Introduction to Database Systems
CSE 414

Lecture 28:
Transactions Wrap-up

Announcements

- 2 late days for HW 8 are now free
  – No more than 2 late days allowed. Monday Dec. 10 is the hard cut off
- Office hours changes
  – Ryan tomorrow at 11am instead of 10:30
  – Andrew additional office hours Friday

A New Problem: Non-recoverable Schedule

T1	T2
$L_1(A)$; $L_1(B)$; READ(A)	
A :=A+100	
WRITE(A); $U_1(A)$	
	$L_2(A)$; READ(A)
	A := A*2
	WRITE(A);
	$L_2(B)$; BLOCKED…
READ(B)	
B :=B+100	
WRITE(B); $U_1(B)$;	
	…GRANTED; READ(B)
	B := B*2
	WRITE(B); $U_2(A)$; $U_2(B)$;
	Commit
Rollback	

A New Problem: Non-recoverable Schedule

T1	T2
L_1(A); L_1(B); READ(A)	
A :=A+100	
WRITE(A); U_1(A)	
	L_2(A); READ(A)
	A := A*2
	WRITE(A);
	L_2(B); BLOCKED…
READ(B)	
B :=B+100	
WRITE(B); U_1(B);	
	…GRANTED; READ(B)
	B := B*2
	WRITE(B); U_2(A); U_2(B);
	Commit
Rollback	

Elements A, B written by T1 are restored to their original value.

A New Problem: Non-recoverable Schedule

T1	T2
$L_1(A)$; $L_1(B)$; READ(A)	
A :=A+100	
WRITE(A); $U_1(A)$	
	$L_2(A)$; READ(A)
	A := A*2
	WRITE(A);
	$L_2(B)$; BLOCKED…
READ(B)	
B :=B+100	
WRITE(B); $U_1(B)$;	
	…GRANTED; READ(B)
	B := B*2
	WRITE(B); $U_2(A)$; $U_2(B)$;
	Commit
Rollback	

Dirty reads of A, B lead to incorrect writes.

Elements A, B written by T1 are restored to their original value.

A New Problem: Non-recoverable Schedule

T1	T2
$L_1(A)$; $L_1(B)$; READ(A)	
A :=A+100	
WRITE(A); $U_1(A)$	
	$L_2(A)$; READ(A)
	A := A*2
	WRITE(A);
	$L_2(B)$; BLOCKED…
READ(B)	
B :=B+100	
WRITE(B); $U_1(B)$;	
	…GRANTED; READ(B)
	B := B*2
	WRITE(B); $U_2(A)$; $U_2(B)$;
	Commit
Rollback	

Dirty reads of A, B lead to incorrect writes.

Elements A, B written by T1 are restored to their original value.

Can no longer undo!

Strict 2PL

The Strict 2PL rule:

All locks are held until commit/abort:
All unlocks are done together with commit/abort.

With strict 2PL, we will get schedules that are both conflict-serializable and recoverable

Strict 2PL

T1	T2
$L_1(A)$; READ(A)	
A :=A+100	
WRITE(A);	
	$L_2(A)$; BLOCKED…
$L_1(B)$; READ(B)	
B :=B+100	
WRITE(B);	
Rollback & $U_1(A)$;$U_1(B)$;	
	…GRANTED; READ(A)
	A := A*2
	WRITE(A);
	$L_2(B)$; READ(B)
	B := B*2
	WRITE(B);
	Commit & $U_2(A)$; $U_2(B)$;

Strict 2PL

- Lock-based systems always use strict 2PL
- Easy to implement:
  - Before a transaction reads or writes an element A, insert an L(A)
  - When the transaction commits/aborts, then release all locks
- Ensures both conflict serializability and recoverability

Another problem: Deadlocks

- T_1: R(A), W(B)
- T_2: R(B), W(A)

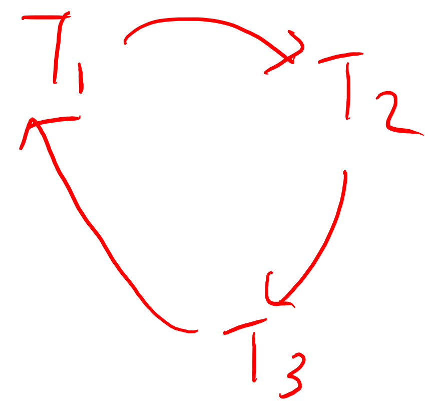
- T_1 holds the lock on A, waits for B
- T_2 holds the lock on B, waits for A

This is a deadlock!

