Transactions

Dr. Shrutilipi Bhattacharjee, Assistant Professor, Dept. of IT, NIT Karnataka, India

Serializability

- **Basic Assumption:** Each transaction preserves database consistency
- Thus, serial execution of a set of transactions preserves database consistency
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule
- Different forms of schedule equivalence give rise to the notions of:
  - **Conflict serializability**
  - **View serializability**

Simplified View of Transactions

- We ignore operations other than **read** and **write** instructions
  - Other operations happen in memory (are temporary in nature) and (mostly) do not affect the state of the database
  - This is simplifying assumptions for analysis

- We assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes

- Our simplified schedules consist of only **read** and **write** instructions

Conflicting Instructions

- Instructions I_i and I_j of transactions T_i and T_j respectively

- Instructions I_i and I_j **conflict** if and only if there exists some item Q accessed by both I_i and I_j, and at least one of these instructions wrote Q
  - 1. I_i = **read**(Q), I_j = **read**(Q) → I_i and I_j don't conflict
  - 2. I_i = **read**(Q), I_j = **write**(Q) → They conflict
  - 3. I_i = **write**(Q), I_j = **read**(Q) → They conflict
  - 4. I_i = **write**(Q), I_j = **write**(Q) → They conflict

- Intuitively, a conflict between I_i and I_j forces a (logical) temporal order between them
  - If I_i and I_j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule

Conflict Serializability

- If a schedule *S* can be transformed into a schedule *S'* by a series of swaps of non-conflicting instructions, we say that *S* and *S'* are **conflict equivalent**

- We say that a schedule *S* is **conflict serializable** if it is conflict equivalent to a serial schedule

Conflict Serializability

- Schedule 3 can be transformed into Schedule 6, a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions
  - Swap T1.read(B) and T2.write(A)
  - Swap T1.read(B) and T2.read(A)
  - Swap T1.write(B) and T2.write(A)
  - Swap T1. write(B) and T2.read(A)

These swaps do not conflict as they work with different items (*A & B*) in different transaction

- Therefore Schedule 3 is conflict serializable

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Schedule 3

T_1	T_2
read (A)	
write (A)	
	read (A)
read (B)	
	write (A)
write (B)	
	read (B)
	write (B)

Schedule 5

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Schedule 6

Conflict Serializability

- Example of a schedule that is not conflict serializable:

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

- We are unable to swap instructions in the above schedule to obtain either the serial schedule < T_3, T_4 >, or the serial schedule < T_4, T_3 >

Example: Bad Schedule

- Consider two transactions:

Transaction 1

UPDATE accounts
SET balance = balance - 100
WHERE acct_id = 31414

Transaction 2

UPDATE accounts
SET balance = balance * 1.005

- In terms of read-write, we can write as follows:
  - Transaction 1: $r_1(A)$, $w_1(A)$ // A is the balance for acct_id = 31414
  - Transaction 2: $r_2(A)$, $w_2(A)$, $r_2(B)$, $w_2(B)$ // B is the balance of other accounts

- Consider schedule S
  - Schedule *S*: r_1(A), r_2(A), w_1(A), w_2(A), r_2(B), w_2(B)
  - Suppose: Account A starts with $200, and account B starts with $100
  - Schedule *S* is very bad! (At least, it's bad if you're the bank!) We withdrew $100 from account *A*, but somehow the database has recorded that our account now holds $201

	A	B
(initial:)	200.00	100.00
$r_1(A)$:		
$r_2(A)$:		
$w_1(A)$:	100.00	
$w_2(A)$:	201.00	
$r_2(B)$:		
$w_2(B)$:		100.50

Schedule S

Example: Bad Schedule

- Ideal schedule is **serial**: (A = \$200, B = \$100)
  - Serial schedule 1: $r_1(A)$, $w_1(A)$, $r_2(A)$, $w_2(A)$, $r_2(B)$, $w_2(B)$ // A = 100.50, B = 100.50
  - Serial schedule 2: $r_2(A)$, $w_2(A)$, $r_2(B)$, $w_2(B)$, $r_1(A)$, $w_1(A)$ // A = 101.00, B = 100.50

- We call a schedule **serializable** if it has the same effect as some serial schedule regardless of the specific information in the database

