Explain each line of the propose algorithm

**Show and explain a diagram that illustrate banking transactions**

* Example: Where X withdraw from acct-x and deposit in acct-y, the Y withdraw from acct-y
* Record the time of transaction X and Y. and the time interval should be in seconds
* Explain how the proposed algorithm efficiently handles the transactions.

1. if ts(Ci) = ts(wv[i])

{

2. Execute Ci,

3. Delete the contents of wv[i],

4. Move up the remaining values of wv

}

1. Let k is the index of the first empty available position

in cv.

2. For each received Commit operation Ci

{

3. If (ts(Ci) = wv[0]) or (ts(Ci) =wv[k])

{

4. Execute Ci;

5. Delete wv[0] Or Delete wv[k];

6. Move up all the remaining values in wv and cv

simultaneously

}

else

{

7. Record Ci at the first empty cell in cv

If a transaction T1 issues Write Operation W(Di).

1: if (Ts(Ti) => Ts(Di)){

2: Execute Ti

3: copy values of data Item to a private workspace place

4: do modifications in the private workplace and not to the database

5: goes to the next phase

6: }

7: if (Copy-Ts(Di) == Ts(Di)){

8: Active Transaction is assigned a new Timestamp Ts(Tv)

9: write operation is executed

10: copies the modified values to database (Update)

11: and sets Ts(D) = Ts(Tv)

12: }

13: else {

14: transaction is abort and rollback, its private workspace will also be cleared

15: }

16:

17:

18: }

If a transaction T1 issues Read Operation R(Di).

1: if(Ts(Ti) <= OR => Ts(Di)){

2: Execute Ti

3: read the values of data Item

4: }

**Thomas’ Write Rule Timestamp Ordering algorithm** resolves transaction conflicts early in their execution phase, with respect to the syntactic information provided.

**Optimistic Concurrency Control algorithm** it ensures that conflict between transactions are not common occurrences and it also allows simultaneous access to the same data until the validation phase.

**A modelling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure.**

**Optimistic concurrency control method for real time database sys**

In the read algorithm. Di is the data item to be read, Ti is the transaction reading the data item, T1(Ti) is the timestamp interval allocated to the transaction, WTS(Di) is the write timestamp of the data item, and RTS(Di) is the read timestamp of the data item. This algorithm is executed for all data item read in the transaction.

**In the read phase when a read operation is executed**, the write timestamp (WTS) of the data item accessed is verified against the timestamp in interval allocated to the transaction. If another transaction has written on the data item outside the timestamp interval, the transaction must be restarted.

**In the read phase when a write operation is executed,** the modification is made to a local copy of the data item. A pre-write operation is used to verify read and write timestamps of the write data item against the timestamp interval allocated to the transaction. If another transaction has read or written on the data item outside the timestamp interval, the transaction must be restarted.

However we can see OPTH is not exactly OCCA, because some conflict checking is already done in the read phase. The idea behind this is to detect non-serializable execution as soon as possible and restart non-serializable transaction early. This will prevent unnecessary execution of the non-serializable transaction to its validation phase, the avoiding resource wasting.

**Optimistic concurrency control method for real time database sys**

* **In read phase:** every transaction is assigned a timestamp Ts(T) at the start of execution, which is used to record temporary serialization order induced during the execution of the transaction. The Ts(T) is use to compare Ts(D), to ensure that the timestamp order of transaction execution is not violated. If this order is violated, then transaction is aborted and is rollback and start as a new transaction with a new timestamp

else transaction read the value from data item to a private workspace area, does the computations and makes local copies of its modifications in the private workplace only and not to the database.

* **In validation phase**: Timestamp is given at V stage, where transaction has to be validated to check if the Ts(D) in the private workplace area is compatible with the Ts(D) in the database, if their compatibility is false that means Ts(D) was already overwritten by newer transaction with larger timestamp value, then the transaction is aborted and its private workspace is also cleared. However, if otherwise, it goes into the next phase.
* **In Write Phase:** Data objects are copied from the local workspace into the database, once a transaction is in the write phase, it is considered to be complete. In the write phase, transaction makes all its updates permanent in the database.

