Lecture 44 & 45

Validation Based Protocols (Optimistic Concurrency Control Scheme)

Validation Based Protocols

- In cases where a majority of transactions are read-only transactions, the rate of conflicts among transactions may be low.
- Thus, many of these transactions, if executed without the supervision of a concurrency-control scheme, would nevertheless leave the system in a consistent state.
- A concurrency-control scheme imposes overhead of code execution and possible delay of transactions. It may be better to use an alternative scheme that imposes less overhead.
- A difficulty in reducing the overhead is that we do not know in advance which transactions will be involved in a conflict. To gain that knowledge, we need a scheme for **monitoring** the system.

Validation Based Protocols

- We assume that each transaction T_i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. The phases are, in order,
  a) **Read phase**. During this phase, the system executes transaction T_i. It reads the values of the various data items and stores them in variables local to T_i. It performs all write operations on temporary local variables, without updates of the actual database.
  b) **Validation phase**. Transaction T_i performs a validation test to determine whether it can copy to the database the temporary local variables that hold the results of write operations without causing a violation of serializability.
  c) **Write phase**. If transaction T_i succeeds in validation, then the system applies the actual updates to the database. Otherwise, the system rolls back T_i.
- Each transaction must go through the three phases in the order shown. However, all three phases of concurrently executing transactions can be interleaved.

Validation Based Protocols

- To perform the validation test, we need to know when the various phases of transactions Ti took place. We shall, therefore, associate three different timestamps with transaction T_i :
  a) **Start**(T_i), the time when T_i started its execution.
  b) **Validation**(T_i), the time when T_i finished its read phase and started its validation phase.
  c) **Finish**(T_i), the time when T_i finished its write phase.

Validation Based Protocols

- We determine the serializability order by the timestamp-ordering technique, using the value of the timestamp Validation(T_i).
- Thus, the value TS(T_i) = Validation(T_i) and, if TS(T_j) < TS(T_k), then any produced schedule must be equivalent to a serial schedule in which transaction T_j appears before transaction T_k.
- The reason we have chosen Validation(T_i), rather than Start(T_i), as the timestamp of transaction T_i is that we can expect faster response time provided that conflict rates among transactions are indeed low.

Validation Based Protocols

- The **validation test** for transaction T_i requires that, for all transactions T_j with $TS(T_j) < TS(T_i)$, one of the following two conditions must hold :
  a) Finish(T_j) < Start(T_i). Since T_j completes its execution before T_i started, the serializability order is indeed maintained.
  b) The set of data items written by T_j does not intersect with the set of data items read by T_i, and T_j completes its write phase before T_i starts its validation phase (Start(T_i) < Finish(T_j) < Validation(T_i)). This condition ensures that the writes of T_j and T_i do not overlap. Since the writes of T_j do not affect the read of T_i , and since T_i cannot affect the read of T_j, the serializability order is indeed maintained.

Validation Based Protocols Example

- Consider the transactions T1 and T2. Suppose that TS(T1) < TS(T2). Then, the validation phase succeeds in the schedule in figure below.
- Note that the writes to the actual variables are performed only after the validation phase of T2. Thus, T1 reads the old values of B and A, and this schedule is serializable.

T1	T2
Read(B)	
	Read(B)
	B :=B – 50
	Read(A)
	A :=A + 50
Read(A)	
<validate>	
Display(A+B)	
	<validate>
	Write(A)
	Write(B)

Validation Based Protocols Example

T1 (2)	T2 (1)
	Begin
	Read(A)
Begin	
Read(A)	
	<validate>
	Write(A)
<validate>	
Write(A)	

T1 (1)	T2 (2)
Begin	
Read(A)	
	Begin
	Read(A)
	<validate>
	Write(A)
<validate>	
Write(A)	

Validation Based Protocols

- The validation scheme automatically guards against cascading rollbacks, since the actual writes take place only after the transaction issuing the write has committed.
- However, there is a possibility of starvation of long transactions, due to a sequence of conflicting short transactions that cause repeated restarts of the long transaction.
- To avoid starvation, conflicting transactions must be temporarily blocked, to enable the long transaction to finish.

