8

Real-Time Concurrency Control Protocol Based on Accessing Temporal Data

Qilong Han
College of Computer Science and Technology,
Harbin Engineering University Harbin,
China

1. Introduction

Concurrency control is one of the main issues in the studies of real-time database systems. On the one hand, it is related closely to active real-time database and real-time application. Concurrency control algorithm seriously affect the performance of the system in real-time, may cause unpredictable consequences. On the other hand, updating data in active real-time database may trigger a new transaction and to further increase the difficulty of the concurrency control. How ensures both the consistency of the database and the finish of the transactions before deadline; it is an important problem to the concurrency control research in the active real-time database systems. In the most literature, the existing research more focuses on the transactions deadline and lack of attention the data temporal constraint. This is mainly due to the real-time data is obtained dynamically by sensors in real-time database, and the transactions of real-time databases read the sensor data only, do not write the sensor data. But the real-time data may miss deadline and become invalid before transaction which reads it committed, most concurrency control algorithms are regardless of the sensor data deadline and its invalid effect to systems. This chapter will further discuss the concurrency control method that transactions access real-time sensor data.

Optimism concurrency control method is widely used in real-time database due to no dead-lock and no-block characteristics, but delay conflicts detection brings with great restart overhead. A dynamic adjustment serialization order method is proposed to reduce unnecessary affairs restart number [Haritsa, Lindstrom], according to [Lindstrom, Wang] a method is proposed by control the reading and writing data to reduce the transaction restart number. In [Brad] discussed about the relation between real-time data and the derived data validity, and [Kuo] proposed the concept of real-time data similarity. According to [Brad, Xiong] proposed the concept of data-deadline, and the transaction scheduling strategy by using mandatory waiting method. The [Liu] also discuss on data deadline and transaction scheduling. A real-time transaction scheduling method is proposed based on the combination of [Kuo, Son] improving real-time data similarity mechanism in [Xiong]. But [Xiong, Son] only take account of single transaction scheduling the real data problems, not consider the concurrency control problem that the transactions did not arrive at deadline and its accessed data expired, which will increase the number of transactions restart.

The chapter is organized as follows. Section 2 reviews concurrency control protocols proposed in real-time database systems (RTDBSs) and describes our choice of concurrency control algorithms for accessing temporal data. Section 3 analyzes the validity of active real-time system model and the effects real-time data to the concurrent control. The relevant definition and the effective examination mechanism of transaction reading data were given in Section 4. In Section 5 based on Data temporal characteristics, concurrency control algorithm RTCC-DD (Real-time Concurrency Control Algorithm Based on Data-deadline) is put forward, and proved that the RTCC-DD was the correctness in theory. Section 6 carries on the analysis comparison between our proposed method and the existed method through experiment. Finally, Section 7 gives summary and conclusions of this chapter and future directions for the work.

2. Concurrency control in Real-Time Database Systems

A Real-Time Database Systems (RTDBS) processes transactions with timing constraints such as deadlines [Ramamritham]. Its primary performance criterion is timeliness, not average response time or throughput. The scheduling of transactions is driven by priority order. Given these challenges, considerable research has recently been devoted to designing concurrency control methods for RTDBSs and to evaluating their performance (e. g. [Kuo, Fishwick, Haritsa]). Most of these methods are based on one of the two basic concurrency control mechanisms: locking [Lam] or optimistic concurrency control (OCC) [Kung].

In real-time systems transactions are scheduled according to their priorities. Therefore, high priority transactions are executed before lower priority transactions. This is true only if a high priority transaction has some database operation ready for execution. If no operation from a higher priority transaction is ready for execution, then an operation from a lower priority transaction is allowed to execute its database operation. Therefore, the operation of the higher priority transaction may conflict with the already executed operation of the lower priority transaction. In traditional methods a higher priority transaction must wait for the release of the resource. This is the priority inversion problem presented earlier. Therefore, data conflicts in concurrency control should also be based on transaction priorities or criticalness or both. Hence, numerous traditional concurrency control methods have been extended to the real-time database systems. In the following sections recent and related work in this area is presented.

2.1 Locking-based algorithms

In classical two-phase locking protocol, transactions set read locks on objects that they read, and these locks are later upgraded to write locks for the data objects that are updated. If a lock requested is denied, the requesting transaction is blocked until the lock is released. Read locks Call be shared. while write locks are exclusive.

For real-time database Systems, two-phase locking needs to be augmented with a priority based conflict resolution scheme to ensure that higher priority transactions are not delayed by lower priority transactions. In High Priority scheme, all data conflicts are resolved in favour of the transaction with the higher priority. When a transaction requests a lock on an object held by other transactions in a conflicting lock mode, if the requester's priority is higher than that of all the lock holders, the holders are restarted and the requester is granted

the lock; if the requester's priority is lower, it waits for the lock holders to release the lock. In addition, a new read lock requester can join a group of read lock holders only if its priority is higher than that of all waiting write lock operations. This protocol is referred to as 2PL-HP. It is important to note that 2PL-HP loses some of the basic 2PL algorithm's blocking factor due to the partially restart-based nature of the High Priority scheme.

Note that High Priority scheme is similar to Wound-Wait scheme, which is added to two-phase locking for deadlock prevention. The only difference is that High Priority scheme uses priority order decided by transaction timing constraints for conflict resolution decisions, while Wound-Wait employs timestamp order usually decided by transaction arrival time. It is obvious that High Priority serves as a deadlock prevention mechanism, if the priority assignment mechanism assigns unique priority value to a transaction and does not dynamically change the relative priority ordering of concurrent transactions. Also, note that 2PL-HP is free from priority inversion problem, because a higher priority transaction never waits for a lower priority transaction, but restarts it.

