Chapter 7.8 Concurrent Control by using Time Stamp

In next step, consider some other methods to guarantee Serializable Transactions besides Lock Schedule and used in some Systems.

1. *Time Stamp* - *Assign ‘Time Stamp’ for every Transaction. The Time Stamp is used to record the last time to read and write each Database Element.* Compare the ‘Time Stamp’ of each Database Element with the ‘Time Stamp’ of the current Transaction, ensure the equality of Serializable Transaction and Actual Transaction Schedule.
2. *Validation* - *When committing one Transaction, check the Time Stamp of the current Transaction and the Database Element: This process is called ‘Validation’ of Transaction.* We need to ensure the equality of Serializable Transaction and Actual Transaction Schedule.

*Two methods are optimistic methods, when there do has some problems, and optimistic methods would choose to abort and re-start the Transaction while conversely, Lock Schedule would delay Transaction, but not abort them.* Normally, some Read Transactions are better than Lock Schedule, since these Transactions themselves would never cause Non - Serializable Behavior.

Chapter 7.8.1 Time Stamp

*Definition:*

In order to use Time Stamp as the method to control Concurrency Control, Schedule needs to assign one *Unique Number* on each Transaction T, which is to say *Time Stamp TS(T)*. The Time Stamp must be sent out ascending when the first time Transaction notifies the Schedule. There have two methods to generate Time Stamp:

1. *Use System Time as Time Stamp, as long as Schedule Operation would not assign Time Stamp to two Transactions so quick in only one Clock Period.*
2. *Schedule tries to maintain one Counter. Each time when the Transaction starts, and adds one on the counter, but the new value would become the Time Stamp of this Transaction. We need to know that the ‘Time Stamp’ has no relation with Real ‘Time’, but they have an important property: The ‘Time Stamp’ that the Transaction starts later is much higher than the earlier Transaction.*

No matter use which method to generate Time Stamp, and Schedule needs to maintain one active Schedule Table and its ‘Time Stamp’ Table.

*No matter to use which method to generate ‘Time Stamp’, we need to relate each Database Element X with two Time Stamp and one additional byte:*

* *RT(X), the read time of Database Element X, it’s the highest time stamp among all reading Transactions.*
* *WT(X), the write time of Database Element X, it’s the highest time stamp among all writing Transactions.*
* *C(X), the committing byte of Database Element X, it’s value equals to True, only when the latest Transaction has been committed.*

This byte is used to avoid such situation when Transaction T reads Database Element A which is written by Transaction U, at such time, Transaction T aborts.

The problem that *Transaction T reads ‘Uncommitted Data Element’ may cause inconsistent status of Database System Status, but any Schedule needs the Mechanism to avoid the ‘Dirty Read’.*

Chapter 7.8.2 Behavior that can not realized in Reality

*Background:*

In order to understand Architecture and Rule of ‘Time Stamp’ Schedule, we need to keep in mind that: Schedule assumes that the ‘Time Stamp’ Sequence of Transaction must be their Execution Serializable Sequence. Therefore, the task of the Schedule is besides assign Time Stamp and update the RT, WT, and C, it needs to check whether no matter when Read and Write happens, if each Transaction is executed at the ‘Time Stamp’, any actual happen things would happen. If not, we can say that this behavior is Non - Realizable actually. Two possible problems may happen:

1. *Too Late to Read:*

|  |  |
| --- | --- |
| Database Element X | WT(X) |
| RT(X) |
| C(X) |
| Transaction T1 | TS(T) |
| Op: Read(X) |

