Database Management Systems

Transaction Processing

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
and
Centre for Artificial Intelligence and Machine Learning
Indian Statistical Institute, Kolkata

April, 2022

Basics

What is a transaction?

A unit of program execution that accesses and possibly updates various data items.

The properties (briefed as ACID) of a transaction maintained by the database system to ensure integrity of the data:

- Atomicity: None or all operations of the transaction are reflected properly in the database.
- Consistency: The database consistency is preserved by the execution of a transaction with no other transaction executing concurrently.
- Isolation: If multiple transactions execute concurrently, the system guarantees that for every transaction pair it appears one of them starts execution after the other finishes.
- Durability: Changes in database after the successful completion of a transaction are retained, even if there are system failures.

An example

Suppose, 10 PCs are transferred from the *PC* attribute of the relation ISI to relation IISc.

The transaction (consisting of six instructions) required for the above operation is as follows:

- I read(ISI_{PC})
- II $\text{ISI}_{PC} \leftarrow \text{ISI}_{PC}$ - 10
- III write(ISI_{PC})
- IV read(IISc_{PC})
- V $\text{IISc}_{PC} \leftarrow \text{IISc}_{PC} + 10$
- VI write(IISc_{PC})

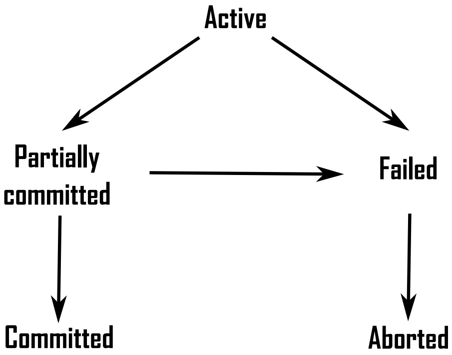
Note: We have to deal with system failures and manage concurrent execution of multiple instructions.

An example

Let us see how the ACID properties are managed:

- Atomicity: If the system fails at the steps 4-5 then this partial execution will not be incorporated.
- Consistency: If at any step the system fails then also the sum of ISI_{PC} and IISc_{PC} should be same.
- Isolation: If any other transaction working on ISI and IISc appears, while executing the steps 3-6, it will wait until the current transaction completes.
- Durability: Once the steps 1-6 are executed the database changes will persist.

Transaction life cycle



State transition diagram

Why concurrent execution of transactions?

- Increased processor and disk utilization
- Better transaction throughput
- Reduced waiting time
- Reduced average response time for transactions – short transactions will not wait behind longer ones

Scheduling of transactions

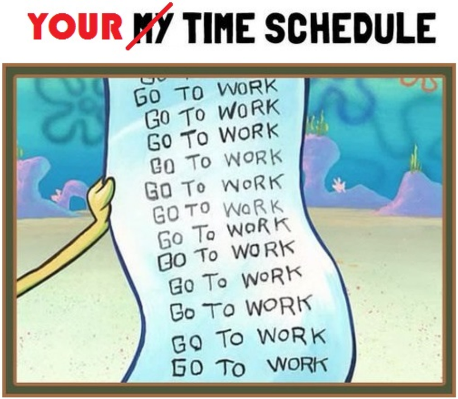
A schedule is a sequence of instructions that specify the chronological order in which instructions of concurrent transactions are executed

Some properties of scheduling:

- A schedule for a set of transactions should comprise all instructions of those transactions
- A schedule should retain the order in which the instructions appear in each individual transaction
- A transaction completing successful execution should have a commit instruction as the last statement
- A transaction that fails to successfully complete its execution should have an abort instruction as the last statement

Note: The number of possible schedules for a set of n transactions is much larger than $n!$.

Importance of a schedule



Scheduling of transactions – Example 1

Serial schedule T_1 is followed by T_2:

	Transaction T_1	Transaction T_2	ISI_{PC}=20, $IISc_{PC}$=40
IN01	read(ISI_{PC})		
IN02	$T \leftarrow ISI_{PC} * 0.05$		T=1
IN03	$ISI_{PC} \leftarrow ISI_{PC} - T$		
IN04	write(ISI_{PC})		ISI_{PC}=19, $IISc_{PC}$=40
IN05	read($IISc_{PC}$)		
IN06	$IISc_{PC} \leftarrow IISc_{PC} + T$		
IN07	write($IISc_{PC}$)		ISI_{PC}=19, $IISc_{PC}$=41
IN08	commit		ISI_{PC}=19, $IISc_{PC}$=41
IN09		read(ISI_{PC})	
IN10		$ISI_{PC} \leftarrow ISI_{PC} - 10$	
IN11		write(ISI_{PC})	ISI_{PC}=9, $IISc_{PC}$=41
IN12		read($IISc_{PC}$)	
IN13		$IISc_{PC} \leftarrow IISc_{PC} + 10$	
IN14		write($IISc_{PC}$)	ISI_{PC}=9, $IISc_{PC}$=51
IN15		commit	ISI_{PC}=9, $IISc_{PC}$=51

Scheduling of transactions – Example 2

Serial schedule T_2 is followed by T_1:

