Using Importance of Transactions and Optimistic Concurrency Control in Firm Real-Time Databases

Jan Lindström [*]
University of Helsinki, Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki,Finland
jan.lindstrom@cs.Helsinki.FI

Kimmo Raatikainen
kimmo.raatikainen@cs.Helsinki.FI

Abstract

In a real-time database system, it is difficult to meet all timing constraints due to the consistency requirements of the underlying database. However, when the transactions in the system are heterogeneous, they are not of the same importance - some are of greater importance than others. In this paper, we propose a new optimistic concurrency control protocol called OCC-PDATI which uses information about the importance of the transactions in the conflict resolution. Performance studies of our protocol have been carried out in the prototype real-time database system. The results clearly indicate that OCC-PDATI meets the goal of favoring transactions of high importance.

1 Introduction

A *real-time database system* (RTDBS) is a database system that must process transactions 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. Considerable research has been devoted to designing concurrency control algorithms for RTDBSs 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: 2PL [4, 6, 12, 13, 16, 21] or *optimistic concurrency control* (OCC) [9, 7, 3, 2, 13, 10, 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 [20].

[*]This work is partially funded by Nokia Telecommunications, Solid Information Technology Ltd., and the National Technology Agency Finland.

Optimistic concurrency control [9, 5] protocols have the properties of non-blocking and deadlock-free which make them especially attractive for RTDBS. As conflict resolution between the transactions is delayed until a transaction is near completion, there will be more information available for making the choice in resolving the conflict. However, the problem with these real-time optimistic concurrency control protocols is the late conflict detection, which makes the restart overhead heavy as some near-to-complete transactions have to be restarted. Since transactions in a real-time database are time-constrained, it is essential that any concurrency control algorithm must minimize the waste of resources [21]. In this paper we concentrate on firm deadlines for real-time transactions. We propose a new optimistic concurrency control protocol called OCC-PDATI, which uses information about the importance of the transactions in the conflict resolution.

The rest of the paper is organized as follows. In Section 2 we discuss recent related work and introduce some notation for the rest of the paper. In Section 3 we introduce the basic mechanisms of the proposed optimistic concurrency control. In Section 4 we describe how the importance of the transaction is taken into account in the OCC-PDATI algorithm. Results from experiments are reported in Section 5.

2 Using Priorities in Concurrency Control

In real-time database systems, the conflict resolution should take into account the importance of the transactions. This is especially true in the case of heterogeneous transactions. Some transactions are more important or valuable than others. Therefore, the goal of any real-time system should be to maximize the value or importance of the completed transactions. In most of the previous approaches the value or importance of a transaction has been equalized to the scheduling priority of a transaction. Unfortunately, that is a very serious restriction if the target is to maximize the value or importance of the completed transactions.

Below we characterize the four typified problems:

- **wasted restart:** A wasted restart occurs if a higher priority transaction aborts a lower priority transaction and later the higher priority transaction is discarded when it misses its deadline.
- **wasted wait:** A wasted wait occurs if a lower priority transaction waits for the commit of higher priority transaction and later the higher priority transaction is discarded when it misses its deadline.
- **wasted execution:** A wasted execution occurs when a lower priority transaction in the validation phase is restarted due to a conflicting higher priority transaction which has not finished yet.
- **unnecessary restart:** An unnecessary restart occurs when a transaction in the validation phase is restarted even when the history would be serializable.

Traditional two-phase locking suffers from the problem of wasted restart and wasted wait. Optimistic protocols suffer the problems of wasted execution and unnecessary restart.

2.1 Real-Time Transactions

A real-time transaction object includes the attributes *priority, deadline*, and *importance*. Priority and deadline attributes are normal object attributes. Therefore the values of the priority and the deadline attributes can be different in every instance of the transaction class. Additionally deadline and priority attributes can be the same for several transaction class instances. The priority attribute can even change in time when a transaction is executed. However, when the transactions in the system are heterogeneous, they are not of the same importance - some are of greater importance than others. Importance attribute is a class attribute, therefore it is the same for all instances of the same transaction class. The importance of the transaction does not depend on the arrival time of the transaction as the deadline attribute does.

A characteristic of most previous real-time concurrency control algorithms is the use of priority based conflict resolution. Here transactions are assigned 'priorities', which are implicit or explicit functions of their deadlines or *criticalness* or both. The criticalness of a transaction is an indication of its level of importance. However, in actuality, these two requirements sometimes conflict with each other. That is, transactions with very short deadlines might not be very critical, and vice versa [1].

Therefore we use importance (or criticalness) of the transactions in place of the priority in the conflict resolution of optimistic concurrency control. This avoids the dilemma of priority based conflict resolution, yet integrates criticalness and deadline such that, not only do the more critical transactions meet their deadlines, but the overall goal is to maximize the net worth of the executed transactions to the system. An importance attribute is used in the real-time scheduler and the concurrency controller in the new proposed algorithm. These extensions were not used in our earlier OCC-DATI [15] algorithm.