1: Read(T1, Di){

2: T1(Ti) = Ti(Ti) ∩ [WTS(Di),∞ [;

3: If T1(Ti) = [] then

4: Restart(Ti);

5: Read(Di);

6: }

7: Pre-write(Ti, Di){

7: T1(Ti) = T1(Ti) ∩ [WTS(Di), ∞ [ ∩ [RTS(Di), ∞ [;

8: If T1(TI) = [] then

9: Restart(Ti);

10: }

Read and Pre-write algorithm for the OPTH

Ε represents a very small number, near zero

1: opth\_validation(Tv){

2: TS(Tv) = min(T1(Tv));

3: for ∀ Di ∈ (RS(Tv) ∪ WS(Tv))

4: for ∀ Ta ∈ active\_conflicting\_transaction(){

5: if Di ∈ (WS(Ta) ∩ (RS(Tv)) then

6: T1(Ta) = T1(Ta) ∩ [TS(Tv), ∞[ ;

7: if Di ∈ (RS(Ta) ∩ (WS(Tv)) then)

8: T1(Ta) = T1(Ta) ∩ [0, TS(Tv) – 1] ;

9: if Di ∈ (WS(Ta) ∩ (WS(Tv)) then

10: T1(Ta) = T1(Ta) ∩ [TS(Tv), ∞[ ;

11: if T1(Ta) = [] then

12: Restart(Ta);

13: }

14: if Di ∈ RS(Tv) then RT(Di) = max(RT(Di), TS(Tv));

15: if Di ∈ WS(Tv) then WT(Di) = max(WT(Di), TS(Tv));

16: }

17: commit WS(Tv) to database

18: }

Validation algorithm for the OPTA

􀀀\_\_\_\_􀀀􀀀\_\_􀀀\_

􀀀

􀀀\_􀀀􀀀􀀀\_ \_ 􀀀\_􀀀􀀀􀀀\_ \_ \_\_􀀀\_􀀀\_􀀀\_\_\_\_ \_

􀀀\_ 􀀀\_􀀀􀀀􀀀\_ \_ \_\_ \_\_\_\_

􀀀\_\_\_\_􀀀\_\_􀀀􀀀\_\_

􀀀\_\_\_\_\_􀀀\_\_

\_

􀀀\_\_􀀀\_\_\_\_􀀀􀀀\_ \_􀀀\_

􀀀

􀀀\_􀀀􀀀􀀀\_ \_ 􀀀\_􀀀􀀀􀀀\_ \_ \_\_􀀀\_􀀀\_􀀀\_\_\_\_ \_ \_\_􀀀\_􀀀\_􀀀\_\_\_\_ \_

􀀀\_ 􀀀\_􀀀􀀀􀀀\_ \_ \_\_ \_\_\_\_

􀀀\_\_\_\_􀀀\_\_􀀀􀀀\_\_

\_

􀀀\_\_\_\_􀀀􀀀\_\_􀀀\_

􀀀

􀀀\_􀀀􀀀􀀀\_ \_ 􀀀\_􀀀􀀀􀀀\_ \_ \_\_􀀀\_􀀀\_􀀀\_\_\_\_ \_

􀀀\_ 􀀀\_􀀀􀀀􀀀\_ \_ \_\_ \_\_\_\_

􀀀\_\_\_\_􀀀\_\_􀀀􀀀\_\_

􀀀\_\_\_\_\_􀀀\_\_

\_

􀀀\_\_􀀀\_\_\_\_􀀀􀀀\_ \_􀀀\_

􀀀

􀀀\_􀀀􀀀􀀀\_ \_ 􀀀\_􀀀􀀀􀀀\_ \_ \_\_􀀀\_􀀀\_􀀀\_\_\_\_ \_ \_\_􀀀\_􀀀\_􀀀\_\_\_\_ \_

􀀀\_ 􀀀\_􀀀􀀀􀀀\_ \_ \_\_ \_\_\_\_

􀀀\_\_\_\_􀀀\_\_􀀀􀀀\_\_

\_

Ts(X) is the last timestamp of an operation performed successfully on data item

Ts(T) is the timestamp of an active transactions.

**Proposed Algorithm 1**

If Ts((T1) < Ts(X)){

read operation is rejected and T1 is rolled back.

}

Else If (Ts(T1) ≥ Ts(X)){

T1 operation is executed

if (Ts (X) < Ts (T1)){

Valid = TRUE

}

2 else{

3 Valid =TRUE

}

4 if ((WriteSet (T1)∩WriteSet (X) ≠ Ø) and (Ts(X) > Ts(T1))){

5 Valid = FALSE

}

7 else if (ReadSet(T1)∩WriteSet (X) = β) {

8 for every element ε of β {

9 if (ε(T1) <Ts(X)) {

10 Valid = FALSE

11 Break

12 }

13 }

14 }

15 }

and set R-Ts(X) = max of R-Ts(X) and Ts(T1)

End if

**Rule two**: Transaction T1 issues write(X) operation.

If Ts(T1) < R-Ts(X) then

write operation is rejected and T1 is rolled back.

Else If Ts(T1) < W-Ts(X) then

write operation is rejected and T1 is rolled back.

Else If Ts(T1) ≥ W-Ts(X) and Ts(T1) ≥ R-Ts(X) then

write operation is executed

and sets W-Ts(X) = Ts (T1)

End if

**Proposed Algorithm 2**

If Ts((T1) < Ts(X)){

read operation is rejected and T1 is rolled back.

