An Optimistic Concurrency Control Approach Applied to Temporal Data in Real-time Database Systems

Walid MOUDANI, Nicolas KHOURY, Mohamad HUSSEIN
Business Computer Department
Lebanese University
LEBANON
wmoudani@ul.edu.lb

Abstract: - Real-time database systems (RTDBS) have received growing attention in recent years. RTDBS is a database system where transactions have explicit timing constraints such as deadlines. The performance and the correctness of RTDBS are highly dependent on how well these deadlines can be met. Scheduling of transactions is driven by priority considerations rather than fairness considerations. Concurrency control is one of the main issues in the studies of RTDBS. Optimistic concurrency control methods have the properties of being non-blocking and deadlock-free which are attractive for RTDBS. Furthermore, in the actual applications, real-time database systems require not only ensuring transactions finished in the specified time limits (deadlines), but also guaranteeing temporal consistency of data objects accessed by transactions. In this paper we propose an optimistic concurrency control method based on Similarity, Importance of transaction and Dynamic Adjustment or Serialization Order called OCC-SIDASO. This method uses dynamic adjustment of serialization order, operation similarity and the transaction importance, for maintaining transaction timeliness level, minimizing transactions wasted restart, and guaranteeing temporal consistency of data and transactions.

Key-Words: - Real-time Database Systems, Optimistic Concurrency Control (OCC), Temporal Consistency, Serialization Order

1 Introduction

In conventional database systems, concurrency control ensures the correct executions for concurrent transactions $T_1, ..., T_n$. Two solutions are possible to ensuring the correctness of concurrent transactions: (i) serial executions and (ii) serializable executions. In serial execution, the transactions $T_1, ..., T_n$ are not concurrent because each transaction is executed to completion before the next one. A serializable execution of the transactions $T_1,...,T_n$ are concurrent and computationally equivalent to a serial execution and produces the same output and has the same effect on the database as a serial execution. The main objective of concurrency control is to process all transactions in serializable way.

However, in Real-Time Database System (RTDBS), the transactions must be processed within definite time bounds, usually defined as a deadline. Failure to complete transactions before their deadlines greatly decreases the usefulness of the transactions. Deadlines may be lost due to problems in scheduling or transaction data contention. In the literature, a considerable research works has been devoted to designing concurrency control methods for RTDBS and to evaluating their performance. Most of these algorithms use serializability as correctness criteria and are based on one of the two basic concurrency control mechanisms: Pessimistic Concurrency Control [3, 12] or Optimistic Concurrency Control [2, 4, 5, 6, 11]. However, 2PL has some inherent problems such as the possibility of deadlocks as well as long and unpredictable blocking times. These problems appear to be serious in real-time transaction processing since real-time transactions need to meet their timing constraints, in addition to consistency requirements. Optimistic concurrency control methods have the properties of non-blocking and deadlock-free which make them especially attractive for RTDBS.

Another important aspect of real-time databases to be considered is temporal data consistency. RTDBS often process both temporal data objects whose state (value) may become invalid with the passage of time, and persistent data objects that remain valid regardless of time. A temporal validity interval is associated with the state (value) of a temporal data object. The values of temporal data objects lose validity after their. A temporal data object models a real world entity, for example, the position of an aircraft, and is updated by a periodic sensor transaction. The values of temporal data objects must reflect the change of the real world entities correctly and timely. Otherwise, decisions based on such data objects will be wrong, even disastrous. In

RTDBS, application transactions obtain the current states of the real world entities by real-time access to temporal data objects and further trigger the corresponding control actions for decision [8]. The traditional real-time concurrency methods [11] ensure logical consistency of data as well as meeting transaction deadlines, while neglect that the temporal consistency of temporal data objects must be guaranteed in real-time applications [8, 10].
In this paper, we propose an optimistic concurrency control method called OCC-SIDASO based on dynamic adjustment of serialization order using timestamp interval. This method uses importance of transaction and operation similarity and can ensure a very well real-time performance by minimizing transactions wasted restart, under circumstances of guaranteeing logical and temporal consistency of data. The remainder of this paper is organized as follows: in section 2 we review the most important existing real time concurrency control methods proposed in the literature and we provide a comparison between the described methods according to different criteria widely known in RTDBS. Afterward, the section 3 introduce the proposed method OCC-SIDASO, by presenting and explaining in detail the task of each step of this method. In section 4, we present performance evaluation of proposed method. Finally, section 5 includes the conclusion of our works and our future perspectives.

2 Related Works

Many researchers have been devoted to design appropriate concurrency control methods for RTDBS. Most concurrency control methods can be classified in one of the following mechanisms:

- The pessimistic concurrency control (PCC) method detects conflicts before making access to the data object.
- The optimistic concurrency control (OCC) method detects conflicts after transactions have accessed the data object.

The remainder of this section is organized as following: firstly, we describe the PCC. Secondly, we present the OCC. Thirdly, we describe several OCC methods. Finally, this section is ended by a comparison of OCC methods.

2.1 Pessimistic Concurrency Control

Pessimistic concurrency control methods are based on data access locking techniques which will possibly cause deadlocks and starvation problem when two transactions are querying two conflicting locks on the same data item [1]. The High Priority Two Phase Locking (HP-2PL) [15] resolves data conflicts in favor of transactions with higher priority by aborting the lower priority transaction and consequently may avoid deadlocks and thereby, eliminates the overhead of deadlock detection and deadlock solution. The favored transaction, the winner of the data conflict, is allowed then to lock the requested data object. HP-2PL is a primitive and non efficient concurrency control method which does not respect the temporal consistency of data and transactions. Furthermore, the use of locking technique will cause deadlocks in the case of mutual blocking of two or more transactions; this is why it is not suitable for real time transactions.
In order to take in account the temporal consistency, an approach based on Temporal Consistency High Priority-Two Phase Locking (TCHP-2PL) was proposed in [8], which is a real-time concurrency control method that can guarantee temporal consistency of data and transactions.TCHP-2PL uses priority of transactions attribute to choose between conflicting transactions and uses information about temporal consistency of data and transactions which are defined as follow:
(i) Temporal consistency of data is satisfied if the following two factors are valid:

- External consistency: a temporal data object is said to meet external consistency at a time t if its value is still valid according to its predefined temporal validity interval.
- Mutual consistency: It is the temporal consistency of a mutual relevance set which is a group of defined temporal data objects, which are used together to make decisions or derive new data.

(ii) A transaction is temporally consistent if every variable, independently, in its data set satisfies temporal consistency, and its mutual relevance sets satisfies mutual consistency.

Comparing with HP-2PL [15], the TCHP-2PL integrates the checking of temporal consistency so it can guarantee temporal consistency of transactions which is not possible with HP-2PL. However both methods use locking techniques and may suffer from starvation problem that can result from the repeated restart or blocking of a transaction in favor of a conflicting one, as well as from lock table overhead in system memory.
The THCP-2PL is enhanced by introducing similarity in order to increase concurrency level. The Similarity Based Temporal Consistency High

Priority Two Phase Locking (STCHP-2PL) first judges whether the conflicting operations meet operation similarity (i.e. if two different transactions operations operate on the same data variable X and returns similar values of it), then it allows them to execute concurrently. Therefore it uses an extended type of locks called share lock which does not conflict with any lock (Read or Write type locks). Therefore, in addition to the priority of transactions and temporal consistency checking, The STCHP-2PL uses operation similarity factor to guarantee temporal consistency of transactions and increase concurrency.

