MODULE 5

TIME STAMP BASED ALGORITHMS

- Here every site maintains a logical clock that is incremented when a transaction is submitted at that site and updated whenever the site receives a message with a higher clock value(every message contains the current clock value of its sender site).
- Each transaction is assigned a unique timestamp and conflicting actions are executed in the order of the timestamp of their transactions.
- A timestamp is generated by appending the local clock time with the site identifier.

Timestamp can be used in two ways

- First,they can be used to determine the currency or outdatedness of a request with respect to the data object it is operating on.
- Second,they can be used to order events(read-write) with respect to another.In timestamp based concurrency control algorithms,the serialization order of transactions is selected apriori and transactions are forced to follow this order

1.Basic Time Stamp Ordering(BTO) Algorithm

- Every transaction is issued a timestamp based on when it enters the system. Suppose, if an old transaction T_i has timestamp $TS(T_i)$, a new transaction T_j is assigned timestamp $TS(T_j)$ such that **$TS(T_i) < TS(T_j)$**. The protocol manages concurrent execution such that the timestamps determine the serializability order. The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. Whenever some Transaction *T* tries to issue a R_item(X) or a W_item(X), the Basic TO algorithm compares the timestamp of *T* with **R_TS(X) & W_TS(X)** to ensure that the Timestamp order is not violated. This describes the Basic TO protocol in the following two cases.

1. Whenever a Transaction *T* issues a **W_item(X)** operation, check the following conditions:
    1. If ***R_TS(X) > TS(T)*** or if ***W_TS(X) > TS(T)***, then abort and rollback T and reject the operation. else,
    2. Execute W_item(X) operation of T and set W_TS(X) to TS(T).
2. Whenever a Transaction *T* issues a **R_item(X)** operation, check the following conditions:
    1. If ***W_TS(X) > TS(T)***, then abort and reject T and reject the operation, else
    2. If W_TS(X) <= TS(T), then execute the R_item(X) operation of T and set R_TS(X) to the larger of TS(T) and current R_TS(X).

Thomas write Rule(TWR)

- Thomas Write Rule provides the guarantee of serializability order for the protocol. It improves the Basic Timestamp Ordering Algorithm.
- The basic Thomas write rules are as follows:
- If TS(T) < R_TS(X) then transaction T is aborted and rolled back, and operation is rejected.
- If TS(T) < W_TS(X) then don't execute the W_item(X) operation of the transaction and continue processing.
- If neither condition 1 nor condition 2 occurs, then allowed to execute the WRITE operation by transaction Ti and set W_TS(X) to TS(T)

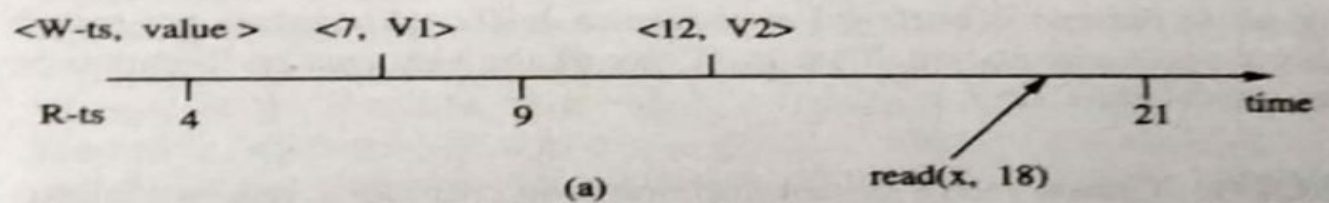
Multiversion Timestamp Ordering Algorithm

- In the multiversion timestamp ordering(MTO) algorithm,
- a history of a set of R-ts ‘s and <W-ts,value> pairs(called versions) is kept for each data object at the respective DM’s.
- The R-ts’s of a data object keep track of the timestamps of all the executed read operations
- The versions keep track of the timestamp and the value of all the executed write operations.