- As an example, consider Schedule T, which has swapped the third and fourth operations from S:
  - Schedule S: $r_1(A)$, $r_2(A)$, $w_1(A)$, $w_2(A)$, $r_2(B)$, $w_2(B)$
  - Schedule T: $r_1(A)$, $r_2(A)$, $w_2(A)$, $w_1(A)$, $r_2(B)$, $w_2(B)$

- Looking just at the first example, we see that the outcome is the same as the serial schedule where the withdrawal happens first and then the interest is credited

- But that's just a peculiarity of the data, as revealed by the second example, where the final value of A can't be the consequence of either of the possible serial schedules

A is \$100 initially

	A	*B*
(initial:)	100.00	100.00
$r_1(A)$:		
$r_2(A)$:		
$w_2(A)$:	100.50	
$w_1(A)$:	0.00	
$r_2(B)$:		
$w_2(B)$:		100.50

A is \$200 initially

	A	*B*
(initial:)	200.00	100.00
$r_1(A)$:		
$r_2(A)$:		
$w_2(A)$:	201.00	
$w_1(A)$:	100.00	
$r_2(B)$:		
$w_2(B)$:		100.50

Example: Good Schedule

- What's a non-serial example of a serializable schedule?
  - We could credit interest to *A* first, then withdraw the money, then credit interest to *B*:
  - Schedule *U*: $r_2(A)$, $w_2(A)$, $r_1(A)$, $w_1(A)$, $r_2(B)$, $w_2(B)$

- Schedule *U* is conflict-equivalent to Serial Schedule 2, as shown by the following series of swaps:

Schedule *U*:	$r_2(A)$, $w_2(A)$, $r_1(A)$, $w_1(A)$, $r_2(B)$, $w_2(B)$
swap $w_1(A)$ and $r_2(B)$:	$r_2(A)$, $w_2(A)$, $r_1(A)$, $r_2(B)$, $w_1(A)$, $w_2(B)$
swap $w_1(A)$ and $w_2(B)$:	$r_2(A)$, $w_2(A)$, $r_1(A)$, $r_2(B)$, $w_2(B)$, $w_1(A)$
swap $r_1(A)$ and $r_2(B)$:	$r_2(A)$, $w_2(A)$, $r_2(B)$, $r_1(A)$, $w_2(B)$, $w_1(A)$
swap $r_1(A)$ and $w_2(B)$:	$r_2(A)$, $w_2(A)$, $r_2(B)$, $w_2(B)$, $r_1(A)$, $w_1(A)$
	= Schedule 2

Serializability

- Are all serializable schedules conflict-serializable? No

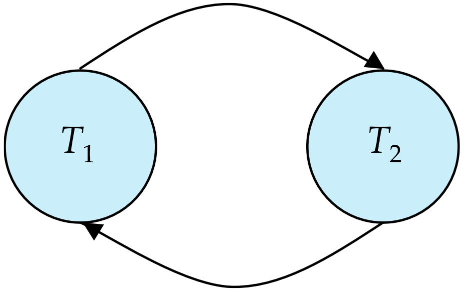
- Consider the following schedule for a set of three transactions:
  - $w_1(A)$, $w_2(A)$, $w_2(B)$, $w_1(B)$, $w_3(B)$

- We can perform no swaps to this:
  - The first two operations are both on *A* and at least one is a write
  - The second and third operations are by the same transaction
  - The third and fourth are both on *B* at at least one is a write, and
  - So are the fourth and fifth
  - So this schedule is not conflict-equivalent to anything else- and certainly not any serial schedules

- However, since nobody ever reads the values written by the $w_1(A)$, $w_2(B)$, and $w_1(B)$ operations, the schedule has the same outcome as the serial schedule:
  - $w_1(A)$, $w_1(B)$, $w_2(A)$, $w_2(B)$, $w_3(B)$

Precedence Graph

- Consider some schedule of a set of transactions T_1, T_2, ..., T_n
- **Precedence graph**
  - A directed graph where the vertices are the transactions (names)
- We draw an arc from T_i to T_j if the two transaction conflict, and T_i accessed the data item on which the conflict arose earlier
- We may label the arc by the item that was accessed
- Example of a precedence graph

