Combined Locking Approach for Scheduling Hard Real-Time Transactions in Real-Time Databases

Qiang Wang, Hong-An Wang and Guo-Zhong Dai

Institute of Software, Chinese Academy of Sciences
P.O.Box 8718, Beijing 100080, China
wq@iel.iscas.ac.cn

Abstract

Previous work has shown the superiority of the optimistic protocols over the lock-based protocols for scheduling soft or firm real-time transactions. However, optimistic protocols cannot provide schedulability analysis for hard real-time transactions because of uncertain transaction restarts. In this paper, we develop new combined locking approach for using optimistic concurrency control to schedule hard real-time transactions. This approach can resolve serious conflicts, which cannot be resolved via dynamic adjustment of serialization order using timestamp intervals, and thus avoid transaction restarts. Furthermore, the combined locking approach can be integrated with the priority ceiling mechanism to achieve single-blocking and deadlock-free properties, and perform schedulability analysis for hard real-time transactions.

1. Introduction

There has been growing interest in the performance of transaction systems that have significant response time requirements. These requirements are usually specified as deadlines on individual transactions and a concurrency control algorithm must attempt to meet the deadlines as well as preserve data consistency. The most serious problem of the traditional concurrency control protocols in real-time databases is priority inversion [1]. In order to alleviate this situation, various conflict resolution strategies have been suggested to resolve data conflicts for different types of real-time transactions, e.g., soft, firm, and hard real-time transactions. The strategies are mainly based on transaction restart or data reservation. The well-known lock-based mechanism in controlling the priority inversion problem is priority ceiling protocol (PCP), which is originally used for scheduling periodic tasks in hard real-time systems and has been extended for hard real-time transactions [1,2]. Although the deadlines of all the hard real-time transactions can be guaranteed, PCP is not suitable for processing soft and firm real-time transactions due to the dynamic and unpredictable behaviors of these transactions [3].

In addition to the lock-based protocols, many real-time concurrency control protocols are based on the optimistic method, in which the resolution of data conflicts is delayed until a transaction has finished all of its operations. A validation test is performed to ensure that the resulted schedule is serializable, and a transaction, which passes the test, is allowed to commit. Various forward validation schemes, such as wait-50 and sacrifice, were proposed [4,5]. In particular, previous work [4,5,6] has shown that the superiority of the optimistic protocols over the locked-based protocols for firm and soft real-time protocols. And the OCC-DATI [7,8], which supports dynamic adjustment of serialization order using timestamp intervals, is the best optimistic concurrency control protocol available.

Concurrency control protocols normally employ transaction blocking to resolve data conflicts among transactions for data consistency [10]. Since the lock-based protocols tend to bring unnecessary transaction blockings, and the optimistic methods increase the concurrency degree of transaction execution at the cost of transaction restarts, it is a favourable approach to combine the lock-based protocols with the optimistic methods for scheduling hard real-time scheduling, so as to reduce unnecessary transaction blockings and avoid restarting the conflicting transactions. In this paper, we proposed a combined locking approach for resolving transaction restarts in optimistic concurrency control protocols. This approach can resolve serious conflicts, which cannot be resolved via dynamic adjustment of serialization order using timestamp intervals, and thus avoid both transaction restarts existing in optimistic concurrency control protocols, and unnecessary transaction blockings existing in lock-based protocols. Furthermore, the combined locking approach can be integrated with the priority ceiling mechanism to achieve single-blocking and deadlock-free properties, and perform schedulability analysis for scheduling hard real-time transactions.

2. The combined locking approach

The predictability will be possible only if the system has the comprehensive knowledge of transactions. For most hard real-time systems, application semantics are well known, and data requirements and execution characteristics of the 'canned' transactions should be available. Otherwise, hard deadlines of transactions cannot be guaranteed. Therefore, we assume that a transaction system consists of a fixed set of transactions, i.e., $\Gamma = \{T_i \mid i=1,2\ldots n\}$. We are interested in the context of uniprocessor priority-driven preemptive scheduling. The real-time database can be either memory-resident or disk-resident.

The execution of a transaction T_i consists of three phases: read phase, validation phase and write phase as in optimistic concurrency control protocols. As we known, abortion strategy used in optimistic concurrency control protocols can reduces the transaction blocking time at the expense of restart overheads. However, it complicates and even disables the system schedulability analysis. Therefore, a concurrency control protocol for hard real-time transactions normally employs blocking to resolve data conflicts among transactions for data consistency. For using optimistic concurrency control methods to schedule hard real-time transactions, we must analyze that under what situations transaction restarts will occur inevitably. The combined locking approach is proposed to prevent these situations from occurring.