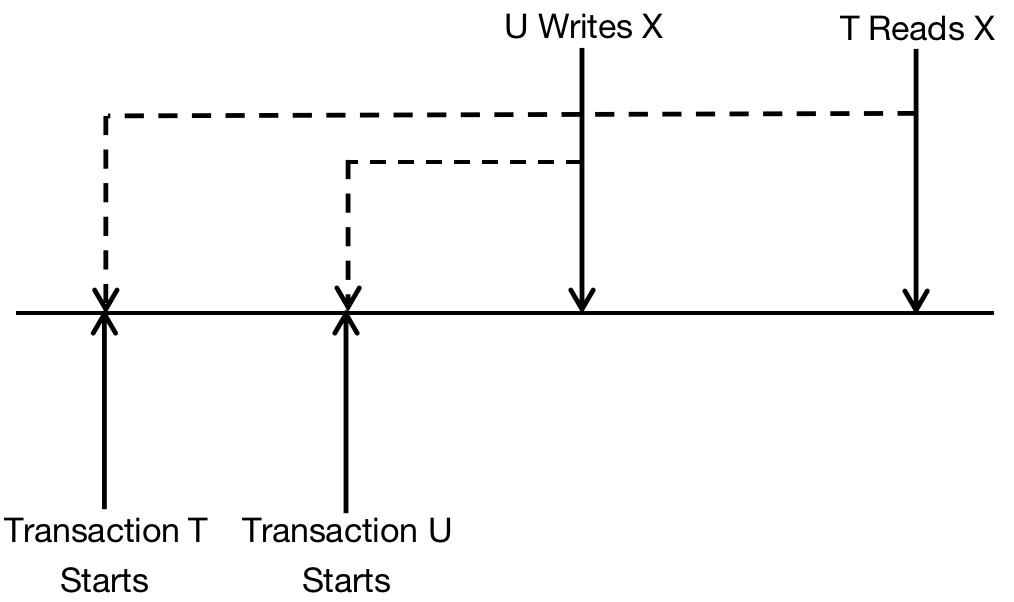
Under the situation which is described above, when TS(T) < WT(X) - which means that Transaction T1 tries to read Database Element X, and only finds that Database Element X has been written after the last recorded Time Stamp at which Database Element X has been written.

*We can draw the pic as below to describe the situation:*

* *Theoretically*, Transaction T and U all operates on Database Element X, and according to the condition described above, TS(T) < WT(X), which means that Write Operation is much early than Read Operation. Transaction T reads Database Element X after Transaction U has written Database Element X.
* *Actually*, Transaction T should not read the Database Element X which is written by Transaction U, since Transaction U should be executed after Transaction T. However, Transaction T has no choice, since now Transaction T can read Database Element X which is written by Transaction U.

*Solution:*

The way to solve this problem is to *abort Transaction T*.



1. *Too Late to Write:*

Transaction T tries to write on Database Element X. The read time of Database Element X signals that some other Transactions should read the value that Transaction T writes but it reads the Database Element X that other Transaction writes. Which is to say, *WT(X) < TS(T) < RT(X)*.

|  |  |
| --- | --- |
| Database Element X | WT(X) |
| RT(X) |
| C(X) |
| Transaction T1 | TS(T) |
| Op: Write(X) |

*Analysis:*

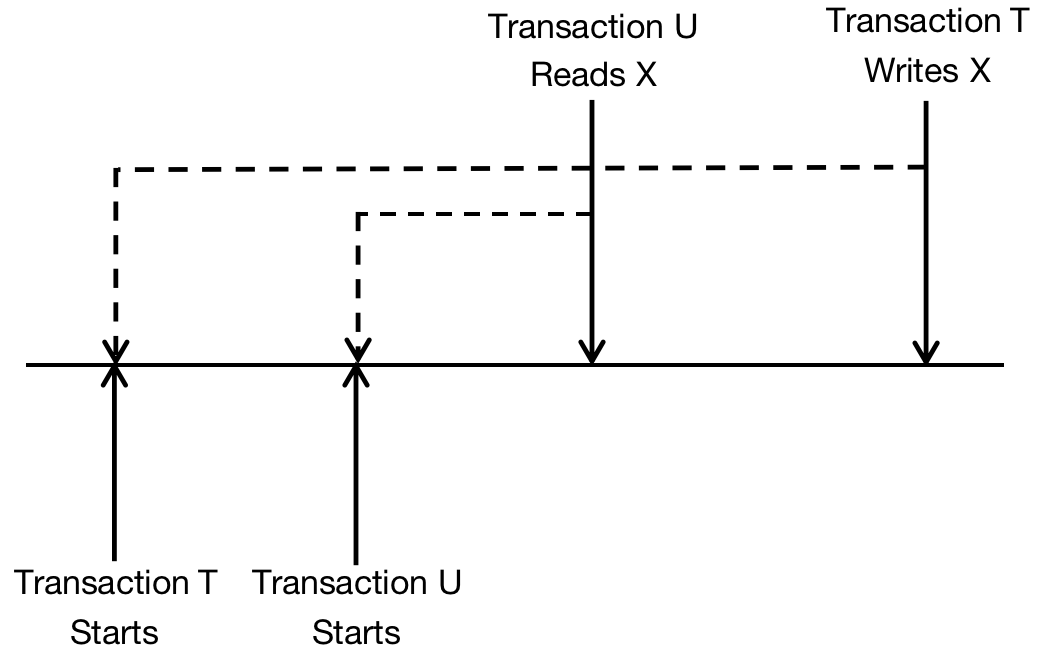
* *WT(X) < TS(T) means:*

Transaction T1 rewrite X after Database Element X has been written by one Transaction which is presented by WT(X).

