Database Systems
(資料庫系統)

January 3/4, 2007

Lecture 13b

Announcement

- Assignment #5 is due on Thur.
- Assignment #6 will be out on the course webpage tomorrow.

Concurrency Control

Chapter 17 (17.1~17.4)

3

The Question

- How to ensure that a schedule is both serializable & recoverable?
  - What is Serializable?
    - Correct results from interleaving of transactions
  - What is Recoverable?
    - Safe from aborted transactions
- Does strict 2PL give serializability and recoverability?
- How about 2PL?
  - Draw a diagram!

4

Conflict Serializable Schedules

- Two schedules are conflict equivalent if:
  - Involve the same (write/read) actions of the same transactions
  - Every pair of conflicting actions is ordered the same way
    - Conflicting actions = actions on the same data object and at least one of the action is a write.
- Schedule S is conflict serializable if S is conflict equivalent to a serial schedule
  - A serial schedule is a schedule with no interleaving actions from different transactions.
  - A serializable schedule is a schedule that produces identical result as some serial schedule.
  - A conflict serializable schedule is serializable + (closer to recoverable)

5

Example

- Is this serializable?
  - Same results as the serial schedule T1, T2, T3.
- Is this conflict serializable (T1, T2, T3)?
  - No, Writes of T1 and T2 (conflicting actions) are ordered differently than the serial schedule of T1,T2,T3.

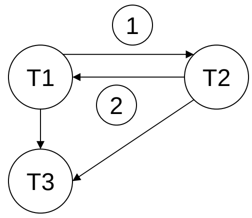
T1	T2	T3
R(A)		
	W(A)	
	Commit	
W(A)		
Commit		
		W(A)
		Commit

6

Precedence Graph

- How do we know a given schedule is conflict serializable?
- Capture the conflicting actions on a precedence graph & check for cycle in the graph.
  - One node per transaction
  - Edge from Ti to Tj if an action of Ti precedes and conflicts (R/W, W/R, W/R) with one of Tj's actions.
- Schedule is conflict serializable if and only if its precedence graph is acyclic.

T1	T2	T3
R(A)		
	W(A)	
	Commit	
W(A)		
Commit		
		W(A)
		Commit



7

Strict 2PL Review

- Strict Two-phase Locking (Strict 2PL) Protocol:
  - If a transaction T wants to read an object, it requests a shared lock. Denote as S(O). If a transaction T wants to write an object, it requests an exclusive lock. Denote as X(O).
  - Locks are released only when transaction is completed (aborted).
  - If a transaction holds an X lock on an object, no other transaction can get a lock (S or X) on that object.

8

Example #1 of Strict 2PL

T1
S(A)
R(A)
X(C)
R(C)
W(C)
Commit

T2
S(A)
R(A)
X(B)
R(B)
W(B)
Commit

T1	T2
	t
t	

Why Strict 2PL allows only conflict serializable schedules?

- Say T1 & T2 have conflicting actions:
  - RW, WR, and WW on the same data object
- If T1 obtains a lock on the data object (X) first, T2's conflicting actions on X must wait until T1 is done.
- When could a backward edge occur from T2 to T1?
  - Must come from conflicting actions on another data object (A).
  - What if T2 obtains a lock on A before T1 obtains a lock on A?

T1	T2
Conflicting action on data #1 (get lock)	
	Conflicting action on data #2 (get lock?)
Conflicting action on data #2 (get lock)	
Commit (release locks)	
	Conflicting action on data #1 (get lock)

11

Two-Phase Locking (2PL)

- 2PL Protocol is the same as Strict 2PL, except
  - A transaction can release locks before the end (unlike Strict 2PL), i.e., after it is done with reading & writing the objects.
  - However, a transaction can not request additional locks after it releases any locks.
  - 2PL has lock growing phase and shrinking phase.

