Transactions; Concurrency; Recovery

Amol Deshpande
CMSC424

Spring 2020 – Online Instruction Plan

- Week 1: File Organization and Indexes
- Week 2: Query Processing
- Week 3: Query Optimization; Parallel Databases 1
- Week 4: Parallel Databases; Mapreduce; Transactions 1
- Week 5: Transactions 2 (Homework Due May 1)
  - ★Transactions: Serializability, Recoverability
  - ★Transactions: Concurrency 1
  - ★Transactions: Concurrency 2: Other Concurrency Schemes
  - ★Transactions: Recovery
- Week 6: Distributed Transactions; Miscellaneous Topics (Homework Due May 8)

Transactions: Overview

- Book Chapters
  - 14.6, 14.7
- Key topics:
  - Conflict equivalence of schedules
  - Conflict serializability and checking by drawing precedence graphs
  - View serializability
  - Recoverability

Another schedule

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
read(B)	
B=B+50	
write(B)	
	read(B)
	B = B+ tmp
	write(B)

Is this schedule okay ?

Lets look at the final effect…

Effect:	Before	After
A	100	45
B	50	105

Further, the effect same as the serial schedule 1.

Called *serializable*

Conflict Serializability

- Two read/write instructions “conflict” if
  - They are by different transactions
  - They operate on the same data item
  - At least one is a “write” instruction

- Why do we care ?
  - If two read/write instructions don’t conflict, they can be “swapped” without any change in the final effect
  - However, if they conflict they CAN’T be swapped without changing the final effect

Equivalence by Swapping

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
read(B)	
B=B+50	
write(B)	
	read(B)
	B = B+ tmp
	write(B)

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
read(B)	
	write(A)
B=B+50	
write(B)	
	read(B)
	B = B+ tmp
	write(B)

Effect:	Before	After
A	100	45
B	50	105

==

Effect:	Before	After
A	100	45
B	50	105

Equivalence by Swapping

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
read(B)	
B=B+50	
write(B)	
	read(B)
	B = B+ tmp
	write(B)

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
read(B)	
B=B+50	
	read(B)
write(B)	
	B = B+ tmp
	write(B)

Effect:	Before	After
A	100	45
B	50	105

! ==

Effect:	Before	After
A	100	45
B	50	55

Conflict Serializability

- Conflict-equivalent schedules:
  - If S can be transformed into S' through a series of swaps, S and S' are called *conflict-equivalent*
  - *conflict-equivalent guarantees same final effect on the database*

- A schedule S is conflict-serializable if it is conflict-equivalent to a serial schedule

Equivalence by Swapping

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
read(B)	
B=B+50	
write(B)	
	read(B)
	B = B+ tmp
	write(B)

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
read(B)	
B=B+50	
	write(A)
write(B)	
	read(B)
	B = B+ tmp
	write(B)

Effect:	Before	After
A	100	45
B	50	105

==

Effect:	Before	After
A	100	45
B	50	105

Equivalence by Swapping

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
read(B)	
B=B+50	
write(B)	
	read(B)
	B = B+ tmp
	write(B)

T1	T2
read(A)	
A = A -50	
write(A)	
read(B)	
B=B+50	
write(B)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
	read(B)
	B = B+ tmp
	write(B)