* *TS(T) < RT(X) means:*

Database Element X has been read by another Transaction, just after X has been written by TS(T).

*The pic below describes such situation:*



*Explanation:*

In the pic above, Transaction U reads the Database Parameter X before Transaction T writes it. However, TS(T) < RT(X) means that Transaction T should be executed far before Transaction U but Transaction U reads the Database Parameter X before the Transaction T writes X. This is used to explain situation.

*Solution:*

The only solution is to *abort Transaction T*.

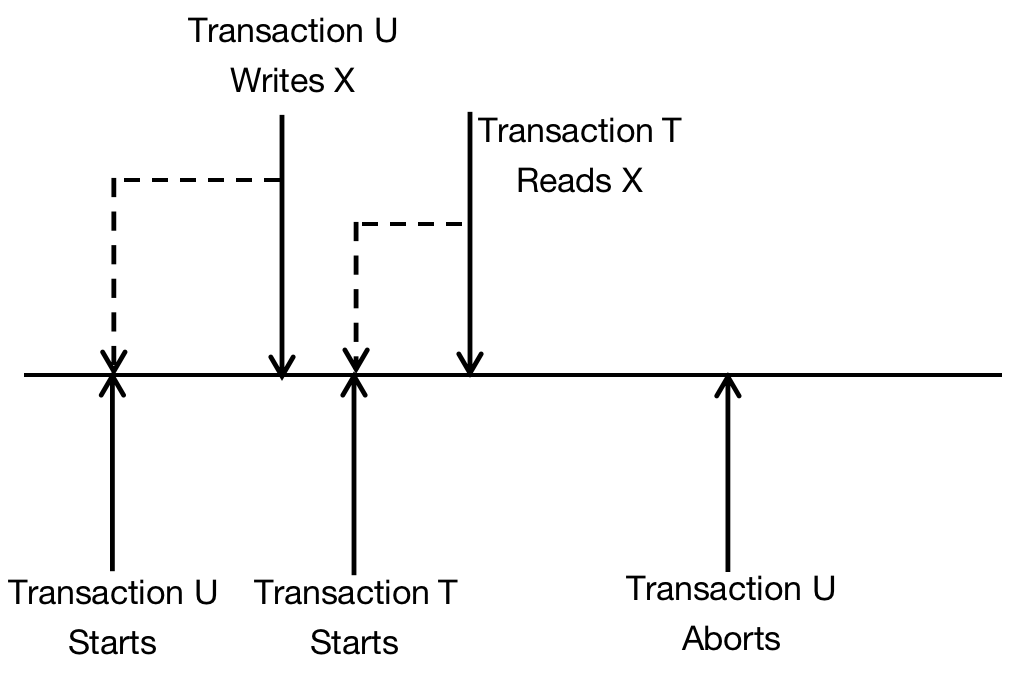
Chapter 7.8.3 The Problem of Dirty Data

*Definition:*

The additional byte is used to solve one type of problem.

*First Problem - Solve ‘Dirty Data’.*

The Transaction T read Database Parameter X, but Database Parameter X is written by Transaction U. The ‘Time Stamp’ of Transaction U is less than the ‘Time Stamp’ of Transaction T, and in the reality, Read operation of Transaction T is after Write Operation of Transaction U, therefore, this fact seems can be realized. However, there is possibility that after Transaction T reads the Database Parameter X, the Transaction U aborts. *(It is possible that the Transaction U meets one mistake, for example divided by 0, which is caused the Transaction U aborts.)*