In 2PL-WP (2PL Wait Promote) [Huang] the analysis of concurrency control method is enhanced from [Lindstrom]. The mechanism presented uses shared and exclusive locks. Shared locks permit multiple concurrent readers. A new definition is made – the priority of a data object, which is defined to be the highest priority of all the transactions holding a lock on the data object. If the data object is not locked, its priority is undefined.

A transaction can join in the read group of an object only if its priority is higher than the maximum priority of all transactions in the write group of an object. Thus, conflicts arise from incompatibility of locking modes as usual. Particular care is given to conflicts that lead to priority inversions. A priority inversion occurs when a transaction of high priority requests and blocks for an object which has lesser priority. This means that all the lock holders have lesser priority than the requesting transaction. This same method is also called 2PL-PI (2PL Priority Inheritance) [Stankovic].

Sometimes High Priority may be too strict policy. If the lock holding transaction T_h can finish in the time that the lock requesting transaction T_h can afford to wait, that is within the slack time of T_r, and let T_h proceed to execution and T_r B wait for the completion of T_h. This policy is called 2PL-CR (2PL Conditional Restart) or 2PL-CPI (2PL Conditional Priority Inheritance) [Lam, Menasce].

In Priority Ceiling Protocol [Sha] the aim is to minimize the duration of blocking to at most one elementary lower priority task and prevent the formation of deadlocks. A real-time database can often be decomposed into sets of database objects that can be modelled as atomic data sets. For example, two radar stations track an aircraft representing the local view in data objects O_1 and O_2. These objects might include e. g. the current location, velocity, etc. Each of these objects forms an atomic data set, because the consistency constraints can be checked and validated locally. The notion of atomic data sets is especially useful for tracking multiple targets.

A simple locking method for elementary transactions is the two-phase locking method; a transaction cannot release any lock on any atomic data set unless it has obtained all the locks on that atomic data set. Once it has released its locks it cannot obtain new locks on the same atomic data set, however, it can obtain new locks on different data sets. The theory of

modular concurrency control permits an elementary transaction to hold locks across atomic data sets. This increases the duration of locking and decreases preemptibility. In this study transactions do not hold locks across atomic data sets.

Priority Ceiling Protocol minimizes the duration of blocking to at most one elementary lower priority task and prevents the formation of deadlocks. The idea is that when a new higher priority transaction preempts a running transaction its priority must exceed the priorities of all preempted transactions, taking the priority inheritance protocol into consideration. If this condition cannot be met, the new transaction is suspended and the blocking transaction inherits the priority of the highest transaction it blocks.

The priority ceiling of a data object is the priority of the highest priority transaction that may lock this object [Sha, Rajkumar, Son]. A new transaction can preempt a lock-holding transaction only if its priority is higher than the priority ceilings of all the data objects locked by the lock-holding transaction. If this condition is not satisfied, the new transaction will wait and the lock-holding transaction inherits the priority of the highest transaction that it blocks. The lock-holder continues its execution, and when it releases the locks, its original priority is resumed. All blocked transactions are alerted, and the one with the highest priority will start its execution.

The fact that the priority of the new lock-requesting transaction must be higher than the priority ceiling of all the data objects that it accesses, prevents the formation of a potential deadlock. The fact that the lock-requesting transaction is blocked only at most the execution time of one lower priority transaction guarantees, the formation of blocking chains is not possible [Sha, Rajkumar, Son].

The Priority Ceiling Protocol is further worked out in [Sha, Rajkumar, Son], where the Read/Write Priority Ceiling Protocol is introduced. It contains two basic ideas. The first idea is the notion of priority inheritance. The second idea is a total priority ordering of active transactions. A transaction is said to be active if it has started but not completed its execution. Thus, a transaction can execute or wait caused by preemption in the middle of its execution. Total priority ordering requires that each active transaction execute at a higher priority level than the active lower priority transaction, taking priority inheritance and read/write semantics into consideration.

A protocol called Real-Time Locking (RTL) that uses locking and dynamic adjustment of serialization order for priority conflict resolution was proposed. The basic idea of serialization order adjustment is to delay the decision of final serialization order among transactions, and to adjust the temporary serialization order dynamically in favour of transactions with high priority. Thus, this scheme can avoid blocking and aborts resulting from a mismatch between serialization order and priority order of transactions. In order to implement the dynamic adjustment of serialization order, in RTL, the execution of a transaction is phase-wise as in OCC. In the first phase called read phase, a transaction reads from database and writes to its local workspace as in OCC. However, unlike OCC where conflicts are resolved only in the validation phase, RTL resolves conflicts in the read phase using transaction priority. In the write phase of RTL, the final serialization order is determined, and updates are made permanent to the database. The use of the phase-dependent control and local workspace for transactions also provides potential for a high degree of concurrency.

2.2 Optimistic Concurrency Control-based algorithms

Optimistic Concurrency Control (OCC), is based on the assumption that conflict is rare, and that it is more efficient to allow transactions to proceed without delays to ensure serializability. When a transaction wishes to commit, a check is performed to determine whether a conflict has occurred. There are three phases to an optimistic concurrency control method:

- Read phase: The transaction reads the values of all data items it needs from the database and stores them in local variables. In some methods updates are applied to a local copy of the data and announced to the database system by an operation named pre-write.
- Validation phase: The validation phase ensures that all the committed transactions have executed in a serializable fashion. For a read-only transaction, this consists of checking that the data values read are still the current values for the corresponding data items. For a transaction that has updates, the validation consists of determining whether the current transaction leaves the database in a consistent state, with serializability maintained.
- Write phase: This follows the successful validation phase for update transactions. During the write phase, all changes made by the transaction are permanently stored into the database.