12

2PL: without strict

T1	T2
X(A)	X(A)
R(A)	R(A)
W(A)	W(A)
X(B)	X(B)
R(B)	R(B)
W(B)	W(B)
Commit // release locks	Commit // release locks

lock growing phase

lock shrinking phase

T1	T2
X(A)	Suspend
R(A)	
W(A)	
X(B)	
Release X(A)	
	X(A)
	R(A)
	W(A)
R(B)	
W(B)	
Release X(B)	
Commit	
	X(B)
	R(B)
	W(B)
	Release X(A)
	Release X(B)
	Commit

13

Non-recoverable Schedule

- Schedule allowed by 2PL may not be recoverable in aborts. Why?
  - Say T1 aborts and we need to undo T1.
  - But T2 has read a value for A that should never been there.
  - But T2 has committed! (may not be able to undo committed actions).

T1	T2
X(A)	
R(A)	
W(A)	
Release X(A)	
	X(A)
	R(A)
	W(A)
	X(B)
	R(B)
	Release X(A)
	W(B)
	Release X(B)
	Commit
Abort / Commit	

14

Two-Phase Locking (2PL)

- What is the benefit of 2PL over strict 2PL?
  - Smaller lock holding time → better concurrency
- What is the benefit of strict 2PL over 2PL?
  - Recoverable schedule vs. non-recoverable schedule
- 2PL also produces conflict serializable schedules.

Revisit the Example: Serializable but not Conflict Serializable

- Serializable:
  - same results as the serial schedule T1, T2, T3.
- Not conflict serializable
  - Writes of T1 and T2 (conflicting actions) are ordered differently than the serial schedule of T1,T2,T3.
- Conflict Serializable is a subset Serializable
- Is there a subset contains conflict serializable but within serializable?
  - Include this left example

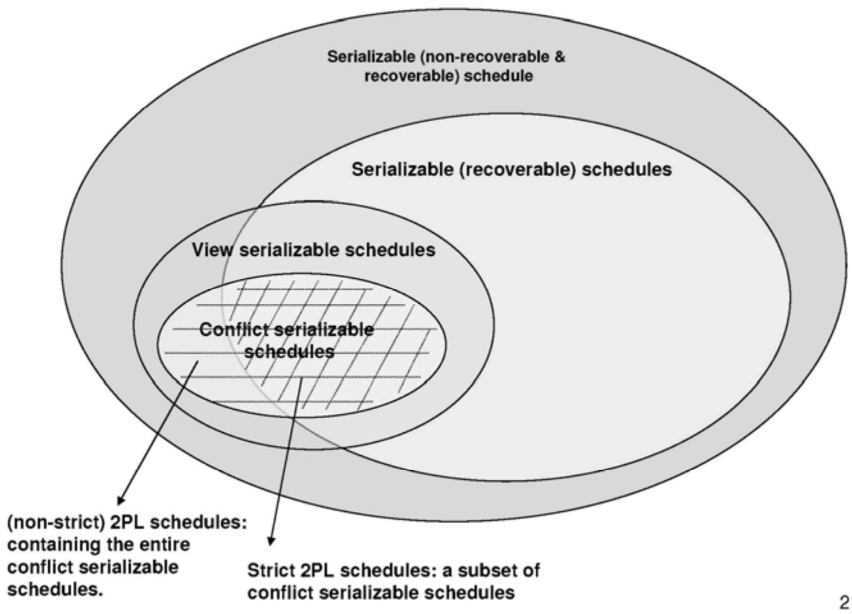
T1	T2	T3
R(A)		
	W(A)	
	Commit	
W(A)		
Commit		
		W(A)
		Commit

View Serializable

- Initial value:
  - If Ti reads the initial value of object A in S1, it must also read the initial value of A in S2 (a serial schedule).
- Read values:
  - If Ti reads a value of A written by Tj in S1, it must also read the value of A written by Tj in S2.
  - That's why it is not view (serializable) equivalent to T2, T1, T3
- Final written values:
  - For each data object A, the transaction that performs the final write on A in S1 must also perform the final write on A in S2.