2.2 Optimistic Concurrency Control

The basic idea of an optimistic concurrency control mechanism is that the execution of a transaction consists of three phases: read, validation and write phases [11]. In the OCC, conflict detection and resolution are both done at the validation phase when a transaction completes its execution

2.2.1 Validation phase in OCC

In the validation phase of transaction T_i, the method checks that T_i does not interfere with any committed transactions or with any other transactions currently in their validation phase. In the OCC methods, the validation phase can be performed in one of two ways:

- **Backward validation**: in methods that perform backward validation, the validating transaction either commits or aborts depending on whether it has conflicts with transactions that have already committed. So this scheme does not allow us to take transaction characteristics into account and it is not suitable for real time database.
- **Forward validation**: in methods that perform forward validation, the validating transaction or the conflicting ongoing transactions can be aborted to resolve conflicts. This scheme can be extended to real time database since the timing characteristics of transaction can be considered and proper decision can be taken in aborting, delaying the committing transaction or aborting the conflicting ongoing transactions [6].

2.2.2 Dynamic adjustment of serialization order

The major performance problem with OCC methods is the late transaction restart. Thus, one important way to improve the performance of OCC methods is to reduce the number of transaction restarts. One way to reduce the number of transaction restarts is to dynamically adjust the serialization order of the conflicting transactions [4]. When some data conflict with the validating transaction is detected, there is no need to restart the conflicting transaction immediately. Instead, a serialization order can be dynamically defined as follows: a forward validation is applied when we have a read-write conflict or write-write conflict between T_v and T_j respectively, and a backward validation is applied when we have write-read conflict between T_v and T_j respectively.

To preserve serializability with OCC methods, if validating transaction T_v has to be serialized before active transaction T_j, the following two conditions must be satisfied [13]:

- **No overwriting**: The writes of T_v should not overwrite the writes of T_j
- **No read dependency**: The writes of T_v should not affect the read phase of T_j

There are three possible types of data conflicts which can induce serialization order between a validating transaction T_v and a conflicting transaction T_j :

- $RS(T_v) \cap WS(T_j) \neq \Phi$ (Read-Write conflict): Read-Write conflict between T_v and T_j can be resolved by adjusting the serialization order between T_v and T_j as $T_v \rightarrow T_j$. It means that the read of T_v cannot be affected by T_j's write. This type of serialization adjustment is called forward ordering.
- $WS(T_v) \cap RS(T_j) \neq \Phi$ (Write-Read conflict): Write-Read conflict between T_v and T_j can be resolved by adjusting the serialization order between T_v and T_j as $T_j \rightarrow T_v$. It means that the read phase of T_j is placed before the write of T_v. This type of serialization adjustment is called backward ordering.
- $WS(T_v) \cap WS(T_j) \neq \Phi$ (Write-Write conflict): Write-Write conflict between T_v and T_j can be resolved by adjusting the serialization order between T_v and T_j as $T_v \rightarrow T_j$ such that write of T_v cannot overwrite T_j's write.

To support dynamic adjustment of serialization order, a dynamic timestamp assignment method is used. For each transaction, T_i there is a timestamp called the latest commit timestamp $LCT(i)$ to indicate its serialization order relative to other transactions. Initially, the value of $LCT(i)$ is set to be ∞. If T_i has been backward adjusted, $LCT(i)$ is also used to detect whether T_i has accessed any invalid data at its validation test. Upon successfully passing its validation test, the validating transaction T_i is assigned a final serialization timestamp.

2.3 OCC methods

The different OCC methods proposed in the literature differ in the way of conflict resolution during the validation phase. Below, we describe shortly the most important optimistic methods in literature. Table 1 shows the different terms and parameters applied in OCC methods.

2.3.1 OCC method using Dynamic adjustment of serialization order using timestamp interval (OCC-DATI)

In [4], a method called OCC-DATI is presented allowing to minimize the number of transaction restarts by adjusting the serialization order dynamically between conflicting and validating transaction. In OCC-DATI all the checking is performed at the validation phase of each transaction, where it will be either forward or backward adjusted based on the conflict type. A serious conflict occurs when a conflicting active transaction has to be both backward and forward adjusted. The validating transaction is allowed to commit if the validity of all its accessed data are still sound and there is no serious conflict.

At the beginning of the validation, the final timestamp of the validating transaction $TS(T_v)$ is determined from the timestamp interval allocated to the transaction T_v. The timestamp intervals of all other concurrently running and conflicting transactions must be adjusted to reflect the serialization order. We set $TS(T_v)$ to the validation time if it belongs to the time interval of T_v or to maximum value from the time interval otherwise. The adjustment of timestamp intervals of active transactions iterates through the ReadSet (RS) and WriteSet (WS) of validating transaction. When access has been made to the same objects both in validating transaction and in the active transaction, the time interval of the active transaction is adjusted. Non-serializable execution is detected when the timestamp interval of an active transaction shuts out and transaction is restarted.

The OCC-DATI is enhanced by using the the importance of transactions found from transaction object attributes [7]. A real-time transaction object includes the attributes priority, deadline, and importance. The conflict resolution section uses dynamic adjustment of serialization order similarly to OCC-DATI but with the following conditions:

- Transactions of high importance should not be restarted because of data conflict with transactions of low importance when forward adjustment is applied.
- Transactions of high importance should not be backward adjusted, but conflicting lower importance should be restarted when backward adjustment is applied.

Using importance or criticalness of the transaction in place of the priority in the conflict resolution of OCC method avoids the dilemma of priority based conflict resolution, because transactions with very short deadline (i.e. very high priority) are not necessarily more critical than transactions with high importance.

A method which is similar to OCC-DATI is proposed in [16]. The conflict resolution in this method uses real-time serializability and the importance of transaction attribute. A transaction of higher importance should precede a transaction of lower importance in real-time history. Therefore, the transaction of higher importance should not be forward adjusted after a transaction of lower importance. Thus, if this is the case a transaction of lower importance is restarted. As well, in backward adjustment, we must ensure real-time serializable execution. Therefore, transactions of high importance should not be backward adjusted; instead, conflicting transactions having lower importance should be restarted.