In optimistic concurrency control, transactions are allowed to execute unhindered until they reach their commit point, at which time they are validated. Thus, the execution of a transaction consists of three phases: read, validation, and write. The key component among these is the validation phase where a transaction's destiny is decided. Validation comes in several flavours, but it can carry out basically in either of two ways: backward validation and forward validation. While in backward scheme, the validation process is done against committed transactions, in forward validation, validating of a transaction is carried out against currently running transactions.

As explained above, in RTDBSs, data conflicts should be resolved in favor of higher priority transactions. In backward validation, however, there is no way to take transaction priority into account in serialization process, since it is carried out against already committed transactions. Thus backward scheme is not applicable to real-time database systems. Forward validation provides flexibility for conflict resolution such that either the validating transaction or the conflicting active transactions may be chosen to restart, so it is preferable for real-time database systems. In addition, forward scheme generally detects and resolves data conflicts earlier than backward validation, and hence it wastes less resources and time.

All the optimistic algorithms used in the previous studies of real-time concurrency control in [Haritsa] are based on the forward validation. The broadcast mechanism in the algorithm, OPT-BC used in [Haritsa] is an implementation variant of the forward validation. We refer to the optimistic algorithm using forward validation as OCC-FV.

A point to note is that unlike 2PL-HP, OCC-FV does not use any transaction priority information in resolving data conflicts. Thus, under OCC-FV, a transaction with a higher priority may need to restart due to a committing transaction with a lower priority. Several methods to incorporate priority information into OCC-FV were proposed and studied in

[Lindstrom] using priority-driven wait or abort mechanism. However, more work is needed to ensure if these methods have any significant impact on improving OCC-FV performance, because the effect of increased waiting time or increased number of aborts in these methods may overshadow the performance gain due to the preferential treatment of transactions.

In the OPT-SACRIFICE [Haritsa] method, when a transaction reaches its validation stage, it checks for conflicts with other concurrently running transactions. If conflicts are detected and at least one of the conflicting transactions has a higher priority, then the validating transaction is restarted, i. e. sacrificed in favour of the higher priority transaction. Although this method prefers high priority transactions, it has two potential problems. First, if a higher priority transaction causes a lower priority transaction to be restarted, but fails in meeting its deadline, the restart was useless. This degrades the performance. Second, if priority fluctuations are allowed, there may be the mutual restarts problem between a pair of transactions. These two drawbacks are analogous to those in the 2PL-HP method [Lee].

When a transaction reaches its validation stage, it checks if any of the concurrently running other transactions have a higher priority. In the OPT-WAIT [Lee] case the validating transaction is made to wait, giving the higher priority transactions a chance to make their deadlines first. While a transaction is waiting, it is possible that it will be restarted due to the commit of one of the higher priority transactions. Note that the waiting transaction does not necessarily have to be restarted. Under the broadcast commit scheme a validating transaction is said to conflict with another transaction, if the intersection of the write set of the validating transaction and the read set of the conflicting transaction is not empty. This result does not imply that the intersection of the write set of the conflicting transaction and the read set of the validating transaction is not empty either [Lee].

The WAIT-50 method is an extension of the OPT-WAIT - it contains the priority wait mechanism from OPT-WAIT method and a wait control mechanism. This mechanism monitors transaction conflict states and dynamically decides when and for how long a low priority transaction should be made to wait for the higher priority transactions. In WAIT-50, a simple 50 percent rule is used - a validating transaction is made to wait while half or more of its conflict set is composed of transactions with higher priority. The aim of the wait control mechanism is to detect when the beneficial effects of waiting are outweighed by its drawbacks [Lee].

We can view OPT-BC, OPT-WAIT and WAIT-50 as being special cases of a general WAITX method, where X is the cut-off percentage of the conflict set composed of higher priority transactions. For these methods X takes the values infinite, 0 and 50 respectively.

In [Lee and Son] a lock based WAIT-50 concurrency control method, OCCL-PW, is presented. The physical implementation of this method uses locks. If the priority of the validating transaction is not highest among the conflicting transactions, the validating transaction waits if at least 50% of the conflicting transactions have higher priority.

The OCC-TI method resolves conflicts using the timestamp 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 transaction together with the allocated time slot. Time slots are adjusted when a transaction commits.

In this method, every transaction in the read phase is assigned a timestamp interval (TI). This timestamp interval is used to record a temporary serialization order during the execution of the transaction. At the start of the execution, the timestamp interval of the transaction is initialized as [0, ∞], i. e., the entire range of timestamp space. Whenever the serialization order of the transaction is changed by its data operation or the validation of other transactions, its timestamp interval is adjusted to represent the dependencies.

OCC-DA is based on the Forward Validation scheme. The number of transaction restarts is reduced by using dynamic adjustment of the serialization order. This is supported with the use of a dynamic timestamp assignment scheme. Conflict checking is performed at the validation phase of a transaction. No adjustment of the timestamps is necessary in case of data conflicts in the read phase. In OCC-DA the serialization order of committed transactions may be different from their commit order.

A new optimistic concurrency control method, called OCC-DATI (Optimistic Concurrency Control with Dynamic Adjustment of Serialization Order using Timestamp Intervals), is proposed to reduce the number of transaction restarts in [Lindstrom], which uses information about the criticality of the transactions in the conflict resolution. The main idea behind this method is to offer better chances for critical transactions to complete according to their deadlines. This is achieved restarting transaction with lower criticality if the critical transaction should be restarted because of a data conflict. The proposed method is demonstrated to produce the correct results and the feasibility of the proposed method in practice is tested.