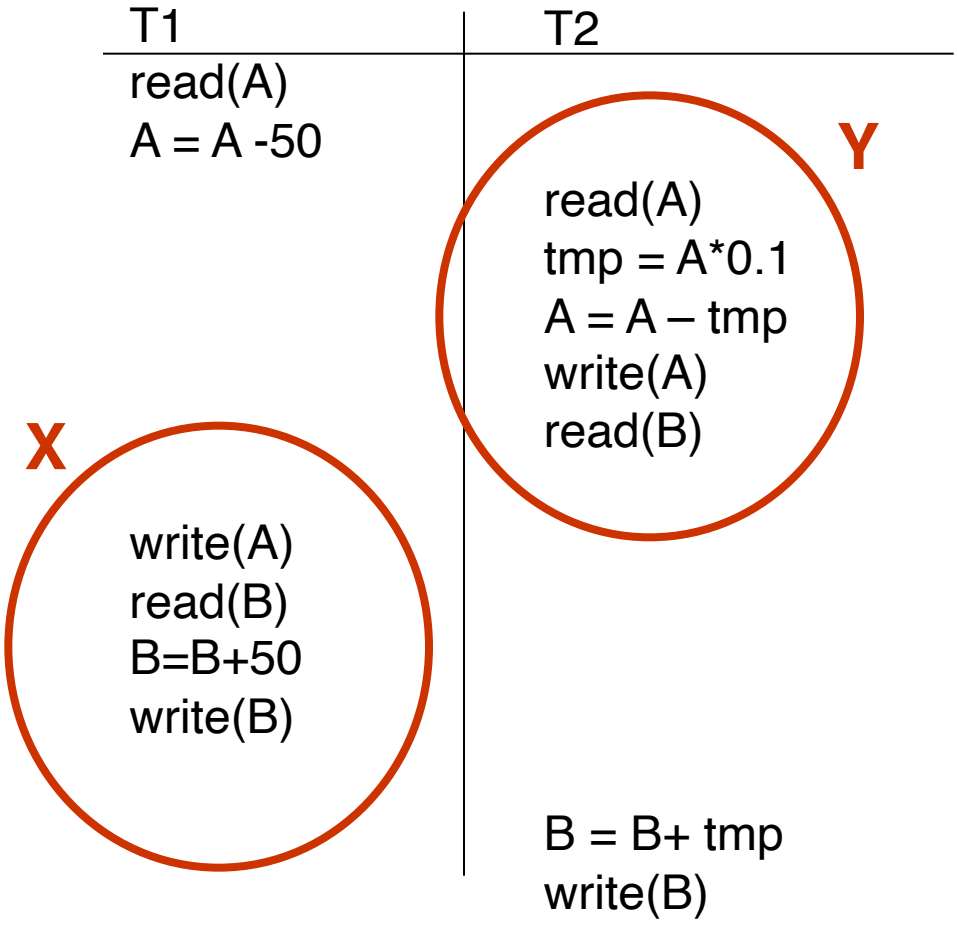
Effect:	Before	After
A	100	45
B	50	105

==

Effect:	Before	After
A	100	45
B	50	105

Example Schedules (Cont.)

A “bad” schedule



Can’t move Y below X
read(B) and write(B) conflict

Other options don’t work either

So: Not Conflict Serializable

Serializability

- In essence, following set of instructions is not conflict-serializable:

T_3	T_4
read(Q)	
	write(Q)
write(Q)	

View-Serializability

- Similarly, following not conflict-serializable

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- BUT, it is serializable
  - Intuitively, this is because the *conflicting write instructions* don't matter
  - The final write is the only one that matters

- View-serializability allows these
  - Read up

Other notions of serializability

T_1	T_5
read(A)	
$A := A - 50$	
write(A)	
	read(B)
	$B := B - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)

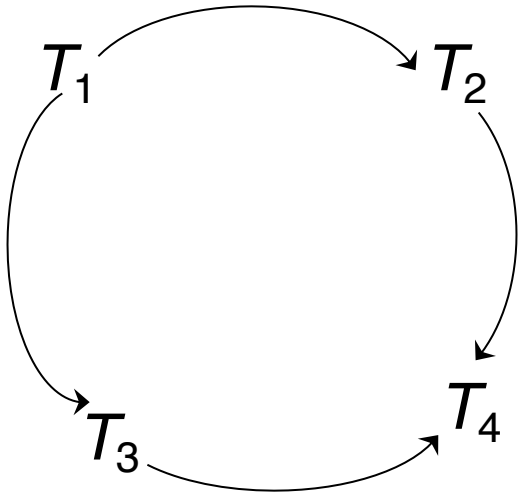
- Not conflict-serializable or view-serializable, but serializable
- Mainly because of the +/- only operations
  - Requires analysis of the actual operations, not just read/write operations
- Most high-performance transaction systems will allow these

Testing for conflict-serializability

- Given a schedule, determine if it is conflict-serializable
- Draw a *precedence-graph* over the transactions
  - A directed edge from T1 and T2, if they have conflicting instructions, and T1's conflicting instruction comes first
- If there is a cycle in the graph, not conflict-serializable
  - Can be checked in at most $O(n+e)$ time, where n is the number of vertices, and e is the number of edges
- If there is none, conflict-serializable
- Testing for view-serializability is NP-hard.

Example Schedule (Schedule A) + Precedence Graph

T_1	T_2	T_3	T_4	T_5
	read(X)			
read(Y)				
read(Z)				
				read(V)
				read(W)
				read(W)
	read(Y)			
	write(Y)			
		write(Z)		
read(U)				
			read(Y)	
			write(Y)	
			read(Z)	
			write(Z)	
read(U)				
write(U)				



Recap so far…

- We discussed:
  - Serial schedules, serializability
  - Conflict-serializability, view-serializability
  - How to check for conflict-serializability

- We haven't discussed:
  - How to guarantee serializability ?
    - Allowing transactions to run, and then aborting them if the schedules wasn't serializable is clearly not the way to go
  - We instead use schemes to guarantee that the schedule will be conflict-serializable
  - Also, ***recoverability ?***

Recoverability

- Serializability is good for consistency

- But what if transactions fail ?
  - T2 has already committed
    - A user might have been notified
  - Now T1 abort creates a problem
    - T2 has seen its effect, so just aborting T1 is not enough. T2 must be aborted as well (and possibly restarted)
    - But T2 is *committed*

T1	T2
read(A)	
A = A -50	
write(A)	
	read(A)
	tmp = A*0.1
	A = A – tmp
	write(A)
	COMMIT
read(B)	
B=B+50	
write(B)	
ABORT	

Recoverability

- Recoverable schedule: If T1 has read something T2 has written, T2 must commit before T1
  - ★ Otherwise, if T1 commits, and T2 aborts, we have a problem

- Cascading rollbacks: If T10 aborts, T11 must abort, and hence T12 must abort and so on.

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

Recoverability

- *Dirty read*: Reading a value written by a transaction that hasn't committed yet

- Cascadeless schedules:
  - A transaction only reads *committed* values.
  - So if T1 has written A, but not committed it, T2 can't read it.
    - *No dirty reads*

- Cascadeless → No cascading rollbacks
  - That's good
  - We will try to guarantee that as well

Recap so far…

- We discussed:
  - Serial schedules, serializability
  - Conflict-serializability, view-serializability
  - How to check for conflict-serializability
  - Recoverability, cascade-less schedules

- We haven't discussed:
  - How to guarantee serializability ?
    - Allowing transactions to run, and then aborting them if the schedules wasn't serializable is clearly not the way to go
  - We instead use schemes to guarantee that the schedule will be conflict-serializable