2.3.2 Real-time OCC Method with Dynamic Adjustment of Serialization Order (OCC-DASO)

A method called OCC-DASO was proposed in [13] using the Thomas' write rule for updating the database with the writes of the validating transactions during the write phase. In this method, the number of transaction restarts is reduced by using dynamic adjustment of serialization order,

Parameters	Description
$RS(T)$	Data read set of transaction T
$RS_t^{to}(T)$	Temporal data read set of transaction T at time t
$WS(T)$	Data write set of transaction T
t	Current time
$s(T)$	Start time of transaction T
$d(T)$	Deadline of transaction T
$LCT(i)$	Latest commit timestamp of transaction T_i
$tsd_t(T)$	Temporal deferrable time : the time which the transaction T can be delayed at time t without affecting its temporal consistency
$VF_t(T)$	Validation factor of transaction T at time t
$TS(T)$	Timestamp of transaction T
TI	Validity timestamp interval of a transaction
$RTS(D_i)$	Largest timestamp of committed transactions that have read data object D_i
$WTS(D_i)$	Largest timestamp of committed transactions that have written data object D_i
T_v	Validating transaction
T_j, T_i	Concurrent transactions
$Avib(D_i)$, $Avie(D_i)$	Absolute validity interval (b: beginning – e: end)of temporal data item D_i
$dd_t(T)$	The data-deadline of transaction T at time t , $dd_t(T) = \min Avie(X_i)$ such that $X_i \in RS_t^{to}(T)$
K	Length of transaction absolute validate interval
$L(T)$	The number of object accesses of T
$L_t^{to}(T)$	The number of remaining temporal object accesses of T at time t
$E_t(T)$	The estimated remaining execution time of T at time t
$C_t(T)$	The estimated completion time of T at time, $C_t(T) = t + E_t(T)$
tib, tie	Timestamp interval beginning and timestamp interval end of a transaction
$S(T)$	Start time of transaction T
$IMP(T)$	Importance of transaction T (equivalent to priority attribute $P(T)$)
$a(T)$	Arrival time of T

Table 1. List of parameters used in OCC

which is supported with the use of a dynamic timestamp assignment scheme, and the Thomas' write rule.

The validation phase of OCC-DASO is divided into four steps: The first step of the validation phase is to test whether T_v has accessed any invalidated data. Second step detects the read-write conflicts between the set of the active transactions, T_j and the validating transaction T_v. The third step is to detect whether a backward-adjusted transaction T_j, also needs forward adjustment with respect to the validating transaction T_v. In the final step of the validation test, If T_v has not been selected for restart; we have to assign the final values to the conflicting and the validating transactions' tables, and update the read and write timestamps of data. By applying Thomas' write rule in its write phase, T_v will only update the database with its writes on the appropriate data item (with the valid timestamp).

The OCC-DASO is enhanced in [11] by using a parameter called transaction finish degree (TFD) which can avoid the near-to-complete transactions restarts. TFD values are calculated for the set of transactions that have conflict with the validating transaction T_v, and whose deadlines are smaller than T_v's deadline. TFD can depict if an adjustment of serialization order is necessary or can be avoided and both the conflicting transactions can meet their deadlines. The validation phase is divided into preparation phase and adjustment phase. In preparation phase, TFD values are computed and serious conflict is checked. The reordering of transaction commitment is performed in adjustment phase. According to the TFD_V values, the commitment order is decided; either forward ordering $T_v \rightarrow T_j$ or backward ordering $T_j \rightarrow T_v$, in condition that a serious conflict does not exist. If a serious conflict occurs between T_v and T_j then the variable versions read by the transactions are modified.

2.3.3 OCC method for accessing temporal data based on Validation factor and transaction deferrable time (OCC-VFTDT)

In [10], the proposed method is designed in which a checking algorithm is carried out to guarantee the use of validate data that fit with the transaction scheduling process. The checking process ensures that all temporal data in the read set of a transaction remain valid during all its execution time which will guarantee the temporal consistency of this transaction.

Afterward, the key factor concurrency control algorithm is adjusting validation rules during validation phase, which schedules the priority transactions that are near to complete by asserting validation factor. The validation algorithm calculates the validation factor of the validating transaction, which is a variable calculated from the current time, the start time and the deadline time of the transaction, and calculates the temporal deferrable time of the transaction $tsd_t(T)$.

2.3.4 OCC with virtual run policy

To support real-time transaction processing, a method is proposed by integrating the new criteria and issues of CPU and I/O scheduling, and the time cognizant conflict resolution scheme into the OCC method [6]. This method considers the timing characteristics of transaction and proper decision can be taken in aborting, delaying the committing transaction or aborting the conflicting ongoing transactions. Three schemes are presented in [6] such as:

- **OCC-forward validation with virtual run policy:** In this scheme (OCC-FV) the transaction that reaches its validation phase is allowed to commit if it is not a virtual first run transaction and all the active conflicting transactions which are in their read phases are immediately aborted and restarted if they are rerun transactions. In case some of the conflicting read phase transactions are in their first run, instead of aborting them they enter their virtual run and continue their read phase so as to bring data objects required to buffer, assuming the system buffer has a high retention effect, then a transaction in its second run and onward does not need to access the disk since the data objects are already in memory. When the virtual run transaction completes its read phase, it is aborted and resubmitted to the system to start its real second run. It is clear that there is no point to allow restarted rerun transaction to complete its read phase in virtual mode since all its data items are already in memory. This scheme does not take the transactions timing constraints into account and favors the validating one to save the amount of progress done by the validating transaction since it is near completion and will definitely complete if it is not restarted.
- **OCC-sacrifice with virtual run policy:** In this scheme (OCC-OS) when a transaction reaches its validation phase, it is aborted if one or more conflicting transactions have higher priority than the validating one; otherwise it commits and all the conflicting read phase transactions are restarted immediately. This method uses transaction priority (timing constraints) in such a way that the validating transaction sacrifices itself for the sake of conflicting ones with higher priority.
- **OCC-abort 50 with virtual run policy:** In this scheme (OCC-A50) when a transaction reaches its validation phase, its priority is checked against those conflicting transactions in the read phase. If more than 50% of the transactions in their read phase have higher priority than the transaction in its validation phase, the validating transaction is aborted and all other transactions are allowed to continue. If the number of transactions in the read phase having higher priority than validating transaction is less than or equal to 50%, the validating transaction is allowed to commit and all the other transactions are restarted.

2.4 Comparison of OCC and PCC methods

In this section we compare the described methods according to different parameters shown in Table 2. In OCC methods, either the validating transaction or the conflicting ongoing transaction can be aborted to resolve conflict. Moreover, dynamic adjustment of serialization order is used to reduce the number of transaction restarts caused by conventional OCC and to take the proper decision in aborting or delaying the committing transaction or aborting the conflicting ongoing transaction. Most of the literature methods based on OCC-DATI don't support temporal consistency of data and transactions except for one (OCC-VFTDT), and many of them favor transactions with higher importance. Our proposed algorithm using the OCC-DATI technique, supports temporal consistency of data and transactions, and takes in consideration the importance of transactions and the criticalness factor of data. The algorithm also attempts to outperform the previous methods by reducing the number of transaction restarts and increasing the concurrency level while maintaining the data valid as much as possible.

3 OCC method using Similarity and Importance of transaction and Dynamic Adjustment of Serialization

In this section, we describe an Optimistic Concurrency Control using Similarity and Importance of transaction and Dynamic Adjustment of Serialization Order called OCC-SIDASO.