3. System model and real-time data

3.1 System model

Active real-time database consists of a set of objects and ECA (Event-Condition-Action) rule. Each object represents a real world entity; the status of entity is usually monitored by sensors. The entire data object is divided into two categories in database:

1. temporal objects and non-temporal objects

The temporal object is possible to be invalid for expired temporal validity; it can be divided into the absolute validity and the relative validity. Non-temporal object have not temporal validity. Temporal object absolute validity expresses as (value (Xi) , avi (Xi)) , value (Xi) describes the current status of data object X, $avi(Xi)=[avi_b(Xi), avi_e(Xi)]$, $avi_b(Xi) \le avi_e(Xi)$ is the absolute validity of value (Xi), Xi shows the ith versions of data objects X, $avi_b(Xi)$ denotes the beginning of the absolute validity of Xi , $avi_e(Xi)$ denotes the end of the absolute validity of Xi , after $avi_e(Xi)$, value (Xi) is effective no longer.

Temporal object relative validity denotes rvi(R), R is relatively consistent set, each element is the version temporal data objects. When t>0, R is correct status, if and only if

- $Xi \in R$, value(Xi)is logical consistent, satisfying all the integrity constraints
- R is temporal consistent:

i. $Xi \in R$, value(Xi) is absolute consistent, $avi_b(Xi) \le t \le avi_e(Xi)$.
ii. R is relatively consistent, $Xi, Yj \in R$, $|avi_b(Xi)-avi_b(Yj)| \le rvi(R)$.

2. the basic objects and derived objects

The basic objects update the database by the sensor and reflect specific entity in the real environment. Derived objects are composed by the new derived data from basic object and others. In Active real-time database, it can trigger transactions according to ECA rules when basic object is updated by sensor. Triggered transactions may update the status from the basic derived objects. In this chapter, the basic object take need scheduling strategy, until transactions need call, it was called by sensor. Transactions T call temporal data X at time t, absolute validity for the start time is t.

Active real-time database system had sensor transaction, which are used to update basic objects, only writing transaction; triggered updating transaction, which are updated and triggered transaction by the basic objects, and used to update derived object; user transaction, which have user transactions of the deadline. When transaction satisfies only the following conditions in system, it commits successfully:

1. Transaction is consistent logically.
2. Transaction satisfies the deadline.
3. The data that transaction read is temporal consistent, and the data still effective when the transaction read it commits.

3.2 Real-time data

In [Kuo, Mok], relative sensor transaction and trigger who used to update the derived object are discuss in detail, this chapter studies only concurrency control related users transaction, similar with [Liu, Son]. User transactions T have the following attributes:

- $a(T)$: Arrival time of transaction T;
- $s(T)$: Start time of transaction T;
- $d(T)$: Deadline of transaction;
- $dd_t(T)$: Data-deadline of transaction T at time t;
- $L(T)$: The number of data objects which transaction T access;
- $L_t^{to}(T)$: The number of temporal data objects that transaction T will access after time t;
- $L_t^{nto}(T)$: The number of non-temporal data objects that transaction T will access after time t;
- $L_t(T)$: The number of data objects which transaction T will access after time t; $L_t(T) = L_t^{to}(T) + L_t^{nto}(T)$;
- $E_t(T)$: Estimated execution time of transaction T at time t;
- $C_t(T)$: Estimated finished time of transaction T at time t;
- $C_t(T) = t + E_t(T)$;
- $RS_t^{to}(T)$: Temporal data object sets which transaction T access at time t ;
- $P_t(T)$: The transaction priority at time t.

In active real-time database, temporal data have lots of features; this section give only related properties which presented algorithm in this chapter. First of all, according to [Son] this article introduces the concept of the data-deadline.

Definition1. Data-deadline of transaction T is the minimum data time validity which transaction T access to the temporal data objects at time t, denote it as $dd_t(T)$,

$$dd_t(T) = \min_{X \in RS_t^{to}(T)} avi_e(X)$$

The temporal relationship between temporal data and transaction is presented by data-deadline. The data-deadline increased the difficulty of transaction scheduling and concurrent control, because no achieving deadline to the transactions may restart or abort due to the data deadline. Example 1 shows that the data-deadline of transactions influences the concurrent control.

Example 1. Transaction T1: w1(x)r1(y); T2:w2(y)r2(z).

The deadline of transaction T1 is t7, and its estimated time of completion is t6. The deadline of transaction T2 is t8; and its estimated time of completion is t6. The validity of temporal data z is [t4, t6].

As shown in Fig. 1, transaction T2 enter into validation phase at time t5, according to literature [Lindstrom, Wang], WS(Tv)∩RS(Ta)={y}≠Ø, transactions execution sequence is T1→T2, T2 will delay to submit. If transaction T1 submits after t6, transaction T2 will die for the temporal data z exceed deadline.

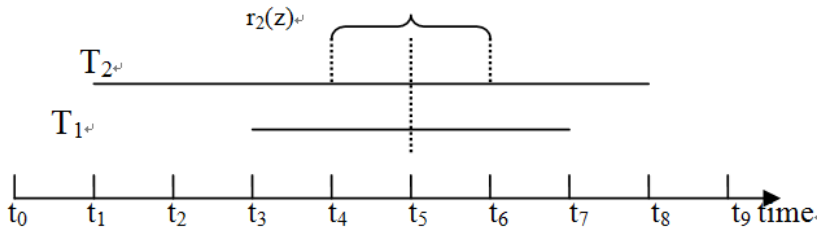


Fig. 1. Execution of Transactions

Do not consider real-time of data, we will succeed scheduling T1, T2 , according to the traditional optimistic method[Lindstrom]; But it increases the real-time nature of data, transaction T2 will die because it can't satisfy data-deadline. Example1 shows the temporal characteristics of the data influence the concurrency control, while the existing concurrency control method is no consideration to the conflict which access temporal data.