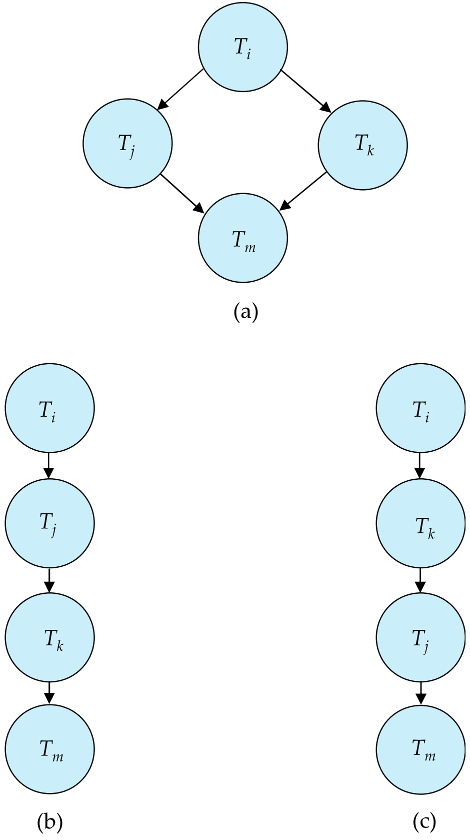


Test for Conflict Serializability

- A schedule is conflict serializable if and only if its precedence graph is acyclic

- Cycle-detection algorithms exist which take order n^2 time, where n is the number of vertices in the graph
  - Better algorithms take order $n + e$ where e is the number of edges

- If precedence graph is acyclic, the serializability order can be obtained by a *topological sorting* of the graph
  - This is a linear order consistent with the partial order of the graph
  - For example, a serializability order for Schedule (a) would be one of either (b) or (c)



Test for Conflict Serializability

- Build a directed graph, with a vertex for each transaction

- Go through each operation of the schedule
  - If the operation is of the form $w_i(X)$, find each subsequent operation in the schedule also operating on the same data element X by a different transaction, that is, anything of the form $r_j(X)$ or $w_j(X)$
    - For each such subsequent operation, add a directed edge in the graph from T_i to T_j

  - If the operation is of the form $r_i(X)$, find each subsequent *write* to the same data element X by a different transaction, that is, anything of the form $w_j(X)$
    - For each such subsequent write, add a directed edge in the graph from T_i to T_j

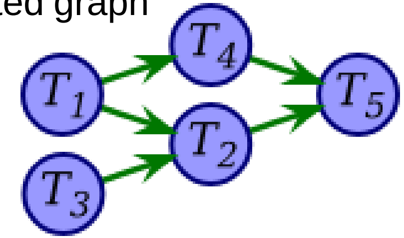
- The schedule is conflict-serializable if and only if the resulting directed graph is acyclic

Test for Conflict Serializability

- Consider the following schedule:
  - $w_1(A), r_2(A), w_1(B), w_3(C), r_2(C), r_4(B), w_2(D), w_4(E), r_5(D), w_5(E)$
- We start with an empty graph with five vertices labeled T_1, T_2, T_3, T_4, T_5
- We go through each operation in the schedule:

$w_1(A)$:	A is subsequently read by T_2, so add edge $T_1 \rightarrow T_2$
$r_2(A)$:	no subsequent writes to A, so no new edges
$w_1(B)$:	B is subsequently read by T_4, so add edge $T_1 \rightarrow T_4$
$w_3(C)$:	C is subsequently read by T_2, so add edge $T_3 \rightarrow T_2$
$r_2(C)$:	no subsequent writes to C, so no new edges
$r_4(B)$:	no subsequent writes to B, so no new edges
$w_2(D)$:	C is subsequently read by T_2, so add edge $T_3 \rightarrow T_2$
$w_4(E)$:	E is subsequently written by T_5, so add edge $T_4 \rightarrow T_5$
$r_5(D)$:	no subsequent writes to D, so no new edges
$w_5(E)$:	no subsequent operations on E, so no new edges

- We end up with the following directed graph



- This graph has no cycles, so the original schedule must be serializable
- Moreover, since one way to topologically sort the graph is T_3 - T_1 - T_4 - T_2 - T_5, one serial schedule that is conflict-equivalent is:
  - $w_3(C), w_1(A), w_1(B), r_4(B), w_4(E), r_2(A), r_2(C), w_2(D), r_5(D), w_5(E)$

Next Lecture

Transactions

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6th Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in