Validation Based Protocols

- This validation scheme is called the **optimistic concurrency control** scheme since transactions execute optimistically, assuming they will be able to finish execution and validate at the end.
- In contrast, locking and timestamp ordering are pessimistic in that they force a wait or a rollback whenever a conflict is detected, even though there is a chance that the schedule may be conflict serializable.

Validation Based Protocols

Advantages **:** The advantages of validation based protocols are:

1. It maintains serializability.
2. It is free from cascading rollback.
3. Less overhead then other protocols.

Disadvantages **:** The disadvantages of validation based protocols are:

1. Starvation of long transactions due to conflicting short transactions.

Graph Based Protocol

- The two-phase locking protocol is both necessary and sufficient for ensuring serializability in the absence of information concerning the manner in which data items are accessed.
- But, if we wish to develop protocols that are not two phase, we need additional information on how each transaction will access the database.
- There are various models that can give us the additional information, each differing in the amount of information provided.
- The simplest model requires that we have prior knowledge about the order in which the database items will be accessed.
- Given such information, it is possible to construct locking protocols that are not two phase, but that, nevertheless, ensure conflict serializability.

Graph Based Protocol

- To acquire such prior knowledge, we impose a partial ordering → on the set **D** = {d_1, d_2, . . ., d_h} of all data items. If $d_i \rightarrow d_j$, then any transaction accessing both d_i and d_j must access *di* before accessing d_j .
- This partial ordering may be the result of either the logical or the physical organization of the data, or it may be imposed solely for the purpose of concurrency control.
- The partial ordering implies that the set **D** may now be viewed as a directed acyclic graph, called a **database graph**.

Tree based protocol

- In the **tree protocol**, the only lock instruction allowed is lock-X.
- Each transaction T_i can lock a data item at most once, and must observe the following rules :
  1. The first lock by T_i may be on any data item.
  2. Subsequently, a data item Q can be locked by T_i only if the parent of Q is currently locked by T_i.
  3. Data items may be unlocked at any time.
  4. A data item that has been locked and unlocked by T_i cannot subsequently be relocked by T_i.
- All schedules that are legal under the tree protocol are conflict serializable.

Tree Based Protocols

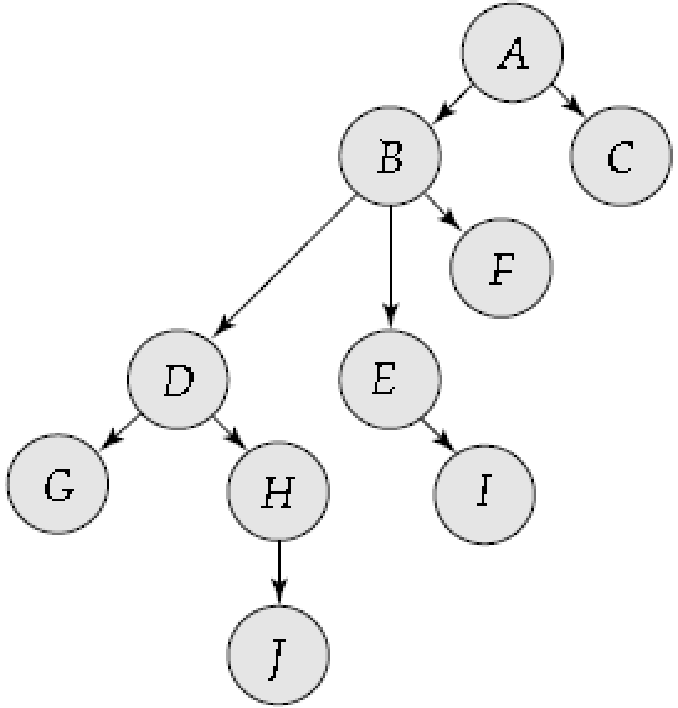
- Consider the database graph of Figure below.
- The following four transactions follow the tree protocol on this graph. We show only the lock and unlock instructions:

  *T*1: lock-X(*B*); lock-X(*E*); lock-X(*D*); unlock(*B*); unlock(*E*); lock-X(*G*); unlock(*D*); unlock(*G*).