In addition to data-deadline, there is another important characteristic is the stability of the data. Real-time data has different rate of change, some change frequently and others not. If $|avi_e(X) - avi_b(X)| < k$, we denoted temporal data as changeful, otherwise stable. According to the different time limit of transactions, the value k dynamic changes along with the transaction access to temporal data, it will be discussed in detail in the next section.

4. Related definition and validation mechanism

As another temporal limit of transaction, data-deadline, if it is less than transaction-deadline, we must consider it; otherwise transactions may read inconsistent temporal data when transactions commit. In order to ensure temporal consistency and the data which transaction read to be effective, we must inspect and estimate the execution time.

Definition2. Transaction-deadline is $d(T)$, estimated time of completion is $C_t(T)$, so delay time is $sd_t(T) = d(T) - C_t(T)$. Because validation transaction has been completed all operation reading and writing, delay time is $sd_t(T) = d(T) - C_t(T)$.

Definition3. If $L_t^{to}(T) = \phi$, we denoted $tsd_t(T)$ as temporal delay time of transaction T.

$$tsd_t(T) = \begin{cases} d(T) - C_t(T), & dd_t(T) \geq d(T) \\ dd_t(T) - C_t(T), & dd_t(T) < d(T) \end{cases}$$

Definition4. At time t, the starting time of transaction T is $s(T)$, the deadline of transaction T is $d(T)$; we denoted $TFD_t(T) = t - s(T)/d(T) - s(T)$ as the completeness of transaction T.

Definition5. Suppose the conflict transaction set of validation transaction Tv is CTS(T), validation factor of Tv shows the ratio between the completeness of Tv and the completeness of CTS(T), denoted $VF_t(T_v) = TFD_t(T_v)/TFD_t(T)$, $T \in CTS(T)$. If $VF_t(T_v) > 1$ at time t, it shows validation transaction is easier to complete t than conflict transaction.

Definition6. Minimum running time of transaction was described from the current time t to the required execution time of transaction.

If the deadline which transaction read temporal data objects is less than minimum running time, transaction can't commit in validity, so the validity is impossibility. Before transaction read data, we use a validation mechanism to check the validity of the data. It is used to prevent transaction from reading invalid data, or the data which has not validity.

Before transaction executing, we declare that transaction access temporal data set $RS_{s(T)}^{to}(T)$ in advance, the number of temporal data is $L_{s(T)}^{to}(T)$. After transaction executing, we access temporal data successively and check the validity of data dynamically in the process. If $dd_t(T) > d(T)$, and $L_t^{to}(T) = \phi$, data-deadline would be longer than transaction-deadline, and transaction can't access other temporal data, so it can't affect the scheduling and concurrency control. If $dd_t(T) > d(T)$, but $L_t^{to}(T) \neq \phi$, transaction will access other temporal data, so we must check whether satisfy $dd_t(T) > d(T)$ or not, when access all the temporal data. CHECKING Algorithm was described as follows.

CHECKING Algorithm ():
INPUT: $RS_{s(T)}^{to}(T) = \{X^1, X^2 ... X^m\}$
OUTPUT: k, $L_t^{to}(T)$
{ $k = \infty$; $N = L_{s(T)}^{to}(T)$; i=1;
While ($RS_t^{to}(T) \neq \varnothing$)
{T accesses Xi from RS(T) ;

$RS_t^{to}(T) = \{X^1, X^2 ... X^m\} - \{X^i\}$;
if(| avie (Xi) −avib (Xi) |<k)
then if $dd_t(T) < C_t(T)$
then Abort(T);
else $k = k \bigcap_{j=1}^{i} | avi_e(X^j) - avi_b(X^j) |$;}
$L_t^{to}(T) = N - 1$;
i=i+1;
}

Before transaction access each temporal data, we call CHECKING Algorithm to ensure the effectiveness of temporal data. At the same time, through dynamically adjusting value k, we make the variable data consistency.

Lemma1. CHECKING Algorithm can ensure the consistency of variable temporal data.

If | avi_e (X) −avi_b (X) | < k , we denoted temporal data as changeful, otherwise stable. Value k denoted the length of absolute validity; it changed dynamically along with the transaction access to temporal data, and the smaller the k value is shows the more unstable temporal data is. If temporal data which transaction access the next is more changeful than the current, | avi_e (Xi) −avi_b (Xi) |<k , we restart to check whether the data can submit before the deadline, if it can satisfy the temporal consistency, value k will be changed the absolute valid length of temporal data. So CHECKING Algorithm check the consistency of each of variable data to ensure transaction to submit correctly.

Theorem1. CHECKING Algorithm can ensure the consistency of transaction scheduling temporal data.

Prove. Induce the number of temporal data n(T) which transaction T need access, When n(T)=1, proposition was established obviously, otherwise transaction will abort. Supposed n(T)=m, proposition was established. When n(T)=m+1, it can be divided into two cases:

1. When | avi_e (Xi) −avi_b (Xi) | ≥ k , it must satisfy the consistency of temporal data. Supposed that transaction T access temporal data Xi at time t, according to the need of the scheduling strategy, avi_b (Xi) = t, so $avi_e(X^i) \geq dd_t(T)$. Supposed n(T)=m, proposition was established, $dd_t(T) \geq C_t(T)$, so $avi_e(X^i) \geq C_t(T)$. So before transactions complete, all the data can satisfy the time limit proposition was established.
2. When | avi_e (Xi) −avi_b (Xi) |<k , the length of the absolute valid which transaction read the next temporal data is less than k. Showing the accessing data is changeful, and compute the value $dd_t(T)$, on the basis of Lemma1, it satisfies temporal consistency of data. Proposition was established. Sum up(1), (2), the proposition is established.