Another problem: Deadlocks

To detect a deadlocks, search for a cycle in the *waits-for graph*:

- T_1 waits for a lock held by T_2;
- T_2 waits for a lock held by T_3;
- . . .
- T_n waits for a lock held by T_1



Relatively expensive: check periodically, if deadlock is found, then abort one transaction.
need to continuously re-check for deadlocks

A "Solution"?: Lock Modes

- S = shared lock (for READ)
- X = exclusive lock (for WRITE)

Lock compatibility matrix:

	None	S	X
None			
S			
X			

A “Solution”?: Lock Modes

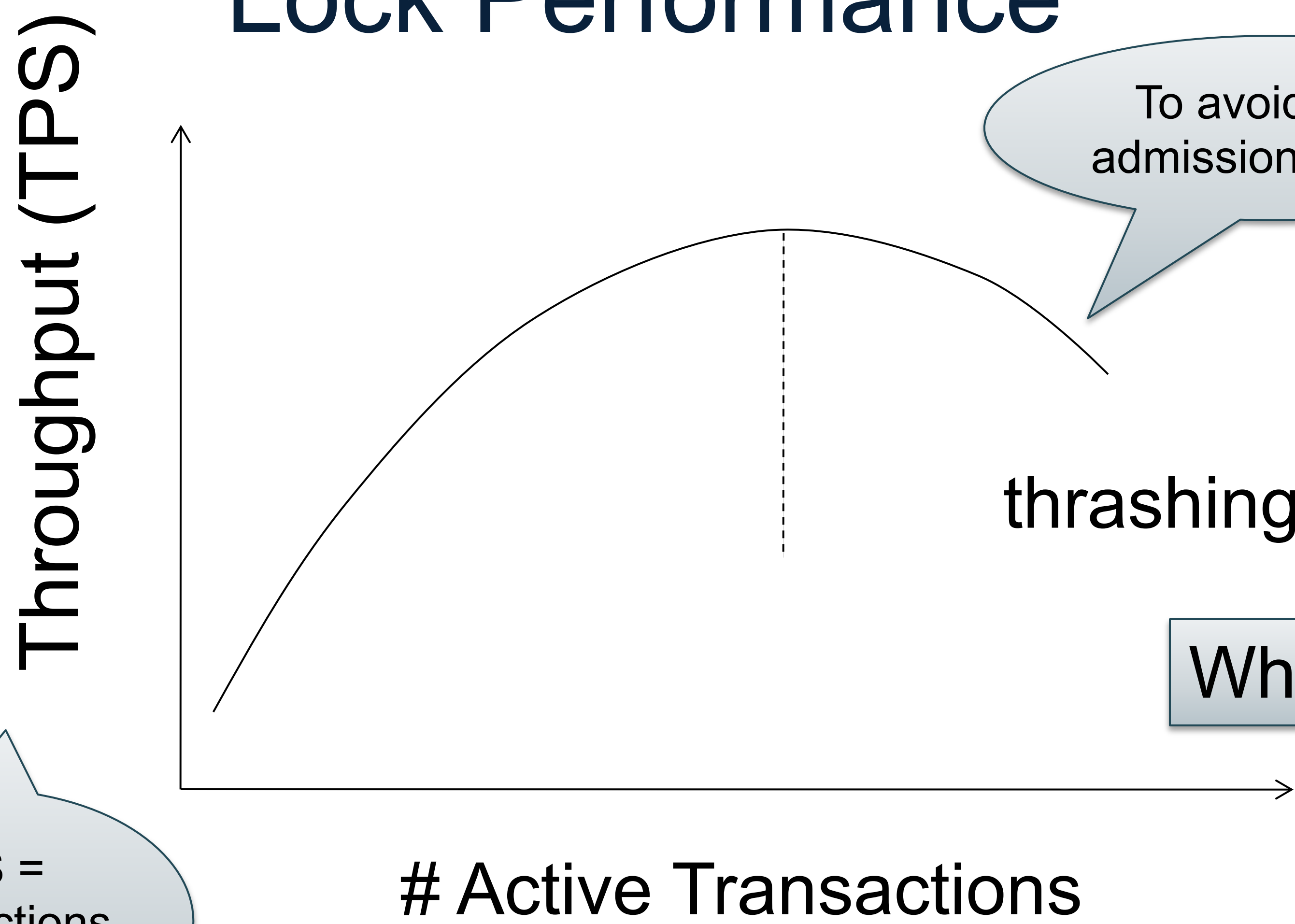
- S = shared lock (for READ)
- X = exclusive lock (for WRITE)

Lock compatibility matrix:

	None	S	X
X			

Can only fix deadlocks if transactions declare exclusive locks in advance.