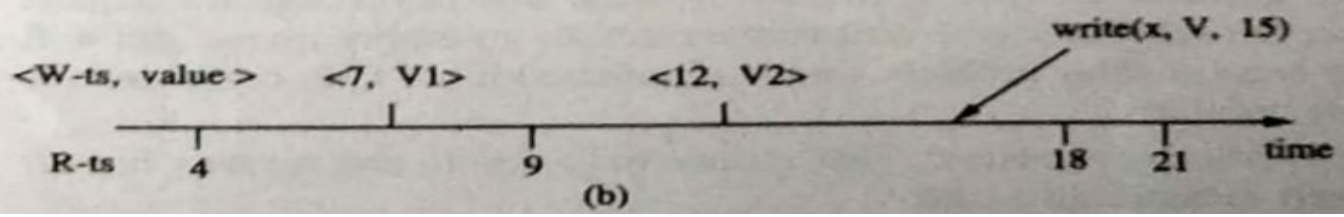
- A read(x, TS) request is executed by reading the version of x with the largest timestamp less than TS and adding TS to the x's set of R-ts's. A read request is never rejected.

Example 20.7. In Fig. 20.7(a), a read(x, 18) is executed by reading the version <12, V2> and the resulting history is shown in Fig. 20.7(b).

- A write(x, v, TS) request is executed in the following way: If there exists a R-ts(x) in the interval from TS to the smallest W-ts(x) that is larger than TS, then the write is rejected, otherwise it is accepted and a new version of x is created with time-stamp TS.



(a)



(b)

Example 20.8. In Figure 20.7(b), write(x, V, 15) is rejected because a read with timestamp 18 has already been executed. However, a write(x, V, 22) is accepted and is executed by creating a version <22, V> in the history.

CONSERVATIVE TIME STAMP ORDERING ALGORITHM

- **System Components:**
- A timestamp ordering scheme for every site (i.e, a transaction manager, TM) in the system.
- A scheduler process that keeps track of all the transaction requests arriving at a certain data manager (DM). This scheduler also maintains two internal queue data structures for each TM, namely a READ and a WRITE queue. These queues, as the name suggests, hold READ and WRITE requests for a TM and are ordered by using the timestamp ordering scheme previously mentioned.

Read

A read (x,TS) request is executed in the following way.

If (every W(Q) is non empty && first write on each W –queue has a timestamp)>TS

read is executed

otherwise read(x,TS) request is executed in the R-queue

Write

A read (x,v,TS) request is executed in the following way.

(If all R queues and all W queues are non empty and first read on each R-queue has a timestamp >TS)

Then Write is executed

Otherwise write(x,v,TS) request is buffered in the queue.

When any read or write request is buffered or executed, buffered requests are tested to see if any of them can be executed. That is, if any of the requests in R-queue or W-queue satisfies condition 1 or 2.

PROBLEMS WITH CTO. Conservative timestamp ordering technique has two major problems.

- Termination is not guaranteed. This is because if some TM never sends a request to some scheduler, the scheduler will wait forever due to an empty queue and will never execute any request. This problem can be eliminated if all TMs communicate with all schedulers regularly
- The algorithm is overly conservative; That is, not only conflicting actions but all actions are executed in timestamp order.

Optimistic Algorithm

- Optimistic Concurrency Control Algorithms are based on the assumption that conflicts do not occur during execution time.
- No synchronization is performed when a transaction is executed. However, a check is performed at the end of the transaction to make sure that no conflicts have occurred.
- If there is a conflict, the transaction will be aborted.
- Otherwise, the transaction is committed.
- Since conflicts do not occur very often, this algorithm is very efficient compared to other locking algorithms.