Notations:

$RS(T_i)$ denotes the read set of transaction T_i.

$WS(T_i)$ denotes the write set of transaction T_i.

$r_i[x]$ denotes that transaction T_i reads a data object x.

$w_i[x]$ denotes that transaction T_i writes a data object x.

$r\_lock_i[x]$ denotes that transaction T_i read-locks a data object x.

$w\_lock_i[x]$ denotes that transaction T_i write-locks a data object x.

2.1 Dynamic adjustment of serialization order

Suppose there are a validating transaction T_v and a set of active transactions T_j (j=1,2…m). There are three possible types of data conflicts that can induce a serialization order between T_v and T_j:

1) $RS(T_v) \cap WS(T_j) \neq \emptyset$ (read-write conflict)

A read-write conflict between T_v and T_j can be resolved by adjusting the serialization order between T_v and T_j as $T_v \rightarrow T_j$ so that the read of T_v cannot be affected by the write of T_j. This type of serialization adjustment is called forward ordering or forward adjustment.

2) $WS(T_v) \cap RS(T_j) \neq \emptyset$ (write-read conflict)

A 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 or backward adjustment.

3) $WS(T_v) \cap WS(T_j) \neq \emptyset$ (write-write conflict)

A write-write conflict between T_v and T_j can be resolved by adjusting the serialization order between T_v and T_j as $T_v \rightarrow T_j$ so that the write of T_v cannot overwrite the write of T_j through forward ordering.

In the OCC-DATI protocol, if an active transaction has to be both backward and forward adjusted with respect to the validating transaction, a serious conflict is said to occur and some of the conflicting transaction has to be restarted. On the other hand, those active transactions, which need only either backward adjustment or forward adjustment, are allowed to continue their execution.

From the above discussion, we can find that there are three situations, which may result in transaction restarts:

1) $RS(T_v) \cap WS(T_j) \neq \emptyset$ and $WS(T_w) \cap RS(T_j) \neq \emptyset$

Let us consider the following transactions T_j, T_v, T_w and history:

T_j: $r_j[x]\ w_j[y]\ v_j\ c_j$

T_v: $r_v[y]\ v_v\ c_v$

T_w: $w_w[x]\ v_w\ c_w$

H_1: $r_j[x]\ w_w[x] v_w\ c_w\ r_v[y]\ v_v\ c_v\ w_j[y]\ v_j$

Based on the OCC-DATI protocol, T_j must be restarted. Similarly, consider T_j and another transaction $T_{v'}$:

$T_{v'}$: $r_{v'}[y]\ w_{v'}[x]\ v_{v'}\ c_{v'}$

H_2: $r_j[x]\ r_{v'}[y]\ w_{v'}[x]\ v_{v'}\ c_{v'}\ w_j[y]\ v_j$

In this case, T_j has to be restarted, too.

2) $WS(T_v) \cap RS(T_j) \neq \emptyset$ and $WS(T_w) \cap WS(T_j) \neq \emptyset$

Consider the following history H_3 over T_j, T_v and T_w, and history H_4 over T_j and $T_{v'}$:

T_j: $r_j[x]\ w_j[y]\ v_j\ c_j$

T_v: $w_v[y]\ v_v\ c_v$

T_w: $w_w[x]\ v_w\ c_w$

$T_{v'}$: $w_{v'}[x]\ w_{v'}[y]\ v_{v'}\ c_{v'}$

H_3: $r_j[x]\ w_w[x]\ v_w\ c_w\ w_v[y] v_v\ c_v\ w_j[y]\ v_j$

H_4: $r_j[x]\ w_{v'}[x]\ w_{v'}[y]\ v_{v'}\ c_{v'}\ w_j[y]\ v_j$

It can be found that, T_j has to be restarted in both histories by the OCC-DATI protocol.

3) $WS(T_v) \cap RS(T_j) \neq \emptyset$ and $WS(T_w) \cap RS(T_j) \neq \emptyset$

Consider the following history H_5 over T_j, T_v and T_w, and history H_6 over T_j and $T_{v'}$:

T_j: $r_j[x]\ r_j[y]\ v_j\ c_j$

T_v: $w_v[y]\ v_v\ c_v$

T_w: $w_w[x]\ v_w\ c_w$

$T_{v'}$: $w_{v'}[x]\ w_{v'}[y]\ v_{v'}\ c_{v'}$

H_5: $r_j[x]\ w_w[x]\ v_w\ c_w\ w_v[y] v_v\ c_v\ r_j[y]\ v_j$

H_6: $r_j[x]\ w_{v'}[x]\ w_{v'}[y]\ v_{v'}\ c_{v'}\ r_j[y]\ v_j$

It can be found that, T_j has to be restarted in both histories by the OCC-DATI protocol.