This method respects the temporal consistency of data items as well as real time transactions. Furthermore, we introduce the concept of similarity and data criticalness factor to obtain a better real-time performance while guaranteeing temporal and logical consistency. Moreover OCC-SIDASO takes into consideration the importance of transaction during conflict resolution and applies a dynamic adjustment of serialization order only if the temporal consistency of data and transactions are not being violated. In addition, we relax serializability criterion by introducing data similarity and operation similarity, by allowing two conflicting operation to commit if they meet operation similarity which means when they are slightly different we consider them as acceptable.

The OCC-SIDASO method resolves conflicts using time intervals of the transactions. Every transaction must be executed within a specific time slot.

When an access conflict occurs, it is resolved using the read and write sets of the conflicting transactions together with the allocated time slot. Time slots are adjusted when a transaction commits. In this protocol, every transaction in the read phase is assigned a timestamp interval. This interval is used to record a temporary serialization order induced during the execution of the transaction. At the start of the execution, the timestamp interval of a transaction T is initialized as $[S(T), d(T)]$. Whenever the serialization order of the transaction is induced by its data operation or the validation of other transactions, its timestamp interval is adjusted to represent the dependencies.

Criteria/methods	*Supports real-time transactions*	*Supports data temporal consistency*	*Support transactions temporal consistency*	*Uses locking techniques (pessimistic method)- risk of Deadlocks*	*Uses optimistic concurrency control*	*CPU and I/O scheduling*	*Causes system memory overhead*	*Uses dynamic adjustment*	*Favors transactions with high importance or priority*
OCC-DASO	√				√		√	√	
THCP-2PL (PCC)	√	√	√	√			√		√
STHCP-2PL (PCC)	√	√	√	√			√		√
OCC-VFTDT	√	√	√		√		√	√	
OCC-DATI	√				√		√	√	
OCC-PDATI	√				√		√	√	√
MVOCC-TFD	√				√		√	√	
OCC-FV with virtual run policy					√	√			
OCC-sacrifice with virtual run policy	√				√	√			√
OCC-abort50 with virtual run policy	√				√	√			√

Table 2. Parameters used to compare OCC and PCC methods

OCC-SIDASO– VALIDATE (T_v) {

$IF(Validation\_time \in TI(T_v))$
$TS(T_v) = Validation\_time;$
Else
$TS(T_v) = \max(TI(T_v))$;

$For(\forall D_i \in RS(T_v) \cup WS(T_v))$
{
$IF(D_i \in RS(T_v))$
$TI(T_v) = TI(T_v) \cap [WTS(D_i), \infty[;$
$IF(D_i \in WS(T_v))$
$TI(T_v) = TI(T_v) \cap [WTS(D_i), \infty[\cap [RTS(D_i), \infty[;$
$IF(TI(T_v)) = [])$ $restart(T_v);$

$For(\forall T_a \in active\ conflicting\ transactions(T_v))$
{
$IF(D_i \in (WS(T_a) \cap RS(T_v)))$
$Forward - adjustment(T_a, T_v, D_i);$
$IF(D_i \in (RS(T_a) \cap WS(T_v)))$
$Backward - adjustment(T_a, T_v, D_i);$
$IF(D_i \in (WS(T_a) \cap WS(T_v)))$
$Forward - adjustment(T_a, T_v, D_i);$
$IF(TI(T_a)) = [])$ $restart(T_a);$
}
$IF(D_i \in RS(T_v))$
$RTS(D_i) = \max(RTS(D_i), TS(T_v))$;
$IF(D_i \in WS(T_v))$
$WTS(D_i) = \max(WTS(D_i), TS(T_v))$;
}
Commit $WS(T_v)$ to database;
}

Fig 1. Validation algorithm of OCC-SIDASO

$Forward - adjustment(T_a, T_v, D_i)$ {

$IF(Imp(T_v) \geq Imp(T_a))${
$IF((C(D_i) = False) AND (\ |V(Op_a, D_i) - V(Op_v, D_i)| \leq \alpha)\ )$
{
Commit $WS(T_v)$ to database;
Return;
}
Else IF $(C(D_i) = True\ AND\ CHECK\_POTENTIAL\_TC(T_a) = True\ )$ OR
$(C(D_i) = False\ AND\ |V(Op_a, D_i) - V(Op_v, D_i)| > \alpha$ AND
$CHECK\_POTENTIAL\_TC(T_a) = True)$

$TI(T_a) = TI(T_a) \cap [TS(T_v) + 1, \infty[;$
Else
$Abort\ (T_a);$
}
Else

$Restart\ (T_v);$
}

Fig 2. Forward Adjustment of OCC-SIDASO

In the remain of this section we present the : (i) validation phase algorithm of OCC-SIDASO, (ii) Operation similarity and data criticalness and (iii) temporal consistency checking algorithm.

3.1 Validation phase algorithm of OCC-SIDASO

This section presents the algorithm of validation phase OCC-SIDASO shown in figure 1. At the beginning of the validation, the final timestamp of the validating transaction $TS(T_v)$ is determined from the timestamp interval allocated to the transaction T_v. The timestamp intervals of all other concurrently running and conflicting transactions must be adjusted to reflect the serialization order. We set $TS(T_v)$ to the validation time if it belongs to the time interval of T_v or to maximum value from the time interval otherwise.

The adjustment of timestamp intervals of an active transaction iterates through the readset (RS) and writeset (WS) of validating transaction. When access has been made to the same objects both in validating transaction and in the active transaction and at least one of the operations is a write operation, then we have a conflict and the following procedures should be called accordingly:

- The forward-adjustment procedure is called if $D_i \in (WS(T_a) \cap RS(T_v))$ *or* $D_i \in (WS(T_a) \cap WS(T_v))$.
- The backward-adjustment procedure is called if $D_i \in (RS(T_a) \cap WS(T_v))$.

Non-serializable execution is detected when the timestamp interval of an active transaction shuts out, which means that it has to be both forward and backward adjusted, and then the transaction has to be restarted.

3.1.1 Forward adjustment of serialization order

During the validation phase of OCC-SIDASO method, the conflict type between validating transaction T_v and active transaction T_a is detected. When conflict data D_i is such that $D_i \in (WS(T_a) \cap RS(T_v)) ORD_i \in (WS(T_a) \cap WS(T_v))$, the forward-adjustment algorithm shown in figure 2 is called. If the active conflicting transaction is more important from the validating one then the validating transaction is restarted. If it is not the case, the two conflicting operations Op_a and Op_v are allowed to commit concurrently if there are operation similarity and the conflict data item D_i is not critical. Otherwise, a forward adjustment of serialization order is applied to the active conflicting transaction T_a by adjusting the timestamp interval of T_a. This adjustment of timestamp interval of the active transaction iterates through the readset (RS) and writeset (WS) of validating transaction T_v. Note that, forward adjustment will only be allowed if the forwarding of the conflicting transaction T_a does not violate the temporal consistency of data and transactions. This temporal consistency check is done by the *CHECK_POTENTIAL_TC(T_a)* procedure which will be explained later in this section.