}

Else If (Ts(T1) ≥ Ts(X)){

T1 operation is executed

1: valid = TRUE;

2: or each Ti (i=1,2,…,n)

3: If RS(Tv) ∩ WS(Ti) ≠ {} then

4: Valid = FALSE;

5: Endif;

6: Endfor;

7: If valid then commit;

8: Else abort;

9: Endif;

RS – read set and WS – write set.

and set R-Ts(X) = max of R-Ts(X) and Ts(T1)

End if

**Rule two**: Transaction T1 issues write(X) operation.

If Ts(T1) < R-Ts(X) then

write operation is rejected and T1 is rolled back.

Else If Ts(T1) < W-Ts(X) then

write operation is rejected and T1 is rolled back.

Else If Ts(T1) ≥ W-Ts(X) and Ts(T1) ≥ R-Ts(X) then

write operation is executed

and sets W-Ts(X) = Ts (T1)

End if

1 if (Ti (ET) < Tj (ST)) Valid = TRUE

2 else{

3 Valid =TRUE

4 if ((WriteSet (Tj)∩WriteSet (Ti) ≠ Ø) and (Ti(ET) > Tj(V)))

5 Valid = FALSE

7 else if (ReadSet(Tj)∩WriteSet (Ti) = β) {

8 for every element ε of β {

9 if (ε(Tj) <Ti(ET)) {

10 Valid = FALSE

11 Break

12 }

13 }

14 }

15 }

1: valid = TRUE;

2: for each Ti (i=1,2,…,n)

3: If RS(Tv) ∩ WS(Ti) ≠ {} then

4: Valid = FALSE;

5: Endif;

6: Endfor;

7: If valid then commit;

8: Else abort;

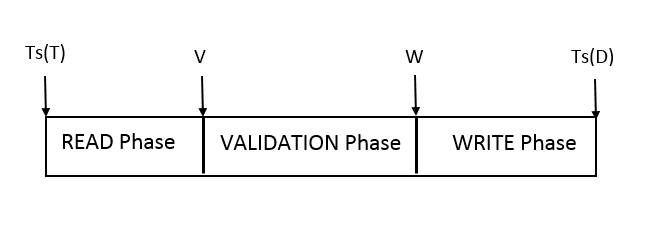
9: Endif;

RS – read set and WS – write set.

This work proposed a hybrid of optimistic concurrency control and Thomas’ Write Rule Timestamp Ordering Algorithms which consists of several phases: Read phase, Validation phase and Write phase.

In Fig --- shows each transactions at various stages and some symbols are assigned to each transaction.

* Ts(T): Timestamp of transaction
* V : Validation stage of transaction
* W : Write stage of transaction
* Ts(D) :which is the last timestamp of operation performed successfully on data item



Thomas’ Write Rule Timestamp Ordering is use at the read phase while optimistic concurrency control at the validation phase.

**In the read phase:** At Ts(T) stage, timestamp is given to transaction and will be compare to Ts(D) to ensure that the timestamp order of transaction execution is not violated. If this order is violated, then transaction is aborted and is rollback and start as a new transaction with a new timestamp else transaction read the value from data item to a private workspace area, does the computations and makes local copies of its modifications to the private workplace only and not to the database.

**In the validation phase**: Timestamp is given at V stage, where transaction has to be validated to check if the Ts(D) in the private workplace area is compatible with the Ts(D) in the database, if their compatibility is false that means Ts(D) was already overwritten by newer transaction with larger timestamp value, then the transaction is aborted and its private workspace is also cleared. However, if otherwise, it goes into the next phase.

**In Write Phase:** Data objects are copied from the local workspace into the database, once a transaction is in the write phase, it is considered to be complete. In the write phase, transaction makes all its updates permanent in the database.

**Case1**: If a transaction T1 issues read(X).

1. If Ts(T1) < W-Ts(X), then T1 needs to read a value of X that was already overwritten by newer transaction with larger timestamp value, hence, the read operation is rejected, and T1 is rolled back and start as a new transaction.
2. If Ts(T1) ≥ W-Ts(X), then the read operation is executed, and R-Ts(X) is set to the maximum of R-Ts(X) and Ts(T1).

**Case2**: If a transaction T1 issues write (X).

1. If Ts(T1) < R-Ts(X), then the value of X that T1 is producing was needed previously, and the system assumed that the value would never be produced. Then the write operation is rejected and T1 is rolled back.
2. If Ts(T1) < W-Ts(X), then T1 is attempting to write an obsolete value of X. Hence, the write operation is rejected and T1 is rolled back.
3. If Ts(T1) ≥ W-Ts(X) and Ts(T1) ≥ R-Ts(X), then the write operation can be performed and sets W-Ts(X) to Ts (T1).