5. Real-Time concurrency control protocol based on accessing temporal data (RTCC-DD)

The concurrency control mechanism in the database must guarantee the consistency of the database; serializability is one correctness standard of concurrency control in the database.

Methods discussed in this chapter are based on the optimistic concurrency control strategy, when the transactions reading stage we use CHECKING Algorithm to guarantee access temporal data consistency. When transactions come into the verify stage, we make the reading and writing adjustment detection, at this stage the changes of the transactions of the database is readable and effective, conflict transactions can read the private cache of transactions to get the change. The transactions submit if all conflicts transactions are serializable. Specific adjustment rules are as follows.

Rule1 At time t, when Lt(T)=0, assigned serial number for the transaction T ser(T) and enter the validation phase, the scope of ser (T) is 1, 2 . . .. to n. The number of the first transaction access the validation phase is 1, transactions access later cumulative.

Rule2 At time t, $avi_e(Xi)<C_t(T)$ and the value of the ith version of X is similar with the next version, then adjust the X temporal period to the next version, that temporal object is (value (Xi) , avi (Xi)) , $avi(Xi)=[avi_b(Xi), avi_e(Xi+1)]$.

Rule3 At time t, $VF_t(T)<1$ and $tsd_t(Tv) > tsd_t(Ti)$, $T_i \in CTS(T_v)$ adjust the serialization order of transactions, ser(Ti)=ser(Tv), ser(Tv)=ser(Tv)+1. If $RS(Tv)\cap WS(Ti)\neq\emptyset$, Tv read the data of Ti from private cache.

Rule4 At time t, submit transaction T, if only $\forall T_i \in CTS(T)$, $ser(T) < ser(T_i)$.

Rule1 ensures transactions enter into validation phase and distribute serial number after complete to access data. Rule2 ensures to extend the validity of temporal data valid interval if next version data similar with the current version, and reduces unnecessary transaction to restart. Rule3 ensures to run conflict transaction when the conflict of transaction is more nearer completion than validation transaction, and validation transaction can delay after the conflict transaction committed, and adjust the executive order, when verification transaction is conflict with other transaction, allow validation transaction to read data which conflict transaction update. Rule4 ensures that the submitted transactions are serializability.

To ensure the effectiveness of the temporal data, RTCC-DD method adopts the checking algorithm to check the access data set before transactions executing. And then, using an optimistic approach, adjust the rules of the transaction validation phase. When verify transaction and other transaction are conflict, to judge by the factor of authentication, priority to scheduling the transaction which will be finished; taking consider the state that verify the transaction when the delay time dynamic adjustment of transaction execution in order to ensure the transactions of temporal data scheduling to satisfy the temporal consistency. While it ensured the limit of the data and transaction, minimizing the unnecessary transaction restarts. RTCC-DD concurrency control methods are described below:

if T_v conflicts with T_i, $T_i \in CTS(T_v)$
  if $RS(T_v)\cap WS(T_i)\neq\emptyset$ then
    if $(VF_t(T_v) < 1) \wedge (tsd_t(T_v) > tsd_t(T_i))$
    then adjusts the execute order to T_i, T_v
    else the execute order is T_v, T_i;
  else if $RS(T_i)\cap WS(T_v)\neq\emptyset$ then

if $(VF_t(T_v) \geq 1) \wedge (tsd_t(T_i) > tsd_t(T_v))$
then adjusts the excute order to T_v, T_i
else the execute order is T_i, T_v;
else if $WS(T_i) \cap WS(T_v) \neq \emptyset$ then
if $(VF_t(T_v) < 1) \wedge (tsd_t(T_v) > tsd_t(T_i))$
then adjusts the execute order to T_i, T_v
else the execute order is T_v, T_i;
endif
endif

RTCC-DD method dynamically adjusts the data by checking the value of k, for each variable temporal data are calculated to ensure that the scheduling of variable data; For the next version of similar state cannot meet the time limit to extend the data deadline for its validity; at the same time we using the optimistic concurrency control method, this method won't produce priority inversion and congestion; And it is the first time to solve the problem about temporal data of concurrency control. Improved the problem of which is discussed in the literature about the transaction scheduling problems of access to temporal data.

Example 2. The transaction and implementation in example 2 is the same with example 1, transaction T1: w1(x)r1(y); T2: w2(y) r2(z). The period of validity of T1 is t7, the period of validity of temporal data z is [t4, t6].

At time t5 when T2 access to validation phase, there is WS(Tv)∩RS(Ta)={y}≠Ø, according to the RTCC-DD,

$$VF_t(T_v) = \frac{4}{7} \Big/ \frac{2}{4} = \frac{8}{7} > 1\ ,\ tsd_{t_5}(T_1) = t_7 - t_6\ ,\ tsd_{t_5}(T_2) = t_6 - t_6\ ,$$

thus satisfying the conditions $(VF_t(T_v) \geq 1) \wedge (tsd_t(T_i) > tsd_t(T_v))$, Adjust the order of transaction execution to T2, T1. After the submit of T2, T1 can also meet the constraint of the period of validity, scheduling T2, T1 successfully, so use method RTCC-DD, can avoid transaction unnecessary restarts.

Serializability is one standard of correctness of concurrency control in the database. RTCC-DD can guarantee the concurrency control of temporal data, at the same time satisfy the serializability. We discuss the superiority and accuracy problems of RTCC-DD method as below.

Lemma2 The traditional optimistic concurrency control methods can use RTCC-DD to schedule.

Because the time constraint of access to transaction will influence the scheduling and concurrency control, The traditional optimistic concurrency control methods only consider the case of the transaction deadline, not discuss the deadline for data on the effect to transaction concurrency control. In order to solve the problem, RTCC-DD take the optimistic method; if the scheduling data is not temporal constrain, RTCC-DD will degenerate to the traditional optimistic concurrency control methods, so the traditional optimistic concurrency control methods can use RTCC-DD to scheduling.