3.1.2 Backward adjustment of serialization order

During the validation phase, if the conflict type between validating transaction T_v and active conflicting transaction T_a, is such that conflicting data $D_i \in (RS(T_a) \cap WS(T_v))$, the backward-adjustment ( ) procedure shown in figure 3 is called. If the active conflicting transaction is more important from the validating one then the validating transaction is restarted. If it is not the case, the two conflicting operations Op_a and Op_v are allowed to commit concurrently if there are operation similarity and the conflict data item D_i is not critical. Otherwise, a backward adjustment of serialization order is applied to the active conflicting transaction T_a by adjusting the timestamp interval of T_a.This adjustment of timestamp interval of the active transaction iterates through the readset (RS) and writeset (WS) of validating transaction T_v. Note that, backward adjustment will only be allowed if the backwarding of the conflicting transaction T_a does not violate the temporal consistency of data and transactions. This temporal consistency check is done by the *CHECK_POTENTIAL_TC(T_a)*.

3.2 Operation similarity and data criticalness

The OCC-SIDASO method introduces the concept of similarity (operation similarity) for non-critical temporal data items which is defined as follows:

Suppose t_m and t_n are a pair of concurrent transactions, $Op_i \in t_m$, $Op_j \in t_n$ and Op_i and Op_j operate on the same non-critical data object D (conflicting operations). If the following condition is satisfied:

$\left|V(Op_i,D)-V(Op_j,D)\right| \le \alpha$ (α is the threshold value whose value depends on the application semantics,), then Op_i and Op_j are said to be operation similarity, notated by $Op_i \approx Op_j$ A temporal data D is critical ($C(D) = TRUE$) if catastrophic results occur when $0 \prec \left|V(Op_i,D)-V(Op_j,D)\right| \le \alpha$ While, a temporal data D is non-critical ($C(D) = FALSE$) if no catastrophic results occur when $0 \prec \left|V(Op_i,D)-V(Op_j,D)\right| \le \alpha$.

3.3 Temporal consistency checking

In our method, the maintenance of data temporal consistency and transactions is done in 2 phases: phase A and phase B.

Phase A: Before each temporal data access by the running transaction, the algorithm *Check_TC()* shown in figure 4 is called to check the temporal validity of accessed data and to guarantee the correct data is being scheduled. *Check_TC ()* takes as input the temporal data readset $RS^{to}_{t}(T)$of a transaction T and for every member D_i the following condition is checked: $If\ |\ avie\ (D_i) - avib\ (D_i)\ | < k$, then temporal data X is fugitive, otherwise it is steady. The value k is the length of transaction absolute validate interval.

Next the algorithm checks whether the transaction can commit before its data-deadline $dd_t(T)$ and before its timestamp interval ending. If it is the case, temporal consistency is then satisfied thus we change the value of k as the length of temporal data absolute validate interval and the function will return a TRUE value. Otherwise, the transaction will be aborted. So every fugitive data will be checked for the temporary consistency by the algorithm, which guarantee the transaction can commit correctly.

CHECK_TC (T)

INPUT: $RS^{to}_{s(t)}(T) = \{D_i, D_2, \dots D_m\}$

$\{k = \infty; N = L^{to}_{s(t)}; i = 1;$

While ($RS^{to}_{t}(T) \neq \phi$) {

T accesses D_i from $RS(T)$;

$RS^{to}_{t}(T) = RS^{to}_{t}(T) - \{D_i\};$

$If\ |\ avie\ (D_i) - avib\ (D_i)\ | < k$ *Then*

if ($dd_t(T) < C_t(T)$ *OR* $tie(T) < C_t(T)$)

Return False;

Else

$k = k \bigcap_{j=1}^{i} |avie(D_j) - avib(D_j)|;$

$i = i + 1;$ }

Return True;

}

Fig 4. Temporal consistency checking algorithm

Backward − adjustment (T_a, T_v, D_i) {

$IF(Imp(T_v) \ge Imp(T_a))$ {

$IF\big((C(D_i) = False) AND\ (\ |V(Op_a,D_i) - V(Op_v,D_i)| \le \alpha)\ )$ {

Commit $WS(T_v)$ to database;

Return;

}

Else IF$(C(D_i) = True\ AND\ CHECK_POTENTIAL_TC(T_a) = True\ )$ OR

$(C(D_i) = False\ AND\ |V(Op_a,D_i) - V(Op_v,D_i)| > \alpha$ AND $CHECK\_POTENTIAL\_TC(T_a) = True)$

$TI(T_a) = TI(T_a) \cap [0, TS(T_v) - 1];$

Else

Abort (T_a);

}

Else

$Restart\ (T_v);$

}

Fig 3. Backward adjustment of OCC-SIDASO

Phase B: At the validation time and as we mentioned above, in forward-adjustment and backward-adjustment procedures, no transaction serialization order can be adjusted without maintaining the temporal consistency of data and transactions. This condition is insured by the function *CHECK_POTENTIAL_TC()* showed in figure 5 which is similar to the previous *CHECK_TC()* with a difference that presumptive timestamp interval and completion time of the conflicting transaction are calculated and used in the function to check the transaction temporal consistency in case it is dynamically adjusted .Those two calculated values will replace the real timestamp interval and completion time of $T_a$if it passes the temporal consistency checking.

3.4 Analysis

In this section, we have proposed the OCC-SIDASO method based on the dynamic adjustment of serialization order, the importance of transaction and the operation similarity factor. The method maintains the temporal consistency of data items and real time transactions simultaneously. OCC-SIADSO has a main advantage over the earlier optimistic concurrency control techniques, which is the presumptive temporal consistency checking that avoids unnecessary and inaccurate adjustment of transactions. This critical feature correctness will be verified later in the performance evaluation section.

4 Performance Evaluation

The scheduling decision of conflicting transactions is taken according to the conflict type , the importance of conflicting transactions, the similarity and temporal consistency factors. In order to evaluate the performance of OCC-SIDASO methods, we present the simulation results of OCC-SIDASO for two different numeric scenarios, each containing two transactions with data items and an execution schedule with their specific attributes. To demonstrate the efficiency of our proposed algorithm, we will simulate the same two scenarios with two existing methods from literature which are: STCHP-2PL and OCC-VFTDT. These methods are chosen for the reason that they both respect and maintain the temporal consistency of data and transactions. In addition, STCHP-2PL introduces the concept of similarity and OCC-VFTDT makes use also of the same temporal consistency checking function that we use in our proposed method. The simulation result will be compared and discussed to prove the correctness and the outperformance of our proposed method.

CHECK_Potential_TC **(*T*)**

INPUT: $RS^{to}_{s(t)}(T) = \{ D_i, D_2, \dots D_m\}$

$TI'(T_a) = TI(T_a) \cap [TS(T_v) + 1, \infty[$;
// If we have a forward-adjustment procedure

$TI'(T_a) = TI(T_a) \cap [0, TS(T_v) - 1]$;
// If we have a backward-adjustment procedure

$C'_t(T_a) = tib'(T_a) + E(T_a)$

$\{k = \infty; N = L^{to}_{s(t)}; i = 1;$

While $(RS^{to}_{t}(T) \neq \phi)$ {

T accesses D_i from $RS(T)$;

$RS^{to}_{t}(T) = RS^{to}_{t}(T) - \{D_i\}$;

$If \mid avie(D_i) - avib(D_i) \mid < k$ Then

$if\ (dd_t(T) < C'_t(T)\ OR\ tie'(T) < C'_t(T))$

Return False;

Else

$k = k \bigcap_{j=1}^{i} |avie(D_j) - avib(D_j)|;$

$i = i + 1;$ }

Return True;

}

Fig 5. Potential temporal consistency checking algorithm

4.1 Simulation prototype

In order to validate our proposed method, we develop a simulation proptotype of the proposed algorithm built on Microsoft Visual Studio 2005 developement environment and the C++ programming language. The main objective of the prototype is to simulate the OCC-SIDASO on a given schedule and then generating a serializable execution of the schedule respecting transactions deadlines and importance and maintaining data temporal consistency.

