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) |

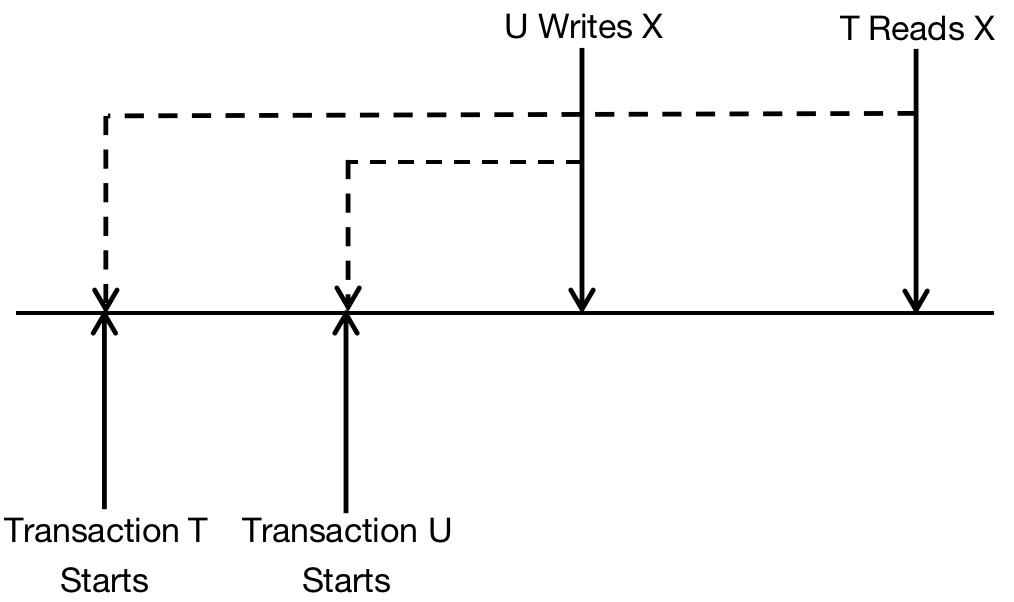
Under the situation which is described above, when TS(T) < WT(X) - which means that Transaction T1 tries to read Database Element X, and only finds that Database Element X has been written after the last recorded Time Stamp at which Database Element X has been written.

*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:*

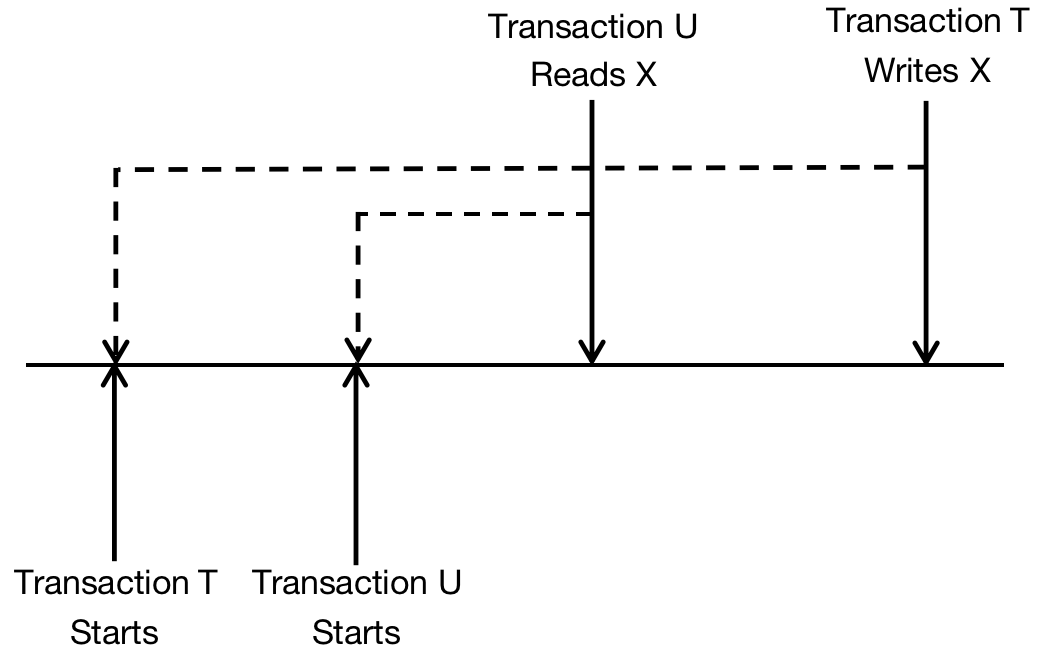
* *WT(X) < TS(T) means:*

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

Chapter 7.8.4 Rule to Schedule that based on Time Stamp

Chapter 7.8.5 Multi - Version Time Stamp

Chapter 7.8.6 Time Stamp and Lock