Theorem 2 RTCC-DD can guarantee the serializability of the transactions scheduling

Proof, according to the rule 1, if $L_t(T) = 0$, assigned serial number for the transaction T ser(T), guarantee only when the transaction access into the validation phase we assigned serial number, and the serial number is only, according to rule 3 we dynamic adjust the conflict transaction, and guarantee the least serial number first submit. According to the rule 4, only when all the conflict transactions serialize after this transaction, submit this transaction, so submitting the transactions is constrain by serializability number, ensure the serializability.

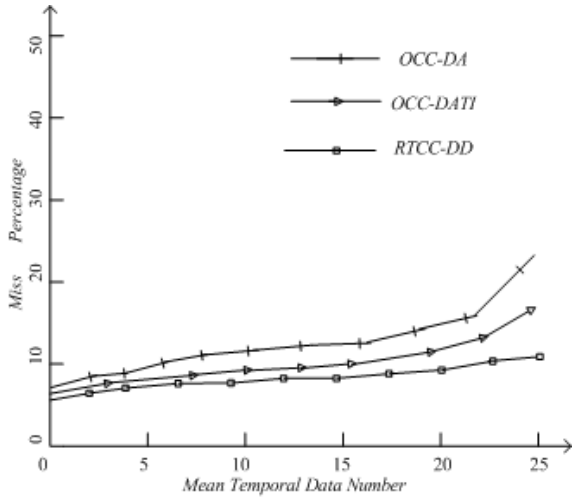
6. Experiment

A set of experiments have been carried out in order to examine the feasibility of the algorithms in practice. The RTCC-DD method and the OCC-DA, OCC-DATI method are implemented using C++ and compiled with gcc. All tests were performed on a Dell OptiPlex GX270 PC with 2GB of RAM. The database is composed of pages with a number of records and the records meet uniform distribution; the main performance criteria are the transaction miss percentage.

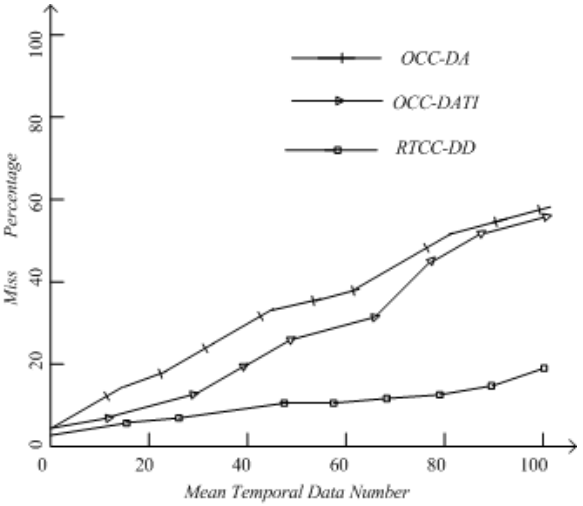
parameter	*value*
Database size	*400*
Transaction execute time	*100ms*
CPU computation time	*10ms*
Disk access time	*20ms*
Page hit rate	*80%*
Operation number per transaction	*5~50*
Restart overhead	*10*
Mean transaction arrive time	*10~200ms*

Table 1. Simulation Parameter

The traditional optimistic concurrency control methods (OCC-DA, OCC-DATI) and the RTCC-DD method are compared in the experiment. The experiment results are shown in Fig. 2. In Fig. 2a, the average temporal data number of transactions accesses changing from 0 to 25, the performance of RTCC-DD is better than OCC-DA and OCC-DATI. The reason is that transactions scheduling considers the deadline of data in RTCC-DD method. If the submitted time exceeds the deadline, the transaction will be restarted. And the methods of OCC-DA and OCC-DATI do not consider the constraint of temporal data. When transaction accesses a little temporal data, the difference of the performance are small, but as shown in Fig. 2b, if there are a lots of temporal data, RTCC-DD method is obvious better than the traditional concurrency control methods. With increasing the temporal data numbers in system, the influence of the time constrain will increase. The more transactions will miss its deadlines because of not satisfy the data deadline.



a



b

a. Little Number of Temporal Data b. Much Number of Temporal Data

Fig. 2. Transactions Miss Percentage

7. Conclusion

The concurrency control method is one of the key problems in the database systems. The optimistic methods are widely used in database system because it is not exists deadlock and block. Unnecessary transactions restarting and nearing completing transactions missing its deadline are the key factor effecting optimistic concurrency control method performances.

Most of the research about the optimism concurrency control method are focus on how reduces unnecessary transactions restarts and the transactions near to completed missing its deadline. The most research is based on dynamic adjustment serialization method. When the transactions access temporal data with time limit, the traditional concurrency control method cannot schedule effectively because it is not consider data-deadline. This chapter improved the validation phase rules and proposed an optimistic concurrency control method based on temporal data (RTCC-DD), which considered the influence between temporal data time limit and the transaction deadline. Theoretical analysis and experimental results demonstrate that the RTCC-DD method can outperform the previous ones for reducing effectively unnecessary restart number of transactions and more suitable for real-time database system.

8. Acknowledgment

The chapter is sponsored by the National Natural Science Foundation of China under Grant No. 41176082, 61073182; Heilongjiang Natural Science Foundation under Grant No. F201024; The Fundamental Research Funds for the Central Universities No. HEUCFZ1010, HEUCF100602.

9. References

A. Brad, K. Ben and G. M. Hector. Database support for efficiently maintaining derived data. Technical Report, Stanford University, 1995: 223~240.