4.2 Simulation scenarios

In our simulation, we use two scenarios including two transactions: T_1 and T_2. Each transaction contains read and write operations, and three data items: X, Y and Z. The tables 3 and 4 presents the transactions attributes: importance or priority, start time, execution time, completion time, deadline (or timestamp interval end) and timestamp interval.

Scenario 1										
T_1: W1(X) R1(Y) $\alpha = 1$ $R_{mvi} = 1$ Imp(T_1) =P(T_1)= 3, C(T_1) = t6, S(T_1) = t3 E(T_1) = 3 d(T_1) = tie(T_1) = t12, TI(T_1) = [3,12]						T_2: W2(Y) W2(Z) R2(Z) Imp(T_2) =P(T_2)= 5, C(T_2) = t10, S(T_2) = t1 E(T_2) = 9, d(T_2) = tie(T_2) = t14, TI(T_2) = [1,14]				
X=5, RTS(X)=1, WTS(X)=1 C(X)=TRUE, ST(X)=1, VI(X)=13			Y=3, RTS(Y)=1, WTS(Y)=1 C(Y)=TRUE, ST(Y)=0, VI(Y)=13				Z=6, RTS(Z)=1, WTS(Z)=1 C(Z)=FALSE, ST(Z)=4, VI(Z)=2			
Operands	+2	+1	+3							
Schedule : W2(Y) W2(Z) W1(X) R2(Z) V2 C2 R1(Y) V1 C1										
Time in OCC	t1	t2	t3	t4	t5					
Timeline STCHP-2PL	t1,t2	t3,t4	t5,t6	t6,t7			t8,t9			

Table 3. Transactions, data items and schedule information of scenario 1

Scenario 1										
T_1: R1(Y) W1(X) $\alpha = 1$ $R_{mvi} = 1$ Imp(T_1) =P(T_1)= 3, C(T_1) = t8, S(T_1) = t2 E(T_1) = 6, d(T_1) = tie(T_1) = t10, TI(T_1) = [2,10]						T_2: W2(X) R2(Z) Imp(T_2) =P(T_2)= 1, C(T_2) = t6, S(T_2) = t1 E(T_2) = 5, d(T_2) = tie(T_2)=t11, TI(T_2) = [1,11]				
X=2,RTS(X)=1,WTS(X)=1 C(X)=TRUE, ST(X)=1, VI(X)=13			Y=4,RTS(Y)=1,WTS(Y)=1 C(Y)=TRUE,ST(Y)=0, VI(Y)=13				Z=6,RTS(Z)=1,WTS(Z)=1 C(Z)=FALSE,ST(Z)=2, VI(Z)=10			
Operands	+2	+0	+0	+5						
Schedule : W2(X) R1(Y) R2(Z) W1(X) V1										
Time in OCC	t1	t2	t3	t4	t5	t6				
Timeline STCHP-2PL	t1,t2	t3,t4	t5,t6	t6,t7						

Table 4. Transactions, data items and schedule information of scenario 2

Furthermore, tables 3 and 4 presents data items attributes are: read timestamp, write timestamp, criticalness, start time and validity interval. We notice that for the pessimistic method (STCHP-2PL), we consider that an operation requires two time units: one for acquiring the requested lock and one for execution. In the figure 6 and 7, we give for each of the two scenarios a timeline graph to illustrate its time relativity.

4.3 Simulation results applied on scenario 1

In this section, we present the simulation results based on scenario 1 for three methods: OCC-SIDASO, *OCC-VFTDT and STCHP-2PL*.

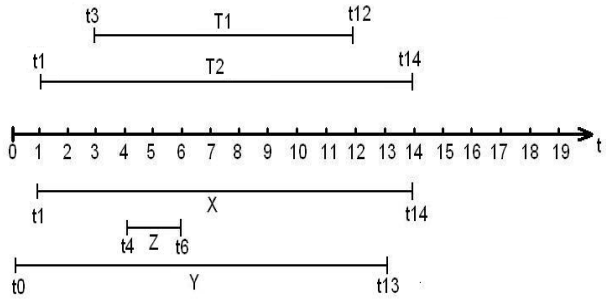

Fig. 6. Timeline of scenario 1 in optimistic methods

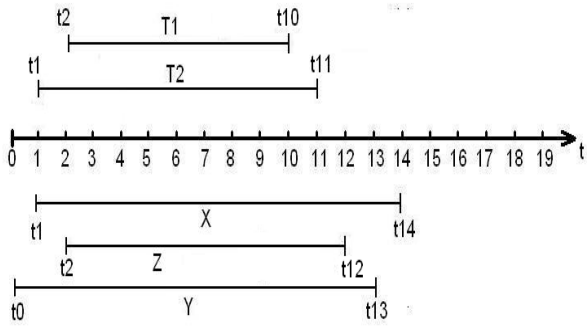

Fig. 7: Timeline of scenario 2 in optimistic methods

4.3.1 Simulation result with OCC-SIDASO

In the schedule of the first scenario and at time t5 when T_2 is validating, the validation procedure is called where it decides to backward adjust the conflicting transaction T_1. Afterward, according to the conflict type detected between the validating transaction T_2 and the conflicting transaction T_1, the OCC-SIDASO algorithm decides to call the backward-adjustment procedure that checks the properties and the attributes of the schedule and its elements (transactions and operations) and chooses the appropriate conflict resolution method which is

on our case aborting the conflicting transaction T_1.Moreover, during the Backward-adjustment (T_1,T_2,Y), a backward adjustment of the serialization order of the conflicting transaction T_1 is allowed only if it does not violate the temporal consistency of accessed data items and transactions. In this scenario, this criterion is not verified according to the $CHECK\_Potential\_TC\ (T_1)$.

According the simulation result applied on scenario 1, the OCC-SIDASO algorithm decides to abort the active conflicting transaction $T_1$for the favor of the validating transaction T_2. This decision was taken due to the following reasons:

- The importance of T_2 is greater than the importance of T_1 which means T_2should not be restarted.
- Conflict data item Y is critical which means no operation similarity is allowed.
- A presumptive $TI'(T_1)$and $C_t'(T_1)$ are calculated to check later the temporal consistency of T_1 in case it is backward adjusted.
- Further, the presumptive temporal consistency checking shows that the backward adjustment of T_1 will violate the transaction temporal consistency, therefore T_1 is aborted and T_2 commits to database.