  *T*2: lock-X(*D*); lock-X(*H*); unlock(*D*); unlock(*H*).

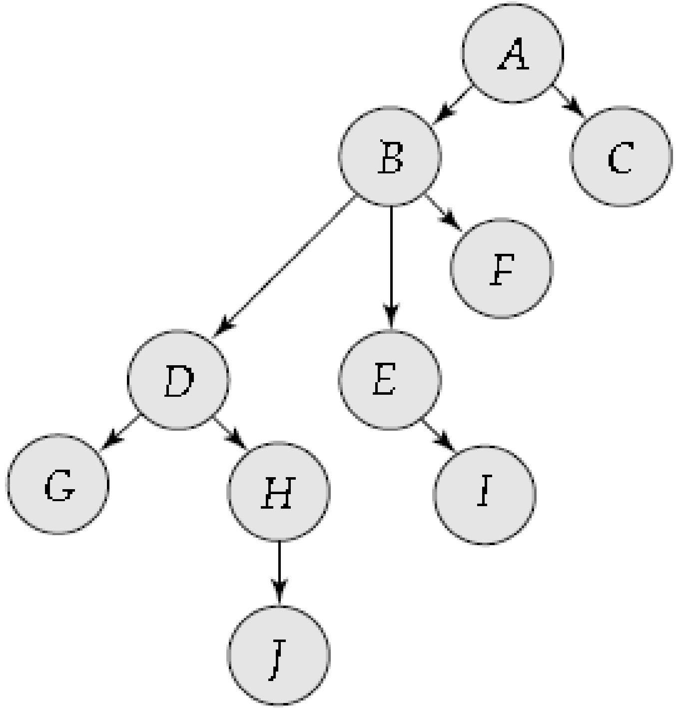
  *T*3: lock-X(*B*); lock-X(*E*); unlock(*E*); unlock(*B*).

  *T*4: lock-X(*D*); lock-X(*H*); unlock(*D*); unlock(*H*).



Tree Based Protocols

T1	T2	T3	T4
Lock-X(B)			
	Lock-X(D)		
	Lock-X(H)		
	Unlock(D)		
Lock-X(E)			
Lock-X(D)			
Unlock(B)			
Unlock(E)			
		Lock-X(B)	
		Lock-X(E)	
	Unlock(H)		
Lock-X(G)			
Unlock(D)			
			Lock-X(D)
			Lock-X(H)
			Unlock(D)
			Unlock(H)
		Unlock(E)	
		Unlock(B)	
Unlock(G)			



*T*1: lock-X(*B*); lock-X(*E*); lock-X(*D*); unlock(*B*); unlock(*E*); lock-X(*G*); unlock(*D*); unlock(*G*).
*T*2: lock-X(*D*); lock-X(*H*); unlock(*D*); unlock(*H*).
*T*3: lock-X(*B*); lock-X(*E*); unlock(*E*); unlock(*B*).
*T*4: lock-X(*D*); lock-X(*H*); unlock(*D*); unlock(*H*).

Tree Based Protocol

***Advantages* :** The advantages of tree based protocol are:

1. It maintains conflict serializability.

2. It is free from deadlock..

3. Locks can be unlocked anytime.

***Disadvantages* :** The disadvantages of validation based protocols are:

1. Unnecessary locking overheads may happen sometimes.
2. ***Cascading Rollbacks*** is still a problem.

For Video lecture on this topic please subscribe to my youtube channel.

The link for my youtube channel is

https://www.youtube.com/channel/UCRWGtE76JlTp1iim6aOTRuw?sub_confirmation=1