Lecture 43

Concurrency Control - Time Stamp Ordering

Timestamp based Concurrency Control Algorithm

A monotonically increasing variable (integer) indicating the age of an operation or a transaction. A larger timestamp value indicates a more recent event or operation.

Timestamp based algorithm uses timestamp to serialize the execution of concurrent transactions.

Basic Timestamp Ordering

1. Transaction T issues a write_item(X) operation:
   - If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then an younger transaction has already read the data item so abort and roll-back T and reject the operation.
   - If the condition in part (a) does not exist, then execute write_item(X) of T and set write_TS(X) to TS(T).

2. Transaction T issues a read_item(X) operation:
   - If write_TS(X) > TS(T), then an younger transaction has already written to the data item so abort and roll-back T and reject the operation.
   - If write_TS(X) ≤ TS(T), then execute read_item(X) of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

Strict Timestamp Ordering

1. Transaction T issues a write_item(X) operation:
   - If TS(T) > read_TS(X), then delay T until the transaction T' that wrote or read X has terminated (committed or aborted).

2. Transaction T issues a read_item(X) operation:
   - If TS(T) > write_TS(X), then delay T until the transaction T' that wrote or read X has terminated (committed or aborted).

Thomas's Write Rule

If read_TS(X) > TS(T)

then abort and roll-back T and reject the operation.

If write_TS(X) > TS(T),

then just ignore the write operation and continue execution. This is because the most recent writes counts in case of two consecutive writes.

If the conditions given in 1 and 2 above do not occur,

then execute write_item(X) of T and set write_TS(X) to TS(T).

Practice Example

Suppose user has a schedule in which two transactions T_1 and T_2. Now, **TS(T_1) < TS(T_2)**. This means T_1 arrived after T_2 and hence has a larger TS value than T_1. This implies that serializablity of schedule allowed is **T_2 –> T_1** . Consider the partial schedule given below:

	T_1	T_2
1.		R(A)
2.	W(A)	
3.		W(A)

$T_1 \rightarrow T_2$ which is not allowed. Ignore this Outdated Write operation according to Thomas Write Rule.

Allowed TS ordering $T_2 \rightarrow T_1$

Practice Example

Suppose user has a schedule in which two transactions T_1 and T_2. Now, **TS(T_1) < TS(T_2)**.
This implies that serializablity of schedule allowed is **T_2 –> T_1** . Consider the two protocols, let us see what types of Operation will be allowed and not allowed under them. **R_i(A)** implies Read and **W_i(A)** implies Write operation.

Basic TO Protocol

- **Not Allowed**
  - $R_1(X)\ W_2(X)$
  - $W_1(X)\ R_2(X)$
  - $W_1(X)\ W_2(X)$
- **Allowed**
  - All operations where T_2 occurs before T_1.
  - $R_1(X)\ R_2(X)$

Thomas Write Rule

- **Not Allowed**
  - $R_1(X)\ W_2(X)$
  - $W_1(X)\ R_2(X)$
- **Allowed**
  - All operations where T_2 occurs before T_1.
  - **Outdated Writes**: $W_1(X)\ W_2(X)$
  - $R_1(X)\ R_2(X)$

Practice Problem

Consider the following schedule:

T_1	T_2
$R(A)$ $R(B)$ $R(C)$	
	$R(A)$ $R(B)$ P: ______
Q: ______	
	$R(C)$

The possible values of P & Q for which the above schedule is allowed under Thomas write rule but not under basic timestamp ordering protocol if timestamp(T_2)>timestamp(T_1)

1. W(B),W(C)
2. W(A),W(A)
3. Both (a)&(b)
4. None of these

Practice Problem Solution

Option 1 : We haven't any W->W or W->R or R->W dependency from T2 to T1, So Option 1 is valid for Simple TimeStamp. and so, valid for TWR (bcz in TWR we can even ignore W->W dependency).

Option 2 : There will be W(A)->W(A) dependency from T2 to T1, which voilate Simple TimeStamp Rule but valid for TWR (bcz TWR will be voilate only if W->R or R->W dependency is here from T2 to T1)

So Option 2

Note: It should be given that Time-Stamp of T2 is greater than that of T1.

THANKS!!