4.3.2 Simulation result with OCC-VFTDT

In this section, we present the simulation result of the first scenario under the method OCC-VFTDT in order to compare its results with OCC-SIDASO. In the schedule of this scenario at time t5 when T_2 is validating, the OCC-VFTDT algorithm is called. First, the algorithm calculates the following parameters that will be used in the conflict resolution decision taking:

$$VF_{t5}(T_2) = t - s(T_2) \,/\, d(T_2) - s(T_2) = (5-1)\,/\,(14-1) = 0.3$$

$dd_{t5}(T_2) = 13 < d(T_2) = 14$ then
$tsd_{t5}(T_2) = dd_{t5}(T_2) - C_{t5}(T_2) = 13 - 10 = 3$

$dd_{t5}(T_1) = 14 \geq d(T_1) = 12$ then
$tsd_{t5}(T_1) = d(T_1) - C_{t5}(T_1) = 12 - 6 = 6$

Second, the OCC-VFTDT algorithm chooses the method of conflict resolution by deciding the type of serialization order to follow according to the conflict type detected between the validating transaction T_2 and the conflicting transactionT_1, when T_2 is validating at time t5, OCC-VFTDT detects a data conflict on item Y such that $RS(T_1) \cap WS(T_2) = Y$. The validation factor of transaction T_2 is calculated which is not greater or equal than 1 then the condition is not validated and the execution order will be T_1, T_2 which means that T_1 will be backward adjusted. Conversely, the same transaction was aborted by OCC-SIDASO algorithm due to its temporal consistency violation in case it is adjusted.

4.3.3 Simulation result with STCHP-2PL

In this section, we present the simulation result of the first scenario under the method STCHP-2PL to compare later its result with the one of OCC-SIDASO. Under STCHP-2PL, the schedule in scenario 1 will be running according to the following timeline (Figure 8).

In this scenario, T_1 requests for a read lock on data item Y which has been locked with an exclusive write lock by T_2. The STCHP-2PL does not detect any operation similarity in history between the conflicting operations therefore one of the conflicting transactions must be aborted or blocked. The requesting transaction T_1has a lower importance than T_2 which is holding the requested lock and the set of data items that T_1 has accessed or willing to access meets mutual consistency, then the algorithm decides to block T_1 and to keep on executing T_2.

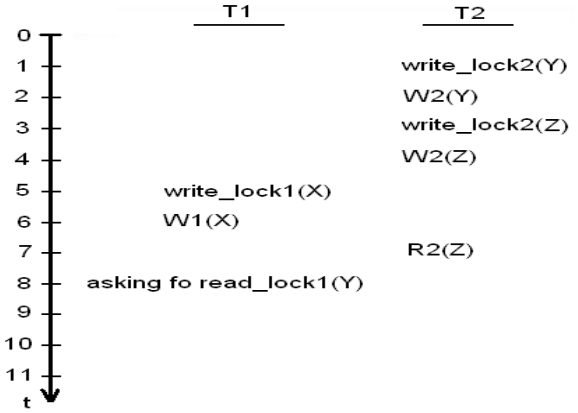

Fig 8. Timeline of scenario 1 applied in STCHP-2PL

4.4 Simulation results applied on scenario 2

In this section, we present the simulation results based on scenario 2 for three methods: OCC-SIDASO, *OCC-VFTDT and STCHP-2PL.*

4.4.1 Simulation result with OCC-SIDASO

In the schedule of the second scenario at time t5 when T_1 is validating, the validation procedure is called where it decides to forward adjust or

backward adjust or restart the conflicting transaction T_2. Afterward, according to the conflict type detected between the validating transaction T_1 and the conflicting transaction T_2, the OCC-SIDASO algorithm decides to call the Forward-adjustment procedure that checks the properties and the attributes of the schedule and its elements (transactions and operations) and chooses the appropriate conflict resolution method which is on our case forward adjusting the conflicting transaction T_2. Moreover, during the Forward-adjustment (T_2, T_1, X) procedure a forward adjustment of the serialization order of the conflicting transaction T_2 is allowed only if it does not violate the temporal consistency of accessed data items and transactions. In this scenario this criterion is verified according to the CK_Potential_TC (T_2) .

According the simulation applied on scenario 2, the OCC-SIDASO algorithm decides to forward adjust the serialization order of the conflicting transaction $T_2$and to keep the schedule of T_1 as it is. This decision was taken due to the following reasons:

- The importance of T_1 is greater than the importance of T_2 which means T_1 should not be restarted.
- Conflict data item X is critical which means no operation similarity is allowed.
- A presumptive $TI'(T_2)$ and $C_t'(T_2)$ are calculated to check later the temporal consistency of T_2 in case it is forward adjusted.
- Further, the temporal consistency checking shows that the forward adjustment of T_2 will maintain the transaction and data temporal consistency, therefore T_2 will be rescheduled to time t6 and T_1 commits to database

4.4.2 Simulation result with OCC-VFTDT

In this section, we present the simulation result of the second scenario under the method OCC-VFTDT to compare its results with OCC-SIDASO. In the schedule of the second scenario, and at time t5 when T_1 is validating, the OCC-VFTDT algorithm is called. The algorithm calculates the following parameters that will be used in the conflict resolution decision taking:

$$VF_{t5}(T_1) = t - s(T_1) / d(T_1) - s(T_1)$$
$$= (5 - 2) / (10 - 2) = 0.3$$

$$dd_{t5}(T_1) = 14 \geq d(T_1) = 10 \text{ then}$$
$$tsd_{t5}(T_1) = d(T_1) - C_{t5}(T_1) = 10 - 8 = 2$$

$$dd_{t5}(T_2) = 14 \geq d(T_2) = 11 \text{ then}$$
$$tsd_{t5}(T_2) = d(T_2) - C_{t5}(T_2) = 11 - 6 = 5$$

In this scenario, when T_1 validates at time t5 a data conflict is detected on item X such that $WS(T_2) \cap WS(T_1) = X$. The validation factor of transaction T_1 is calculated which is less than 1 and the transaction temporal deferrable time of T_1 is not greater than the one of T_2, then the condition is not validated and the execution order will be T_1, T_2 which means T_2 will be forward adjusted. Similarly, OCC-SIDASO decides to forward adjust T_2 after checking its presumptive temporal consistency.

4.4.3 Simulation result with STCHP-2PL

In this section, we present the simulation of the second scenario under the method STCHP-2PL to compare its results with the ones of OCC-SIDASO. Under STCHP-2PL, the schedule in scenario 2 will be running according to the following timeline (Figure 9).