	Transaction T_1	Transaction T_2	ISI_{PC}=20, $IISc_{PC}$=40
IN01		read(ISI_{PC})	
IN02		$ISI_{PC} \leftarrow ISI_{PC}$ - 10	
IN03		write(ISI_{PC})	ISI_{PC}=10, $IISc_{PC}$=40
IN04		read($IISc_{PC}$)	
IN05		$IISc_{PC} \leftarrow IISc_{PC} + 10$	
IN06		write($IISc_{PC}$)	ISI_{PC}=10, $IISc_{PC}$=50
IN07		commit	ISI_{PC}=10, $IISc_{PC}$=50
IN08	read(ISI_{PC})		
IN09	$T \leftarrow ISI_{PC}$ * 0.05		T=0.5
IN10	$ISI_{PC} \leftarrow ISI_{PC}$ - T		
IN11	write(ISI_{PC})		ISI_{PC}=9.5, $IISc_{PC}$=50
IN12	read($IISc_{PC}$)		
IN13	$IISc_{PC} \leftarrow IISc_{PC} + T$		
IN14	write($IISc_{PC}$)		ISI_{PC}=9.5, $IISc_{PC}$=50.5
IN15	commit		ISI_{PC}=9.5, $IISc_{PC}$=50.5

Scheduling of transactions – Example 3

Not a serial schedule but equivalent to T_2 is followed by T_1:

	Transaction T_1	Transaction T_2	$ISI_{PC}=20$, $IISc_{PC}=40$
IN01		read(ISI_{PC})	
IN02		$ISI_{PC} \leftarrow ISI_{PC}$ - 10	
IN03		write(ISI_{PC})	$ISI_{PC}=10$, $IISc_{PC}=40$
IN04	read(ISI_{PC})		
IN05	$T \leftarrow ISI_{PC}$ * 0.05		$T=0.5$
IN06	$ISI_{PC} \leftarrow ISI_{PC}$ - T		
IN07	write(ISI_{PC})		$ISI_{PC}=9.5$, $IISc_{PC}=40$
IN08		read($IISc_{PC}$)	
IN09		$IISc_{PC} \leftarrow IISc_{PC} + 10$	
IN10		write($IISc_{PC}$)	$ISI_{PC}=9.5$, $IISc_{PC}=50$
IN11		commit	$ISI_{PC}=9.5$, $IISc_{PC}=50$
IN12	read($IISc_{PC}$)		
IN13	$IISc_{PC} \leftarrow IISc_{PC} + T$		
IN14	write($IISc_{PC}$)		$ISI_{PC}=9.5$, $IISc_{PC}=50.5$
IN15	commit		$ISI_{PC}=9.5$, $IISc_{PC}=50.5$

Scheduling of transactions – Example 4

Not a serial schedule and also inconsistent:

	Transaction T_1	**Transaction** T_2	ISI_{PC}=20, $IISc_{PC}$=40
IN01		read(ISI_{PC})	
IN02		$ISI_{PC} \leftarrow ISI_{PC}$ - 10	
IN03	read(ISI_{PC})		
IN04	$T \leftarrow ISI_{PC}$ * 0.05		T=1
IN05	$ISI_{PC} \leftarrow ISI_{PC}$ - T		
IN06	write(ISI_{PC})		ISI_{PC}=19, $IISc_{PC}$=40
IN07	read($IISc_{PC}$)		
IN08		write(ISI_{PC})	ISI_{PC}=19, $IISc_{PC}$=40
IN09		read($IISc_{PC}$)	
IN10		$IISc_{PC} \leftarrow IISc_{PC} + 10$	
IN11		write($IISc_{PC}$)	ISI_{PC}=19, $IISc_{PC}$=50
IN12		commit	ISI_{PC}=19, $IISc_{PC}$=50
IN13	$IISc_{PC} \leftarrow IISc_{PC} + T$		
IN14	write($IISc_{PC}$)		ISI_{PC}=19, $IISc_{PC}$=51
IN15	commit		ISI_{PC}=19, $IISc_{PC}$=51

Serializability

Assumption: Each transaction preserves database consistency.

So, the serial execution of a set of transactions should preserve the database consistency.

A schedule is serializable if it is equivalent to a serial schedule

Different forms of schedule equivalence give rise to the notions of – `conflict serializability` and `view serializability`. For both the cases our main concern is the read/write operation.

Note: We consider only read() and write() instructions to verify serializability.

Conflict serializability

Definition (Conflict)

Instructions I_i and I_j of transactions T_i and T_j respectively, conflict if and only if there exists some item Q accessed by both I_i and I_j and at least one of them is a write instruction.

Definition (Conflict equivalent)

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.

