BITS, PILANI – K. K. BIRLA GOA CAMPUS

Database Systems

(IS F243)

by

Mrs. Shubhangi Gawali

Dept. of CS and IS

TRANSACTION MANAGEMENT

Example serial schedule 1: T1 is followed by T2

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

Example serial schedule 2: T2 is followed by T1

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

Example Schedule (Cont.)

- 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.

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)	
	read(B)
	$B := B + temp$
	write(B)

In both Schedules , the sum A + B is preserved.

Example Schedules (cont...)

This concurrent schedule does not preserve the value of the the sum *A* + *B*. (WW and RW conflict)

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)	
	$B := B + temp$
	write(B)

Example Schedules (cont...)

This concurrent schedule does not preserves the value of the the sum *A* + *B*. (RW Conflict)

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

Example Schedules (cont...)

This concurrent schedule does not preserve the value of the the sum *A* + *B*. (WR Conflict)

T_1	T_2
read(A)	
$A := A - 50$	
$\boxed{\text{write}(A)}$	
	$\boxed{\text{read}(A)}$
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
read(B)	
$B := B + 50$	
write(B)	

Anomalies due to interleaved executions

- Reading Uncommitted data (WR Conflicts) (Dirty read problem) (due to temporary updates)
  - Eg: T2 reads the data updated by T1 and T1 fails before completing. Thus T2 reads dirty data.
- Unrepeatable Reads (RW Conflicts)(Incorrect summary problem)
  - Eg: giving bonus and transferring funds to/from same account
  - Eg: finding total seats available and cancelling ticket for same train.
- Overwriting Uncommitted Data (WW Conflicts)(The Lost Update Problem)
  - Eg: transferring then funds and debiting the amount from same account

Scheduling Transactions

- *Serial schedule:* Schedule that does not interleave the actions of different transactions.
- *Equivalent schedules*: For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.
- *Serializable schedule*: A schedule that is equivalent to some serial execution of the transactions.

(Note: If each transaction preserves consistency, every serializable schedule preserves consistency.)

Conflict Serializability

- Instructions l_i and l_j of transactions T_i and T_j respectively, **conflict** if and only if there exists some item Q accessed by both l_i and l_j, and at least one of these instructions wrote Q.

1. l_i = **read**(Q), l_j = **read**(Q). l_i and l_j don't conflict.
2. l_i = **read**(Q), l_j = **write**(Q). They conflict.
3. l_i = **write**(Q), l_j = **read**(Q). They conflict
4. l_i = **write**(Q), l_j = **write**(Q). They conflict

Conflict Serializability

- 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 (cont...)

- 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

Example schedule

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)	
	read(B)
	$B := B + temp$
	write(B)

Conflict Serializability (cont...)

This Schedule can be transformed into Schedule 1, a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions. Therefore this schedule is conflict serializable.

T1	T2
Read (A)	
Write (A)	
	Read (A)
	Write (A)
Read (B)	
Write (B)	
	Read (B)
	Write (B)

Conflict Serializability (cont…)

This Schedule can be transformed into Schedule 1, a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions. Therefore this schedule is conflict serializable.

T1	T2
Read (A)	
Write (A)	
	Read (A)
Read (B)	
	Write (A)
Write (B)	
	Read (B)
	Write (B)

Conflict Serializability (cont...)

This Schedule can be transformed into Schedule 1, a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions. Therefore this schedule is conflict serializable.

T1	T2
Read (A)	
Write (A)	
	Read (A)
Read (B)	
Write (B)	
	Write (A)
	Read (B)
	Write (B)

Conflict Serializability (cont...)

This Schedule can be transformed into Schedule 1, a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions. Therefore this schedule is conflict serializable.

T1	T2
Read (A)	
Write (A)	
Read (B)	
	Read (A)
Write (B)	
	Write (A)
	Read (B)
	Write (B)

Conflict Serializability (cont…)

This Schedule can be transformed into Schedule 1, a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions. Therefore this schedule is conflict serializable.

T1	T2
Read (A)	
Write (A)	
Read (B)	
Write (B)	
	Read (A)
	Write (A)
	Read (B)
	Write (B)

Example : non conflict serializable schedule

- 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 >.

Exercise : check whether the schedule is conflict serializable or not

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)

Testing for Serializability

- Consider some schedule of a set of transactions T_1, T_2, …, T_n
- **Precedence graph** — a direct graph where the vertices are the transactions (names).
- Draw an arc from T_i to T_k if the two transaction conflict (i.e one of the transaction writes to same data item), and T_i accessed the data item on which the conflict arose earlier.
- Label the arc by the item that was accessed.