The essential reason for T_j's restart is that, the serialization order of T_j has to be adjusted forward by conflicting transaction T_v after backward adjustment by conflicting transaction T_w. If T_v and T_w may induce T_j to restart, we consider that there exists an induce-to-restart relationship. Without question, all the induce-to-restart relationships among a transaction set can be determined by making an analysis on the read sets and write sets of

these transactions. To avoid potential transaction restarts, the induce-to-restart relationships must be broken.

2.2 Formal analysis

We first define three kinds of dependency relationships in serializable order that capture read/write conflicts of two different transactions on the same data object.

It is said that an active transaction T_j read-depends on committed transaction T_v if T_v writes some data object x and T_j reads the version of x that is just installed into the database by T_v. An active transaction T_j write-depends on committed transaction T_v if T_v installs a version of x and T_j installs x's next version in the database. On the other hand, if an active transaction T_j reads some data object x and committed transaction T_v writes the next version of x, it is said that T_j anti-read-depends on T_v in serializable history. If an active transaction T_j writes some data object x and committed transaction T_v reads the previous version of x, it is said that T_j anti-write-depends on T_v in serializable history.

It can be found that the induce-to-restart relationships come from different combination of dependency relationships. Different strategies can be used to avoid the above three situations:

1) If T_j anti-read-depends on T_w, we must prevent from T_j anti-write-depending on a transaction T_v including T_w.
2) If T_j anti-read-depends on T_w, we must prevent from T_j write-depending on a transaction Tv including T_w.
3) If T_j anti-read-depends on T_w, we must prevent from T_j read-depending on a transaction T_v including T_w.

In these situations, the transaction T_j is denoted as potential restarted transaction, T_w and T_v are denoted as trigger transaction and firer transaction respectively. The trigger and firer transaction may be the same transaction.

Now we define the Dependency Relation Graph or DRG. Each data object x can be represented as a node, with two kinds of marks – read marks and write marks. The indexes of transactions that read data object x are read marks of node x, which are marked on top of the node. The indexes of transactions that write data object x are write marks of node x, which are marked under the node. Furthermore, if a transaction T_j includes operations $r_j[x]$ and $w_j[y]$, and there exist some transactions T_w including operation $w_w[x]$ and some transactions T_v including operation $r_v[y]$ or $w_v[y]$, an directed edge may be drawn from the node x to the node y (see Figure 1.a and 1.b). If a transaction T_j includes operations $r_j[x]$ and $r_j[y]$, and there exist some transactions T_w including operation $w_w[x]$ and some transactions T_v including operation $w_v[y]$, an directed edge may also be drawn from the node x to the node y (see Figure 1.c).

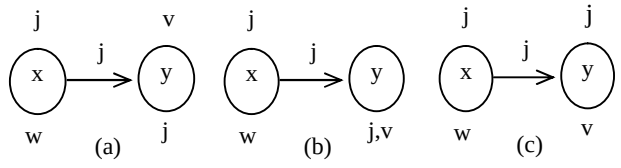

Figure 1. Basic notations of DRG

2.3 Combined locking

We can found that all the induce-to-restart relationships existing in a transaction set can be expressed in its DRG. To avoid transaction restarts, we now propose combined locking to break these relationships. That is to say, we must prevent an active transaction T_j anti-write-depends, read-depends or write depends on another transaction T_v, when T_j has anti-read-depended on other transactions.

Therefore, if a transaction T_j includes operations $r_j[x]$ and $w_j[y]$, and there exist some higher-priority transactions T_w including operation $w_w[x]$ and some higher-priority transactions T_v including operation $r_v[y]$ or $w_v[y]$, then T_j must read-lock x and at the same time write-lock y, which can be represented as $u\_lock_j[x, y]$. If a transaction T_j includes operations $r_j[x]$ and $r_j[y]$, and there exist some transactions T_w including operation $w_w[x]$ and some transactions T_v including operation $w_v[y]$, then T_j must read-lock x and at the same time read-lock y, which can be represented as $r\_lock_j[x, y]$. Furthermore, when T_w write-locks x after $u\_lock_j[x, y]$, read and write locks on y will be denied so as to avoid restarting T_j. When T_w write-locks x after $r\_lock_j[x, y]$, write locks on y will be denied so as to avoid restarting T_j. The lock compatibility of combined locking is shown in Figure 2. The $u\_lock_j[x, y]$ and $r\_lock_j[x, y]$ are called as combined locks, $w\_lock_w[x]$ is called as trigger lock, $r\_lock_v[y]$ or $w\_lock_v[y]$ is called as denied locks. The trigger locks may be a part of denied locks when they belong to the same transaction, as shown in the first row of Figure 2.