3 OCC-PDATI

We have developed an optimistic concurrency control protocol called OCC-PDATI (Optimistic Concurrency Control using Importance of the Transaction and Dynamic Adjustment of Serialization Order). OCC-PDATI is based on forward validation [5] and the earlier optimistic OCC-DATI [15] protocol. The difference is in the conflict resolution. The conflict resolution of OCC-PDATI uses the importance of the transaction found from transaction object attributes. This section outlines new parts of OCC-PDATI when compared to OCC-DATI. Suppose we have a validating transaction T_v and a set of active transactions $T_j (j = 1, 2, ..., n)$. There are three possible types of data conflicts which can cause 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. When $T_v \rightarrow T_j$, then the reads in T_v cannot be affected by writes in T_j. This type of serialization adjustment is called *forward ordering* or *forward adjustment*.
   Transactions of high importance should not be restarted because of data conflict with transaction of low importance. We should offer greater chances for a

transactions of high importance to complete before its deadline. Therefore, if a dynamic adjustment of the serialization order would cause transactions of high importance to be restarted, we restart transactions of lower importance. This is a wasted execution, but it is required to ensure execution of transactions of high importance.

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 writes of T_v. This type of serialization adjustment is called *backward ordering* or *backward adjustment*.
Again, we must ensure serializable (or another correct order of) execution. We do not know what an active transaction is going to do in the future. These future reads or writes may lead to an empty timestamp interval if we make backward adjustments. Therefore transactions of high importance should not be backward adjusted, but conflicting transactions having lower importance should be restarted. This is wasted execution and unnecessary restart, which must be acceptable when we favor transactions of high importance. We could make backward adjustment of a transaction of high importance if the transaction is not to be restarted due to an empty timestamp. This, however, implies that a transaction of high importance should not be allowed to read or write new data objects that belong to a new database state after a backward adjustment. In other words, future read or write operations could reduce the timestamp interval of the transaction.

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. When $T_v \rightarrow T_j$, then the writes of T_v cannot overwrite the writes of T_j'.
This case is the same as in the read-write conflict.

Figure 1 depicts an implementation outline of a deferred dynamic adjustment of the serialization order using timestamp intervals and information about the importance of the transactions.

We use a conflict resolution table mechanism * to implement an optimistic protocol as proposed in [7]. In the selected mechanism the system maintains a system-wide conflict resolution table to take care of book keeping data access by all concurrently executing transactions. These entries are not traditional locks, instead entries are only for book keeping. Generally, the validation process is carried out by checking the entry compatibility with the conflict resolution table. Such entry-based implementation of the validation test is efficient because its complexity does not depend on the number of active transactions. There are two possible implementations of the write phase: *serial validation-write* (OCCL-SVW) and *parallel validation-write* (OCCL-PVW) [7]. We selected the parallel validation-write because our implementation is based on a multiprocess server.

*This data structure is similar to locking mechanisms. In this data structure there is no waiting for access grants.

```
forward_adjustment($T_a, T_v, adjusted$)
{
  if ($T_a \in adjusted$)
    $TI = adjusted.pop(T_a)$;
  else
    $TI = TI(T_a)$;

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

   if ( $imp(T_v) < imp(T_a)$ )
     if ( $TI == \emptyset$ )
       $restart(T_v)$; /* Validation ends here /*

  $adjusted.push(\{(T_a, TI)\})$;
}

backward_adjustment($T_a, T_v, adjusted$)
{
  if ($T_a \in adjusted$)
    $TI = adjusted.pop(T_a)$;
  else
    $TI = TI(T_a)$;

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

   if ( $imp(T_v) < imp(T_a)$ )
       $restart(T_v)$; /* Validation ends here /*

  $adjusted.push(\{(T_a, TI)\})$;
}
```

Figure 1. Backward and Forward adjustment for OCC-PDATI.

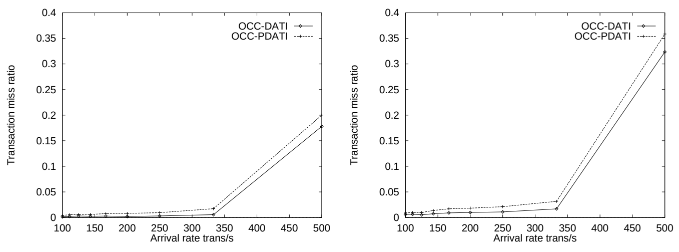
4 Results from Experiments

We have carried out a set of experiments in order to examine the feasibility of the OCC-PDATI algorithm in practice. The prototype system used is based on the *Real-Time Object-Oriented Database Architecture for Intelligent Networks* (RODAIN) specification [14, 8, 17], which is an architecture for a real-time, object-oriented, and fault-tolerant database management system. The RODAIN prototype system is a main-memory database, which uses priority and importance based scheduling and optimistic concurrency control. All experiments were executed in the RODAIN prototype database running on Pentium Pro 200MHz and 64 MB of main memory with the Chorus/ClassiX operating system [18].