Precedence Graph

- Given a schedule S, involving transactions T1 and T2, perhaps among other transactions, we say that T1 takes precedence over T2, written as T1 <T2, if there are actions A1 of T1 and A2 of T2, such that:

1. A1 is ahead of A2 in S,
2. Both A1 and A2 involve the same database element, and
3. At least one of A1 and A2 is a write action

Precedence Graph for
(a) Schedule 1 and (b) Schedule 2

Example

	T1	T2	T3
t1		R(Z)	
t2		R(Y)	
t3		W(Y)	
t4			R(Y)
t5			R(Z)
t6	R(X)		
t7	W(X)		
t8			W(Y)
t9			W(Z)
t10		R(X)	
t11	R(Y)		
t12	W(Y)		
t13		W(X)	

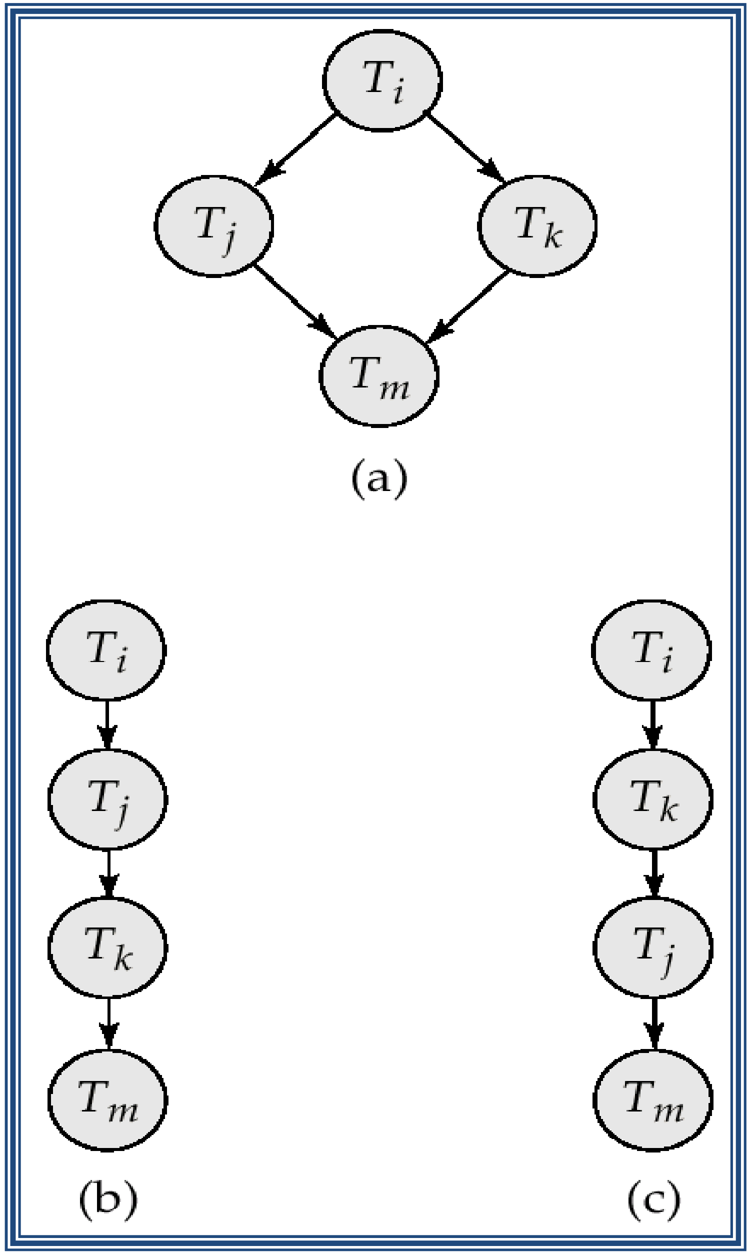
Test for Conflict Serializability

- **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.

Illustration of Topological Sorting



View Serializability

- 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:
  1. For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule $S´$, also read the initial value of Q.
  2. For each data item Q if transaction T_i executes **read**(Q) in schedule S, and that value was produced by transaction T_j (if any), then transaction T_i must in schedule $S´$ also read the value of Q that was produced by transaction T_j .
  3. For each data item Q, the transaction (if any) that performs the final **write**(Q) operation in schedule S must perform the final **write**(Q) operation in schedule $S´$.

As can be seen, view equivalence is also based purely on **reads** and **writes** alone.

Blind writes

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

View Serializability (Cont.)

- A schedule S is **view serializable** it is view equivalent to a serial schedule.
- Every conflict serializable schedule is also view serializable.
- Schedule below - a schedule which is view-serializable but *not* conflict serializable.

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

- Every view serializable schedule that is not conflict serializable has **blind writes.**

Recoverability

- Need to address the effect of transaction failures on concurrently running transactions.

- **Recoverable schedule** — if a transaction T_j reads a data items previously written by a transaction T_i , the commit operation of T_i appears before the commit operation of T_j.
- The following schedule is not recoverable if *T8* commits immediately after write(A)

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

- If T_8 needs to abort, T_9 would have read (and possibly shown to the user) an inconsistent database state. Hence database must ensure that schedules are recoverable.

Recoverability (Cont.)

Cascading rollback – a single transaction failure leads to a series of transaction rollbacks.
Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

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

- If T_{10} fails, T_{11} and T_{12} must also be rolled back.
- Can lead to the undoing of a significant amount of work

Recoverability (Cont.)

- **Cascadeless schedules** — cascading rollbacks cannot occur; for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.

- Every cascadeless schedule is also recoverable.

- It is desirable to restrict the schedules to those that are cascadeless.