T1	T2	T3
R(A)		
	W(A)	
	Commit	
W(A)		
Commit		
		W(A)
		Commit

17



21

Lock Management

- Lock and unlock requests are handled by the lock manager.
- Each lock manager maintains a lock table of lock table entries.
- Each lock table entry keeps info about:
  - The data object (page, record) being locked
  - Number of transactions currently holding a lock (>1 if shared mode)
  - Type of lock held (shared or exclusive)
  - Lock request queue

Lock Management Implementation

- If a shared lock is requested:
  - Check if the request queue is empty. Check if the lock is in exclusive/share mode. If (yes,share), grant the lock & update the lock table entry.
- When a transaction aborts or commits, it releases all lock.
  - Update the lock table entry. Check for lock request queue, …
- Locking and unlocking have to be atomic operations:
  - E.g., cannot have concurrent operations on the same lock table entry.

Lock Conversions

- Lock upgrade: transaction that holds a shared lock can be upgraded to hold an exclusive lock.
  - Get shared lock on each row in a table.
  - When a row meets the condition, get an exclusive lock.
- Alternative approach is lock downgrade.
  - Get exclusive lock on each row in a table.
  - When a row does not meet the condition, downgrade to shared lock.

```
UPDATE Sailors S
SET     S.age=10
WHERE S.name="Joe"
          AND S.rating=8
```

21

Deadlocks

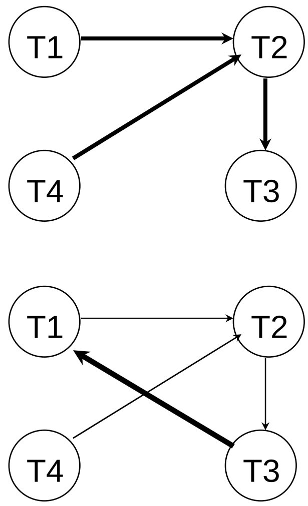
- Deadlock: Cycle of transactions waiting for locks to be released by each other.
- Two ways of dealing with deadlocks:
  - Deadlock detection
  - Deadlock prevention
- Deadline Detection:
  - Create a waits-for graph:
    - Nodes are transactions
    - There is an edge from Ti to Tj if Ti is waiting for Tj to release a lock
  - Periodically check for cycles in the waits-for graph,
    - cycle = Deadlock
    - Resolve a deadlock by aborting a transaction on a cycle.

T1	T2
S(A)	
R(A)	S(B)
	R(B)
X(B): W(B)	
	X(A): W(A)

22

Deadlock Detection

T1	T2	T3	T4
S(A)			
R(A)			
	X(B)		
	W(B)		
S(B)			
		S(C)	
		R(C)	
	X(C)		
			X(B)
		X(A)	



Deadlock Prevention

- Make sure that deadlock will not occur.
- How to do it?
  - Assign priorities to transactions based on timestamps when they start up. (lower timestamps = higher priority)
  - Lower priorities cannot wait for higher-priority transactions.

Deadlock Prevention

- Say Ti wants a lock that Tj holds. What would a deadlock prevention policy do?
  - Wait-Die: If Ti has higher priority, Ti waits for Tj; otherwise Ti aborts (lower-priority T never waits for higher-priority T)
  - Wound-wait: If Ti has higher priority, Tj aborts; otherwise Ti waits. (higher-priority T never waits for lower-priority T)
- Why these two policies prevent deadlocks?
  - The oldest transaction (highest priority one) will eventually get all the locks it requires!

25

Wait-Die Policy

- Lower-priority T never waits for higher-priority T
- If Ti has higher priority, Ti waits for Tj;
- Otherwise Ti aborts

T1	T2	T3	T4
S(A)			
R(A)			
	X(B)		
	W(B)		
S(B)			
		S(C)	
		R(C)	
	X(C)		
			X(B) // abort
		X(A) // abort	

26

Wound-Wait Policy

- Higher-priority T never waits for lower-priority T
- If Ti has higher priority, Tj aborts;
- Otherwise Ti waits.

T1	T2	T3	T4
S(A)			
R(A)			
	X(B)		
	W(B)		
S(B) // Abort T2	Abort		

27

Deadlock Prevention (2)

- If a transaction re-starts, make sure it has its original timestamp.
  - It will not be forever aborted due to low priority.
- How to design a locking protocol that ensures no deadlock?
  - Conservative 2PL: ensure no deadlock (no blocking) during transaction
    - Each transaction begins by getting all locks it will ever need
  - What is the tradeoff?
    - Longer lock holding time -> reduce concurrency

28