Combined Locks / Trigger Locks	u_lock[x, y]	r_lock[x, y]
---	u_lock[y, x] and w_lock[x, y]	w_lock[x, y] (x≠y)
w_lock[x]	r_lock[y] and w_lock[y]	w_lock[y]

Figure 2. The lock compatibility of combined locking

The combined locking can be applied with two-phase technique to control conflicting data accesses in read phase. Then transaction restarts will be impossible to occur in the validation phase, where the validation test is similar to that of OCC-DATI.

2.4 Priority ceiling management

The lock compatibility of combined locking shows the necessary condition to avoid a transaction locking data objects for both the requirements of data consistency and of no transaction restarts. However, it is not yet sufficient to maintain the single-blocking and deadlock-free properties, which are important for schedulability analysis of hard real-time transactions. Therefore, it is necessary to

utilize the priority ceiling mechanism for achieving these two important properties. In most priority ceiling protocols, two types of priority ceilings are defined for each data object for the semantics of transaction's read/write operations. The purpose of priority ceilings is to block the operations of lower priority transactions that may conflict and block those of higher priority transactions in advance, so that a higher priority transaction will only be blocked by a single lower priority transaction. Since combined locking approach is not for single data object, it is for the induce-to-restart relationships. Therefore, one priority ceiling, called trigger priority ceiling, which is the priority of the highest priority transaction that may be blocked because a lower priority obtain a combined lock on [x, y], is needed to control potential transaction restarts. The trigger priority ceiling will come into effect only when another transaction obtains the trigger lock on x.

3. Discussion and future work

In this paper, combined locking approach is proposed for resolving potential serious conflicts that cannot be resolved through dynamic adjustment of serializability order using timestamp intervals. This approach can avoid both transaction restarts existing in optimistic concurrency control protocols, and unnecessary transaction blockings existing in lock-based protocols. Through integrating the combined locking with the priority ceiling mechanism, a new concurrency control protocol called Combined Locking with Dynamic Adjustment of serialization order using Timestamp Intervals (CL-DATI) has been proposed to achieve single-blocking and deadlock-free properties. The schedulability condition for hard real-time transactions under CL-DATI is better than that under other priority ceiling protocols because CL-DATI can avoid unnecessary conflicts blocking and ceiling blocking.

The combined locking approach is mainly for scheduling hard real-time transactions. In fact, the formal analysis method proposed in this paper can also be used in conflict check algorithm to schedule firm/soft real-time transactions, as does in serialization graph test algorithm.

Moreover, different types of real-time transactions may co-exist in a real-time database system. The performance objective is usually to minimize the number of deadline missing of soft/firm real-time transactions, and at the same time, to guarantee the deadline satisfaction of hard real-time transactions. It can be observed that the combined locking approach can be integrated with optimistic concurrency control protocols easily for scheduling mixed transactions in real-time databases.

4. References

[1] L. Sha, R. Rajkumar, S.H. Son and C.H. Chang. A Real-Time Locking Protocol. IEEE Transactions on Computers, 40(7): 793-800, 1991.

[2] L. Sha, R. Rajkumar and J.P. Lehoczky. Priority Inheritance Protocol: An Approach to Real-Time Synchronization. IEEE Transactions on Computers, 39(9): 1175-1185, September 1990.

[3] K. Ramamritham. Real-Time Databases. International Journal of Distributed and Parallel Databases, 1(2), 1993.

[4] J.R. Haritsa, M.J. Carey, and M. Livny. On Being Optimistic about Real-Time Constraints. In Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp.331-343, 1990.

[5] J.R. Haritsa, M.J. Carey, and M. Livny. Data Access Scheduling in Firm Real-Time Database Systems. Journal of Real-Time Systems, 4(3), 203-242, 1992.

[6] A. Chiu, B. Kao, and K. Lam. An Analysis of Lock-Based and Optimistic Concurrency Control Protocols in Multiprocessor Databases. Journal of Systems and Software, 42(3): 273-286, 1998.

[7] J. Lindström and K. Raatikainen. 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, pp.13-20, 1999.

[8] J. Lindström. Optimistic Concurrency Control Methods for Real-Time Database Systems. Report C-2001-9, Department of Computer Science, University of Helsinki, Finland, 2001.

[9] K.-W. Lam, S.H. Son, S.-L. Hung, and Z. Wang. Scheduling Transactions with Stringent Real-Time Constraints. Information Systems, Vol.25 No.6, pp.431-452, 2000.

[10] P.A. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, Massachusetts, 1987.