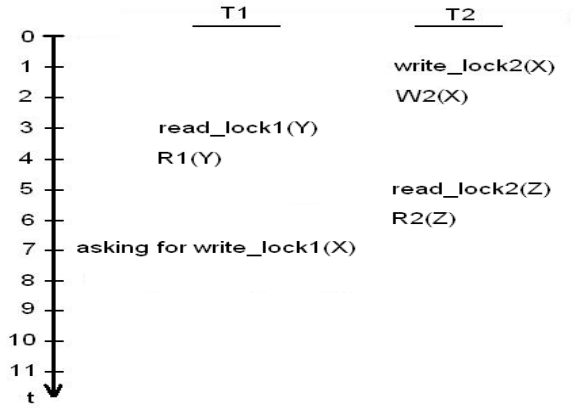

Fig 9. Timeline of scenario 2 applied in STCHP-2PL

In this scenario, T_1 requests for a write lock on data item X which has been locked with an exclusive write lock by T_2. The STCHP-2PL does not detect any operation similarity in history between the conflicting operations therefore one of the conflicting transactions must be aborted or blocked. The requesting transaction T_1 has a higher importance than T_2 which is holding the requested lock and the set of data items that T_1 has accessed or willing to access meets mutual consistency, then the algorithm decides to abort T_2 and grant T_1 the requested lock. However, in the first scenario, OCC-SIDASO decides to forward adjust the serialization order of the active conflicting transaction T_2 and to keep on validating T_1.

	OCC-SIDASO	STCHP-2PL	OCC-VFTDT
Scenario 1	Conflict transaction is aborted for violating temporal consistency if adjusted	**Unsafe restart**: the conflicting transaction is blocked, which will allow restarting it later and violating the temporal consistency of data and transaction.	**Incorrect Adjustment**: the conflicting transaction is backward adjusted which is a wrong decision because this transaction will be violating its temporal consistency according to the result of Check_Potential_TC() checking function in OCC-SIDASO.
Scenario 2	Conflict transaction is forward adjusted after validating its presumptive temporal consistency	**Wasted abort**: the transaction holding the lock is aborted and the lock is granted to the requesting (conflicting) transaction. This is a wasted abort which is prevented by our proposed method where the validating transaction keeps on executing.	**Inaccurate adjustment**: the conflicting transaction is forward adjusted and it has a deferrable time greater than the validating transaction. This condition is not accurate for adjusting a transaction because it is possible that this transaction would not have the sufficient time to complete before its deadline or data deadline.

Table 5. OCC-SIDASO, STCHP-2PL and OCC-VFTDT simulations results comparison

4.5 Discussion

The two scenarios simulation results prove that our proposed method can take more accurate decisions in the conflict resolution between real-time transactions than two existing outperforming methods in literature which are STCHP-2PL and OCC-VFTDT. As we can see in table 5, the reasons behind this dominance are the weak points of STCHP-2PL method which are the unsafe restart and the wasted abort, as well as the issues caused by OCC-VFTDT algorithm which are the incorrect adjustment and the inaccurate adjustment.

Furthermore, OCC-SIDASO makes use of the importance of transactions, the operation similarity factor, the simultaneous and the presumptive temporal consistency checking, to get to the most precise decision which can cover the maximum number of data conflict scenarios. Consequently, we conclude that our method is equivalent to the other existing methods in some cases and outperforms them in other cases.

4 Conclusion and perspectives

In this paper, we proposed an optimistic concurrency control called OCC-SIDASO with the capability of predicting the correctness of the transactions history in case it is rescheduled.

Furthermore, we used the concept of similarity between conflicting operations to obtain a better real-time performance, and the transaction importance criterion in order to favor transactions with higher importance in data conflict resolution. Also, a simulation implementation and a performance comparison between OCC-SIDASO and two real-time concurrency control methods from literature show that our method can ensure a very well real-time performance while guaranteeing temporal consistency and can even outperform these methods in some cases.

Future work could include the use of fuzzy logic techniques by creating a set of fuzzy rules that will form the fuzzy logic engine in order to deal with the importance, the criticalness and the similarity attributes. By using these rules, fuzzy logic will try to provide an easy conflict resolution method between transactions.

Moreover, we can try to implement our proposed method on a real-time database test platform like MMRTDBTP (Main-Memory Real-Time Database Platform), and on a real database management system to obtain more accurate results.

References:

[1] R. EL Masri, S. Navathe, *Fundamentals of database systems*, Addison-Wesley, 6th edition, 2011.

[2] L. Kwok-wa, L. Kam-yiu, H. Sheung-lun, Real-time Optimistic Concurrency Control protocol with Dynamic Adjustment of Serialization Order, *Conference Real-Time Technology and Applications Symposium*, 1995.

[3] P.A. Bernstein, N. Goodman, Concurrency Control in Distributed Database Systems. *ACM, Computing Surveys*, Vol. 13(2), 1981.

[4] J. Lindström, Dynamic Adjustment of Serialization Order using Timestamp Interval in Real-Time Databases, *IEEE Conference on Transaction Processing*, 1999.

[5] H. Qilong, H. Zhongxiao, Real-time Optimistic Concurrency Control based on Transaction Finish Degree, *Journal of Computer Science*, Vol. 1(4): 471-476, 2005.

[6] A. Abu-Ali, On Optimistic Concurrency Control for Real-Time Database Systems, *American Journal of Applied Sciences*, Vol. 3 (2), pp. 1706-1710, 2006.

[7] J. Lindström, K. Raatikainen, Using Importance of Transactions and Optimistic Concurrency Control in Firm Real-Time Databases. *Seventh International Conference on Real-Time Computing Systems and Applications (RTCSA'00)*, pp. 463-467, 2000.

[8] X. Ying-yuan, H. Zhang, W. Fa-yu, Maintaining Temporal Consistency in Real-Time Database Systems, *International Conference on Convergence Information Technology*, 2007.

[9] K. Ramamritham, H.S. Sang, L.C. DiPippo, Real-Time Databases and Data Services, *Journal Real-Time Systems*, Vol. 28, Issue 2-3, pp. 179-215, 2004.

[10] H. Qilong, P. Haiwei, Y. Guisheng, A Concurrency Control Algorithm Access to Temporal Data in Real-time Database Systems, *International Multi-symposiums on Computer and Computational Sciences*, 2008.

[11] Baothman F., Sarje A.K. and Joshi R.C.: On optimistic concurrency control for RTDBS. *American Journal of Applied Sciences*, Vol. 3 (2): 1706-1710, 2006.

[12] A. Chiu, K. Ben, L. Kam-yiu, Comparing Two-Phase Locking and Optimistic Concurrency Control Protocols in Multiprocessor Real-Time Databases, *Proceeding of the 1997 Joint Workshop on Parallel and Distributed Real-Time Systems*, 1997.

[13] J. Huang, J.A. Stankovic, K. Ramamrithan, D. Towsley, Experimental evaluation of real-time optimistic concurrency control schemes, *Proceedings of the 17th International Conference on Very Large Data Bases*, 1991.

[14] W. Peng, P. Zilong, Research on the improvement of the concurrency control protocol for real-time transactions, *International Conference on Machine Vision and Human-machine Interface*, 2010.

[15] C. Lau, V. Lee, Real Time Concurrency Control For Data Intensive Applications, *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2005.

[16] J. Lindström, K. Raatikainen, Using Real-Time Serializability and Optimistic Concurrency Control in Firm Real-Time Databases. *University of Helsinki Report*, 17th March 2000.

[17] W. Yongyan, W. Qiang, W. Hongan, D. Guozhong, Dynamic Adjustment of Execution Order in Real-Time Databases, *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, 2004.