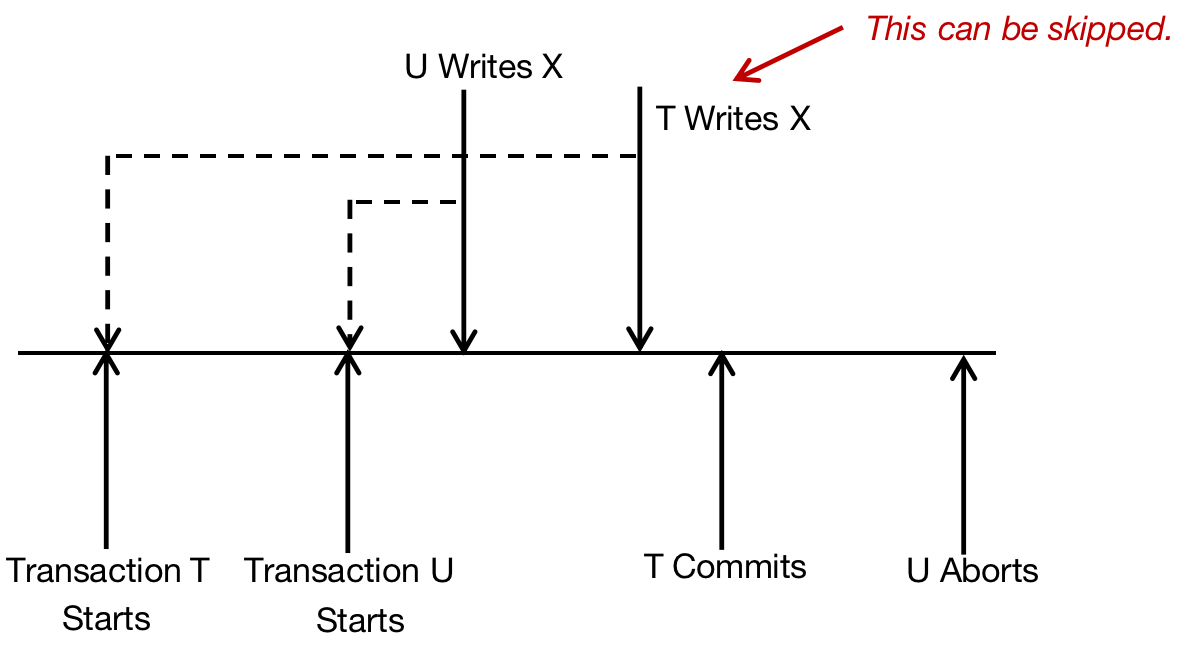
**

*Solution:*

*Under this kind of situation, it’s better to draw execution of Transaction T back to the commit of Transaction U or the abort of Transaction U. We can conclude that Transaction U has not been ended, so the Additional Byte C(X) equals to false.*

*Second Problem:*

Time Stamp of Transaction U is bigger than Time Stamp of Transaction T when writes Database Element X. When Transaction T tries to write, the correct action is to do nothing. This can be concluded as *When ‘Writing Operation’ happens after some other ‘Writing Operations’ which happen later, then the ‘Writing Operation’ can be eliminated, this is called ‘Thomas Write Principle’.*



*Hidden Trouble:*

However, there has one problem of *‘Thomas Write Principle’* - If Transaction U aborted, Database Parameter X should be deleted, and Old Value and Writing Time should be recovered. Since Transaction T has been committed, it seems that value of Database Element X should be read later. However, we can skip the Transaction T, since any failure can not be restored.

*Solution:*

Although there has multi - solutions to solve this problem, we need to get one simpler strategy, it based on the following schedule strategy that is based on *‘Time Stamp’*:

* *When Transaction T writes into Database System X, ‘Write Operation’ is tentative, and if the abortion of Transaction T happens, ‘Write Operation’ can be eliminated. At this time, the value of Additional Byte C(X) equals to false, the Schedule saves the old value of X and its original copy value of WT(X).*

Chapter 7.8.4 Rule that Schedule based on Time Stamp

*Generalization:*

In this Chapter, we generalize the rule that in order to guarantee there has not any un - realizable incidents, Schedule that is using ‘Time Stamp’ strategy should obey the rule. In order to react to any Write and Read requests, Schedule can choose below:

1. *Agree Request.*
2. *Abort Transaction T and restart the Transaction with new Time Stamp. (Normally, abortion adds restart are normally called Rollback.)*
3. *Delay Transaction T, and decide whether to abort the Transaction T or grant the Lock. ( If Request is Read and Read is ‘Dirty’. )*