A. Fishwick. SIMPACK: Getting started with simulation programming in C and C++. Department of Computer & Information Science, University of Florida, 1992. IEEE Computer Society Press

D. Menasce and T. Nakanishi. Optimistic versus pessimistic concurrency control mechanisms in database management systems. Information Systems, 7(1):13–27, 1982.

H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. ACM Transactions on Database Systems, 6(2):213–226, June 1981.

J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley. Experimental evaluation of real-time optimistic concurrency control schemes. In Proceedings of the 17th VLDB Conference, pages 35–46, Barcelona, Catalonia, Spain, September 1991. Morgan Kaufmann.

J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley. On using priority inheritance in real-time databases. In Proceedings of the 12th IEEE Real-Time Systems Symposium, pages 210–221, San Antonio, Texas, USA, 1991. IEEE Computer Society Press.

J. Lee and S. H. Son. Using dynamic adjustment of serialization order for real-time database systems. In Proceedings of the 14th IEEE Real-Time Systems Symposium, pages 66–75, Raleigh-Durham, NC, USA, 1993. IEEE Computer Society Press.

J. Lee. Concurrency Control Algorithms for Real-Time Database Systems. PhD thesis, Faculty of the School of Engineering and Applied Science, University of Virginia, January 1994.

J. Lindstrom and K. Ratikainen. Dynamic adjustment of serialization order using timestamp intervals in real-time databases. In Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications, pages 13–20, Hong Kong, China, 1999. IEEE Computer Society Press.

J. Lindstrom. Optimistic concurrency control methods for real-time database systems. Ph. D. dissertation. University of Helsinki, FINLAND, 2003.

J. R. Haritsa, M. J. Carey and M. Livny. Dynamic real-time optimistic concurrency control. In Proceedings of the 11th real-time symposium, 1990. 94~103.

J. R. Haritsa, M. J. Carey, and M. Livny. Dynamic real-time optimistic concurrency control. In Proceedings of the 11th IEEE Real-Time Systems Symposium, pages 94–103, Lake Buena Vista, Florida, USA, 1990. IEEE Computer Society Press.

J. R. Haritsa, M. J. Carey, and M. Livny. On being optimistic about real-time constraints. In Proceedings of the 9th ACM Symposium on Principles of Database Systems, pages 331–343, Nashville, Tennessee, 1990. ACM Press.

K. Ramamritham. Real-time databases. Distributed and Parallel Databases, 1:199–226, April 1993.

K. W. Lam, K. Y. Lam, and S. Hung. An efficient real-time optimistic concurrency control protocol. In Proceedings of the First International Workshop on Active and Real-Time Database Systems, pages 209–225, Sk¨ovde, Sweden, 1995. Springer.

K. W. Lam, S. H. Son, and S. Hung. A priority ceiling protocol with dynamic adjustment of serialization order. In Preceedings of the 13th IEEE Conference on Data Engineering, Birmingham, UK, 1997. IEEE Computer Society Press.

L. Sha, R. Rajkumar, and J. P. Lehoczky. Concurrency control for distributed real-time databases. ACM SIGMOD Record, 17(1):82–98, March 1988.

L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. IEEE Transactions on Computers, 39(9):1175–1185, September 1990.

L. Sha, R. Rajkumar, S. H. Son, and C. -H. Chang. A real-time locking protocol. IEEE Transactions on Computers, 40(7):793–800, January 1991.

M. Xiong, R. M. Sivasankaran, J. A. Stankovic, K. Ramamritham and D. Towsley. Scheduling transactions with temporal constraints: exploiting data semantics. 17th IEEE Real-time Systems Symposium. 1996.

S. H. Son, K. J. Lin. Real-Time Database Systems: Issues and Applications. Kluwer Academic Publishers, 1997. 167-191

T. Kuo and A. K. Mok. Real-time data semantics and similarity-based concurrency control. IEEE Transactions on Computers. 2000, 49(11): 1241~1254.

T. Kuo, A. K. Mok, SSP: a semantics-based protocol for Real-time data access. 14th IEEE real-time systems symposium. 1993.

Y. S. Liu, G. H. Li. The effect of real-time database data characteristics on transactions. Journal of computer research & development. 1999, 36(3): 364~368.

Y. Y. Wang, Q. Wang, H. A. Wang. Dynamic adjustment of execution order in real-time database. In Proceedings of 18th International Parallel and Distributed Processing Symposium, 2004. 1219~1225.

Real-Time Systems, Architecture, Scheduling, and Application

Edited by Dr. Seyed Morteza Babamir

ISBN 978-953-51-0510-7
Hard cover, 334 pages
Publisher InTech
Published online 11, April, 2012
Published in print edition April, 2012

This book is a rich text for introducing diverse aspects of real-time systems including architecture, specification and verification, scheduling and real world applications. It is useful for advanced graduate students and researchers in a wide range of disciplines impacted by embedded computing and software. Since the book covers the most recent advances in real-time systems and communications networks, it serves as a vehicle for technology transition within the real-time systems community of systems architects, designers, technologists, and system analysts. Real-time applications are used in daily operations, such as engine and break mechanisms in cars, traffic light and air-traffic control and heart beat and blood pressure monitoring. This book includes 15 chapters arranged in 4 sections, Architecture (chapters 1-4), Specification and Verification (chapters 5-6), Scheduling (chapters 7-9) and Real word applications (chapters 10-15).

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Qilong Han (2012). Real-Time Concurrency Control Protocol Based on Accessing Temporal Data, Real-Time Systems, Architecture, Scheduling, and Application, Dr. Seyed Morteza Babamir (Ed.), ISBN: 978-953-51-0510-7, InTech, Available from: http://www.intechopen.com/books/real-time-systems-architecture-scheduling-and-application/real-time-concurrency-control-protocol-based-on-accessing-temporal-data

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821