Database Management Systems

Module 47: Transactions/2: Serializability

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- A task in a database is done as a transaction that passes through several states
- Transactions are executed in concurrent fashion for better throughput
- Concurrent execution of transactions raise serializability issues that need to be addressed
- All schedules may not satisfy ACID properties

Module Objectives

- To understand the issues that arise when two or more transactions work concurrently
- To introduce the notions of Serializability that ensure schedules for transactions that may run in concurrent fashion but still guarantee and serial behavior
- To analyze the conditions, called conflicts, that need to be honored to attain Serializable schedules

Module Outline

- Serializability
- Conflict Serializability

Serializability

Serializability

- **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:
  a) **Conflict Serializability**
  b) **View Serializability**

Reacp Schedule 3: Serializable

- Let T_1 and T_2 be the transactions defined previously. The following schedule is not a serial schedule, but it is **equivalent** to Schedule 1

Schedule 3

T_1	T_2
read (A) $A := A - 50$ write (A)	
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A)
read (B) $B := B + 50$ write (B) commit	
	read (B) $B := B + temp$ write (B) commit

Schedule 1

T_1	T_2
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit

A	B	A+B	Transaction	Remarks
100	200	300	@ Start	
50	200	250	T1, write A	
45	200	245	T2, write A	
45	250	295	T1, write B	@ Commit
45	255	300	T2, write B	@Commit

Consistent @ Commit
Inconsistent @ Transit
Inconsistent @ Commit

Note: In schedules 1, 2 and 3, the sum "$A + B$" is preserved

Recap Schedule 4: Not Serializable

- The following concurrent schedule does not preserve the sum of "$A + B$"

T_1	T_2
read (A) $A := A - 50$	
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B)
write (A) read (B) $B := B + 50$ write (B) commit	
	$B := B + temp$ write (B) commit

A	B	A+B	Transaction	Remarks
100	200	300	@ Start	
90	200	290	T2, write A	
50	200	250	T1, write A	
50	250	300	T1, write B	@ Commit
50	210	260	T2, write B	@ Commit

Consistent @ Commit
Inconsistent @ Transit
Inconsistent @ Commit

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 a simplifying assumption 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

- Let I_i and I_j be two Instructions from 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 write to Q
  a) $I_i = \textbf{read}(Q)$, $I_j = \textbf{read}(Q)$. I_i and I_j don't conflict
  b) $I_i = \textbf{read}(Q)$, $I_j = \textbf{write}(Q)$. They conflict
  c) $I_i = \textbf{write}(Q)$, $I_j = \textbf{read}(Q)$. They conflict
  d) $I_i = \textbf{write}(Q)$, $I_j = \textbf{write}(Q)$. 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

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 (2)

- Schedule 3 can be transformed into Schedule 6, a serial schedule where T_2 follows T_1, by a 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 or B) in different transactions

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 (3)

- 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

	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

- In terms of read / write we can write these as:
  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 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:* 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!

Example: Bad Schedule (2)

- Ideal schedule is serial:
  Serial schedule 1:
  $r_1(A), w_1(A), r_2(A), w_2(A), r_2(B), w_2(B)$
  Serial schedule 2:
  $r_2(A), w_2(A), r_2(B), w_2(B), r_1(A), w_1(A)$
- 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 : $r_1(A), r_2(A), w_2(A), w_1(A), r_2(B), w_2(B)$

T1	Schedule 1: T1-T2		Schedule 2: T2-T1	
T2	A	B	A	B
Initial Value	200.00	100.00	200.00	100.00
Final Value	100.00	100.00	201.00	100.50
Initial Value	100.00	100.00	201.00	100.50
Final Value	100.50	100.50	101.00	100.50

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

Schedule T

- By first example, the outcome is the same as Serial schedule 1. 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.
- So neither S nor T are serializable

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 $U : r_2(A), w_2(A), r_1(A), w_1(A), r_2(B), w_2(B)$
    - Initial: $A = 200, B = 100$
    - Final: $A = 101, B = 100.50$
- Schedule U is conflict serializable to Schedule 2:

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

Source: *Serializability*

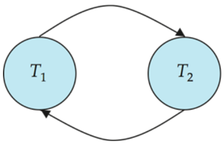
Serializability

- 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 least one is a write; and
  - So are the fourth and fifth.
  - So this schedule is not conflict-equivalent to anything – 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:
  - $w_1(A), w_1(B), w_2(A), w_2(B), w_3(B)$

Source: *Serializability*

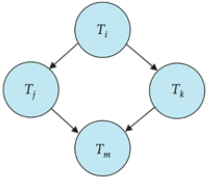
Precedence Graph

- Consider some schedule of a set of transactions $T_1, T_2, \cdots, T_n$
- **Precedence Graph**
  - A direct graph where the vertices are the transactions (names)
- We draw an arc from T_i to T_j if the two transactions 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**

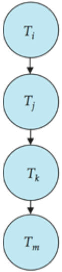


Testing for Conflict Serializability

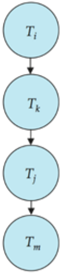
- *A schedule is conflict serializable if and only if its precedence graph is acyclic*
- Cycle-detection algorithms exist which take order n^2 time, where n is the number of vertices in the graph
  - (Better algorithms take order $n + e$ where e is the number of edges)
- If precedence graph is acyclic, the serializability order can be obtained by a *topological sorting* of the graph
  - That is, a linear order consistent with the partial order of the graph.
  - For example, a serializability order for the schedule (a) would be one of either (b) or (c)



(a)



(b)

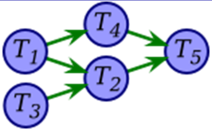


(c)

Testing for Conflict Serializability (2)

- 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.
- Moreover, we can perform a topological sort on the graph to discover the serial schedule to which the schedule is conflict-equivalent.

Testing for Conflict Serializability (3)



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

Module Summary

- Understood the issues that arise when two or more transactions work concurrently
- Learnt the forms of serializability in terms of conflict and view serializability
- Acyclic precedence graph can ensure conflict serializability

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".