before other transactions that may perform any conflict operation. Therefore, the ordering of conflict operations is determined by the order of the first unlocks of the transactions, which is equivalent to a serial ordering of transactions.

Problems of Basic 2PL

Basic 2PL may have two problems.

- Cascading abort. The abort of a transaction forces another transaction to abort
 - Caused by releasing locks too early
- **Deadlock**. A situation in which a set of transactions wait for each other to release locks and no one can make any progress
 - Caused by not releasing locks early

Deadlock Management

There are two basic approaches to handle deadlock

- Deadlock Resolution.
 - Periodically detect for deadlock. If detected, abort some transactions involved in the deadlock.
 - Need to avoid **starvation**, a situation in which a transaction is always chosen as a victim
- Deadlock Prevention.
 - Prevent deadlock from occurring by additional rules about granting and releasing locks
 - Techniques include
 - Transaction prioritization
 - Conservative 2PL

Example of Deadlock

Example

Following schedule S_1 can cause a cascading abort

$S_1 : WL_1(X)R_1(X)W_1(X)WL_1(Y)UL_1(X)WL_2(X)R_2(X)W_2(X)$
 $R_1(Y)A_1W_1(Y)UL_1(Y)WL_2(Y)UL_2(X)R_2(Y)W_2(Y)UL_2(Y)$

schedule S_2 can cause a deadlock

$S_2 : WL_1(X)R_1(X)WL_2(Y)R_2(Y)W_1(X)W_2(Y)WL_1(Y)WL_2(X)$
 $R_1(Y)R_2(X)W_2(X)UL_2(Y)W_1(Y)UL_1(X)UL_1(Y)UL_2(X)$

Deadlock Detection

- The **wait-for-graph** (WFG) of a schedule S is a directed graph with each transaction in S being a vertex. There is a directed edge a transaction T_i to another transaction T_j iff T_i is waiting for T_j to release a lock on any data.
- A schedule causes a deadlock iff its WFG contains a cycle.

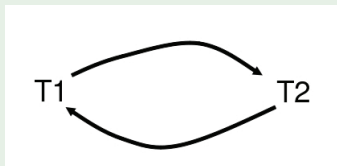
Example of Deadlock Detection

Example

Check to see if the following schedule can cause a deadlock

$S : WL_1(X)R_1(X)WL_2(Y)R_2(Y)W_1(X)W_2(Y)WL_1(Y)WL_2(X)$
 $R_1(Y)R_2(X)W_2(X)UL_2(Y)W_1(Y)UL_1(X)UL_1(Y)UL_2(X)$

Resulted WFG is



Transaction prioritization

- Assign a priority to each transaction and avoid potential deadlocks by aborting transactions with lower priorities.
- One priority measure is time stamp of each transaction. Older transactions have higher priority. Suppose T_i requests an incompatible lock held by T_j .
 - **Wait-Die Rule.** T_i will wait if it is older; dies, otherwise. (i.e., aborted and restarted with the original time stamp)
 - **Wound-Wait Rule.** T_i will wait if it is younger; otherwise, it will wound T_j (i.e., aborts T_j and gets the lock; T_i will restart with its original time stamp).

Example of Transaction Prioritization

Example

Consider following schedule. Without transaction prioritization, it will have a deadlock before $R_2(X)$.

$S : WL_1(X)R_1(X)W_1(X)WL_2(Y)R_2(Y)W_2(Y)$
 $WL_1(Y)R_1(Y)W_1(Y)UL_1(X)UL_1(Y)WL_2(X)$
 $R_2(X)W_2(X)UL_2(Y)UL_2(X)$

Assume that T_1 is older than T_2 .

- With Wait-Die, T_1 waits for T_2 on Y , T_2 dies on X ;
- With Wound-Wait: T_2 is aborted, T_1 gets lock on Y .

Conservative 2PL

- A transaction must request and be granted an appropriate lock before it can access a data. It must request
 - a shared lock for reads
 - an exclusive lock for writes
- CC scheduler uses a lock manager to grant lock requests according to lock compatibility table
- A transaction may not request any more locks once it releases a lock
- **A transaction must obtain all locks it needs at beginning, otherwise, it must release all locks obtained so far**
- A transaction can release a lock at any time