Definition (Conflict serializable)

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

Understanding conflict equivalence

	Transaction T_1	Transaction T_2
↑		read(ISI_{PC})
Block1		$ISI_{PC} \leftarrow ISI_{PC}$ - 10
↓		write(ISI_{PC})
↑	read(ISI_{PC})	
Block2	$T \leftarrow ISI_{PC}$ * 0.05	
.	$ISI_{PC} \leftarrow ISI_{PC}$ - T	
↓	write(ISI_{PC})	
↑		read($IISc_{PC}$)
Block3		$IISc_{PC} \leftarrow IISc_{PC} + 10$
.		write($IISc_{PC}$)
↓		commit
↑	read($IISc_{PC}$)	
Block4	$IISc_{PC} \leftarrow IISc_{PC} + T$	
.	write($IISc_{PC}$)	
↓	commit	

Consider swapping the instructions between the Blocks 2 and 3.

Conflict serializability – Example 1

Transaction T_1	**Transaction** T_2
	read(ISI_{PC})
	write(ISI_{PC})
read(ISI_{PC})	
write(ISI_{PC})	
	read($IISc_{PC}$)
	write($IISc_{PC}$)
read($IISc_{PC}$)	
write($IISc_{PC}$)	

Conflict serializability – Example 1

Transaction T_1	Transaction T_2
	read(ISI_{PC})
	write(ISI_{PC})
read(ISI_{PC})	
write(ISI_{PC})	
	read($IISc_{PC}$)
	write($IISc_{PC}$)
read($IISc_{PC}$)	
write($IISc_{PC}$)	

The above schedule is conflict serializable because it is equivalent to the following serial schedule.

Transaction T_1	Transaction T_2
	read(ISI_{PC})
	write(ISI_{PC})
	read($IISc_{PC}$)
	write($IISc_{PC}$)
read(ISI_{PC})	
write(ISI_{PC})	
read($IISc_{PC}$)	
write($IISc_{PC}$)	

Conflict serializability – Example 2

The following schedule is not conflict serializable because it is not equivalent to any serial schedule. Note that, the conflicting instructions write(ISI_{PC}) in both the transactions can not be swapped.

Transaction T_1	**Transaction T_2**
	read(ISI_{PC})
write(ISI_{PC})	
	write(ISI_{PC})

View serializability

Definition (View equivalent)

Let S and S' be two schedules with the same set of transactions. S and S' are view equivalent if the following three conditions are met, for each data item Q:

- If in schedule S, transaction T_i reads the initial value of Q, then in schedule S' also transaction T_i must read the initial value of Q.
- If in schedule S transaction T_i executes read(Q), and that value was produced by transaction T_j, then in schedule S' also transaction T_i must read the value of Q that was produced by the same write(Q) operation of transaction T_j .
- The transaction (if any) that performs the final write(Q) operation in schedule S must also perform the final write(Q) operation in schedule S'.

View serializability

Definition (View serializable)

A schedule S is view serializable if it is view equivalent to a serial schedule.

Note: A conflict serializable schedule is always view serializable but not the vice versa.

View serializability – An example

Transaction T_1	Transaction T_2	Transaction T_3
	read(ISI_{PC})	
write(ISI_{PC})		
	write(ISI_{PC})	
		write(ISI_{PC})

View serializability – An example

Transaction T_1	Transaction T_2	Transaction T_3
	read(ISI_{PC})	
write(ISI_{PC})		
	write(ISI_{PC})	
		write(ISI_{PC})

The above schedule is view serializable because it is equivalent to the following serial schedule.

Transaction T_1	Transaction T_2	Transaction T_3
	read(ISI_{PC})	
	write(ISI_{PC})	
write(ISI_{PC})		
		write(ISI_{PC})

Note: The top schedule is not conflict serializable because the conflicting instructions write(ISI_{PC}) both in T_1 and T_2 cannot be swapped to obtain a serial schedule.

Testing for conflict serializability

We can test conflict serializability through constructing precedence graphs.

Definition (Precedence graph)

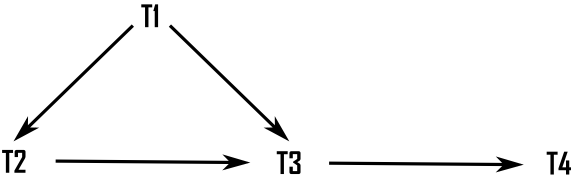
Given a schedule S, a precedence graph is defined as a directed graph $G = (V, E)$, where the set of vertices V consists of all the transactions participating in S and E consists of all the edges $T_i \rightarrow T_j$ for which one of three conditions holds in S:

1. T_i executes write(Q) before T_j executes read(Q).
2. T_i executes read(Q) before T_j executes write(Q).
3. T_i executes write(Q) before T_j executes write(Q).

Testing for conflict serializability

The precedence graph for a conflict serializable schedule is always acyclic.

The following graph corresponds to a conflict serializable schedule because it is acyclic. Notably, $T1 \rightarrow T2 \rightarrow T3$ is not a cycle.



Note: A directed graph is acyclic if it has no cycle (a sequence of non-repeating directed edges except for the first and last one).

Testing for conflict serializability

In general, cycle-detection algorithms incur $O(n^2)$ time, where n is the order of the graph. However, there exists better algorithms incurring $O(n + e)$ time, where e denotes the size of the graph.

From an acyclic precedence graph, the serializability order can be obtained by a topological sorting of the graph.