*The Rule as below:*

1. Assume Schedule receives Request Rt(X).
2. If TS(T) >= WT(X), then Read Lock can be Realized.
   1. If C(X) is true, and this means that Latest Transaction for ‘Write Operation’ has been committed. Read Lock would be granted. If TS(T) > RT(X), then RT(X) = TS(T); Otherwise, we would not change RT(X).
   2. If C(X) is not true, and this means that Latest Transaction for ‘Write Operation’ has not been committed. Transaction T would be delayed till the value of C(X) equals to True or the Latest Transaction for ‘Write Operation’ has been aborted.
3. If TS(T) < WT(X), then Read Lock can not be Realized. We need to Rollback Transaction; Which is to say the Transaction T needs to be aborted and restart with another bigger Time Stamp.
4. Assume Schedule receives Request Wt(X).
5. If TS(T) >= RT(X), and TS(T) >= WT(X), then it means that this Transaction is realizable and must be executed.
   1. Write new value for Database Parameter X.
   2. Set WT(X) = TS(T).
   3. Set C(X) = false.
6. If TS(T) >= RT(X), and TS(T) < WT(X), then this Transaction can be realized, but there already has a late value on X. If C(X) equals to true, then the former write Transaction has been committed, and we can just neglect this value and continues as normal; If C(X) equals to false, then it means that the former write Transaction has not finished and we just need to delay Transaction T.
7. If TS(T) < RT(X), then it means that this Write Operation can not be realized, and Transaction T needs to be rollback.
8. Assume Schedule receives Request to Commit T. It must find all Database Element X, and set C(X) = true. After that we need to check another table to see whether there has any Transactions that is waiting for the Commit T and of course. If there has, then these Transactions can be continued to execute.
9. Assume Schedule receives Request to Abort T, so it can decide to rollback Transaction T just like 1b and 2c. But each Transaction that waits for the Transaction T writing to Database Element X needs to retry to read or write and check whether all these delayed Transactions can be legally executed.

*Example:*

There have one Schedule of three Transactions T1, T2 and T3, these three transactions visits three Database Element A, B, and C. The actual time of the Transaction Happening increase as the page goes down. We also indicate the ‘Time Stamp’ and read/write time of Element.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| T1 | T2 | T3 | A | B | C |
| 200 | 150 | 175 | RT(A) = 0;  WT(A) = 0; | RT(B) = 0;  WT(B) = 0; | RT(C) = 0;  WT(C) = 0; |
| r1(B); |  |  |  | RT(B) = 200; |  |
|  | r2(A); |  | RT(A) = 150; |  |  |
|  |  | r3(C); |  |  | RT(C) = 175; |
| w1(B); |  |  |  | WT(B) = 200; |  |
| w1(A); |  |  | WT(A) = 200; |  |  |
|  | w2(C); |  |  |  |  |
|  | Abort |  |  |  |  |
|  |  | w3(A); |  |  |  |

1. At the very start, each Database Element has its own Read and Write Time which are all 0. (The Transaction gets ‘Time Stamp’ when they notify start to the Schedule. ) Attention that, although the Transaction T1 execute the first Data access, but it has not the smallest Time Stamp. Here, assume that Transaction T2 is the first one to notify to start the Transaction, then the Transaction T3 and the last one is T1.
2. In the first action, Transaction T1 reads B. Since WT(B) is smaller than the ‘Time Stamp’ of Transaction T1, so RT(A) is assigned to 150. Similarly, the Read Time of second and third step, then the RT(B) = 200 and RT(C) = 175.
3. The forth step is Transaction T1 writes on Database Parameter B. Since WT(B) is less than the Time Stamp of Transaction T1, therefore update WT(B) = 200.
4. Transaction T2 tries to write on C. But Database C has already been read by Transaction T3. The Legal Execution Time of T3 equals to 150, but RT(C) = 175. Therefore, Transaction T2 tries to do un - realizable incident, therefore the Transaction T2 must be rollback.
5. Transaction T3 writes A. Since in Transaction T3, WT(A) = 175. There already has Transaction T1 has written on Database Parameter A and this Transaction happens later, WT(A) = 200. So according to ‘Thomas Write Principle’, write a in Transaction T3 can be neglected.

Chapter 7.8.5 Multi - Version Time Stamp

Chapter 7.8.6 Time Stamp and Lock

*Introduction:*

In most of situation, when Read Only or Concurrency Control tries to read and write on the same Database Element, Time Stamp has the great superiority. While under the situation with high conflict, Lock Schema has the great performance. The Demonstration for this Experience Law is that:

* *Lock Schema sometimes puts off the Transaction when Transaction is waiting for another Lock.*
* *If the Concurrency Transactions read and write the common Database Element Frequently, then rollback would also be frequent, which causes more delay than in Lock System.*

*Business Schema:*

Several Business does the interesting compromising. The Schedule divides the Transaction into Write Transaction and Read/Write Transaction. The Write/Read Transaction uses the two - phase Lock to execute, to avoid all transactions to visit each other’s Locked Database Element.

*Read - Only Transaction executes by using Multi - Version Time Stamp. Read - Only Transaction enables to read any version of Database Element. Therefore, read - only Transaction would not be banned, only be put off very little time.*