Lock Performance



To avoid, use admission control

Why ?

TPS = Transactions per second

Lock Granularity

- Fine granularity locking (e.g., tuples)
  - High concurrency
  - High overhead in managing locks
  - E.g., SQL Server
- Coarse grain locking (e.g., tables, entire database)
  - Many false conflicts
  - Less overhead in managing locks
  - E.g., SQL Lite
- Solution: lock escalation changes granularity as needed

Phantom Problem

- So far we have assumed the database to be a *static* collection of elements (=tuples)
- If tuples are inserted/deleted then the *phantom problem* appears

Suppose there are two blue products, A1, A2:

Phantom Problem

T1	T2
SELECT * FROM Product WHERE color='blue'	
	INSERT INTO Product(name, color) VALUES ('A3','blue')
SELECT * FROM Product WHERE color='blue'	

Is this schedule serializable ?

Suppose there are two blue products, A1, A2:

Phantom Problem

T1	T2
SELECT * FROM Product WHERE color=‘blue’	
	INSERT INTO Product(name, color) VALUES (‘A3’,’blue’)
SELECT * FROM Product WHERE color=‘blue’	

$R_1(A1);R_1(A2);W_2(A3);R_1(A1);R_1(A2);R_1(A3)$

Suppose there are two blue products, A1, A2:

Phantom Problem

T1	T2
SELECT * FROM Product WHERE color=‘blue’	
	INSERT INTO Product(name, color) VALUES (‘A3’,’blue’)
SELECT * FROM Product WHERE color=‘blue’	

$R_1(A1);R_1(A2);W_2(A3);R_1(A1);R_1(A2);R_1(A3)$

$W_2(A3);R_1(A1);R_1(A2);R_1(A1);R_1(A2);R_1(A3)$

Phantom Problem

- A “phantom” is a tuple that is invisible during part of a transaction execution but not invisible during the entire execution
- In our example:
  - T1: reads list of products
  - T2: inserts a new product
  - T1: re-reads: a new product appears !

Dealing With Phantoms

- Lock the entire table
- Lock the index entry for ‘blue’
  - If index is available
- Or use predicate locks
  - A lock on an arbitrary predicate

Dealing with phantoms is expensive !

Summary of Serializability

- Serializable schedule = equivalent to a serial schedule
- (strict) 2PL guarantees *conflict serializability*
  - What is the difference?
- **Static database**:
  - *Conflict serializability* implies serializability
- **Dynamic database**:
  - This no longer holds

Isolation Levels in SQL

For better performance

1. "Dirty reads"

   SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

2. "Committed reads"

   SET TRANSACTION ISOLATION LEVEL READ COMMITTED

3. "Repeatable reads"

   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

4. Serializable transactions

   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

ACID

1. Isolation Level: Dirty Reads

- "Long duration" WRITE locks
  - Strict 2PL
- No READ locks
  - Read-only transactions are never delayed

Possible problems: dirty and inconsistent reads

2. Isolation Level: Read Committed

- "Long duration" WRITE locks
  - Strict 2PL
- "Short duration" READ locks
  - Only acquire lock while reading (not 2PL)

Unrepeatable reads:
When reading same element twice, may get two different values

3. Isolation Level: Repeatable Read

- "Long duration" WRITE locks
  - Strict 2PL
- "Long duration" READ locks
  - Strict 2PL

Why ?

This is not serializable yet !!!

4. Isolation Level Serializable

- "Long duration" WRITE locks
  - Strict 2PL
- "Long duration" READ locks
  - Strict 2PL
- Predicate locking
  - To deal with phantoms

Beware!

In commercial DBMSs:

- Default level is often NOT serializable
- Default level differs between DBMSs
- Some engines support subset of levels!
- Serializable may not be exactly ACID
  - Locking ensures isolation, not atomicity
- Also, some DBMSs do NOT use locking and different isolation levels can lead to different pbs
- Bottom line: RTFM for your DBMS!