Conflicts in DBMS

- **Conflicting operations:** Two operations are said to be conflicting if all conditions satisfy:
- They belong to different transactions
- They operate on the same data item
- At Least one of them is a write operation

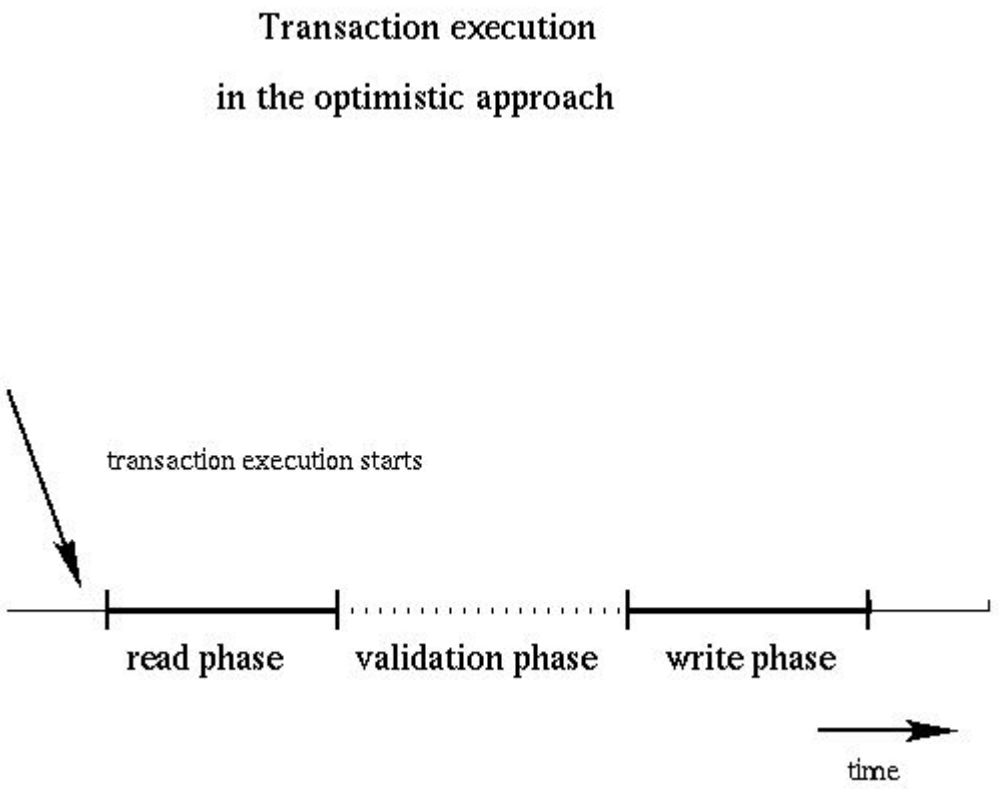
- Example: –
- **Conflicting** operations pair ($R_1(A)$, $W_2(A)$) because they belong to two different transactions on same data item A and one of them is write operation.
- Similarly, ($W_1(A)$, $W_2(A)$) and ($W_1(A)$, $R_2(A)$) pairs are also **conflicting**.
- On the other hand, ($R_1(A)$, $W_2(B)$) pair is **non-conflicting** because they operate on different data item.
- Similarly, (($W_1(A)$, $W_2(B)$) pair is **non-conflicting.**

Kung-Robinson Algorithm

- Kung and Robinson were the first to propose an optimistic method for concurrency control.
- The optimistic situation for this algorithm happens when
  - conflicts are unlikely(not probable) to happen
  - the system consists mainly read-only transactions
- Basic Idea: No synchronization check is performed during transaction processing time, however, a validation is performed to make sure there is no conflicts occurred. If a conflict is found, the tentative write is discarded and the transaction is restarted.

The algorithm

Divide the execution of transaction into three phases:



- **read phase:** data objects are read, the intended computation of the transaction is done, and writes are made on a **temporary** storage.
  validation phase: check to see if writes made by the transaction violate the consistency of the database. If the check finds out any conflicts, the data in the temporary storage will be discarded.
- Otherwise, **the write phase** will write the data into the **database**.

T: a transaction
ts: the highest sequence number at the start of T
tf: the highest sequence number at the beginning of its validation phase
After the read phase of transaction T,the following algorithm is executed in a mutually exclusive manner

valid:=true;

for t:=ts+1 to tf do

if (writeset(t) & readset[T] != {}) then valid :=false;

if valid then {write phase; increment counter, assign T a sequence number}

write phase: If the validation phase passes ok, write will be performed to the database. If the validation phase fails to pass, all temporary written data will be aborted.