In the test environment, transactions arrive to a specific user interface subsystem (URIS) that receives the arriving transactions from an off-line generated test file. Every test session contains 10 000 transactions and is repeated at least 20 times. The reported values are the means of the repetitions. In the experiments, we examined how well our OCC-PDATI algorithm performs when compared to the OCC-DATI algorithm [15].

The test database represents a typical Intelligent Network (IN) service. The size of the database is 30 000 objects. We used four different transactions R1, R2, W1 and W2. Transactions R1 and R2 are a read-only service provision transaction. The transaction R1 reads one user profile. The transaction R2 represents abbreviated dialing or call forwarding services [19]. Transactions W1 and W2 are update service provision transactions. The transaction W1 implements management service of the customer in IN CS-1. The transaction W2 implements updates to abbreviated dialing or call forwarding services in IN CS-1. All transactions are firm real-time transactions.

In the first set of experiments we used a fixed fraction of write transactions. The arrival rate of transactions was the varying parameter. As expected, there is some overhead when information about the importance of the transaction is used in dynamic adjustment of serialization order. As Figure 2 indicates the overhead using additional information from the transactions is quite low. The miss-ratio of the transactions when using the OCC-PDATI algorithm is only slightly higher than in the OCC-DATI.

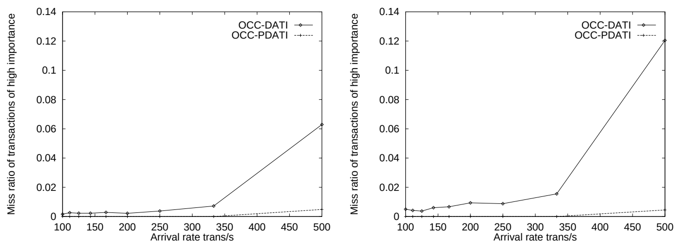


(a) 20% writes

(b) 40% writes

Figure 2. Comparison with varying transaction arrival rate.

Figure 3 shows the miss-ratio of transactions of high importance. Figure 3 demonstrates how the OCC-PDATI favors transactions of high importance. OCC-PDATI clearly offers better chances for high priority transactions to complete according to their deadlines. The results clearly indicate that OCC-PDATI meets the goal of favoring transactions of high importance.

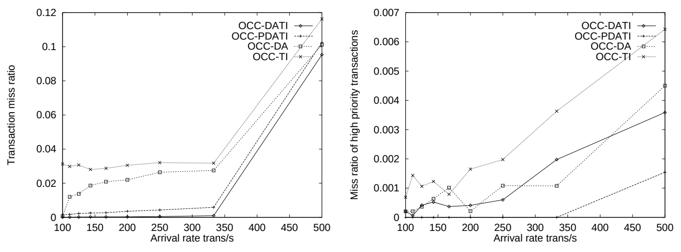


(a) 20% writes

(b) 40% writes

Figure 3. Miss ratio of transaction of high importance.

Finally, we have compared performance of the OCC-PDATI to OCC-DATI, OCC-DA, and OCC-TI. Figure 4(a) demonstrates that the performance of the OCC-PDATI is better than OCC-TI and similar to OCC-DA. Similarly Figure 4(b) demonstrates how the OCC-PDATI favors transactions of high importance. The miss-ratio of transactions of high importance is clearly lower in OCC-PDATI when compared with OCC-DATI, OCC-DA, and OCC-TI.



(a) 30% writes

(b) 30% writes

Figure 4. OCC-PDATI compared with OCC-DATI, OCC-DA, and OCC-TI.

5 Conclusions

Although the optimistic approach has been shown to have a better performance than locking protocols in firm real-time database systems, it has problems of unnecessary restarts and a high restart overhead. In this paper, we have proposed an optimistic concurrency control protocol called OCC-PDATI that takes into account the importance of transactions. It has several advantages over the other concurrency control protocols. The protocol maintains all the nice properties of forward validation, a high degree of concurrency, freedom from deadlock, and early detection and resolution of conflicts, resulting in less waste of resources as well as a smaller number of restarts. All of these are important to the performance of RTDBSs and contribute to greater chances of meeting transaction deadlines.

An efficient method was designed to adjust the serialization order dynamically amongst the conflicting transactions in order to reduce the number of restarted transactions. The method also incorporates the importance of the transaction in conflict resolution. When compared with the OCC-DATI protocol that uses dynamic serialization order adjustment, the OCC-PDATI protocol offers the same efficiency and the overhead is only slightly larger. The most important feature of the OCC-PDATI is that it clearly offers better chances for the transactions of high importance to complete before their deadlines when compared to the OCC-DATI. The results clearly indicate that OCC-PDATI meets the goal of favoring transactions of high importance.

Acknowledgments

We would like to thank Tiina Niklander, Lea Kutvonen, Jaakko Kurhila and Matti Luukkainen from the University of Helsinki for fruitful discussion during this research.

References

[1] S. R. Biyabani, J. A. Stankovic, and K. Ramamritham. The integration of deadline and criticalness in hard real-time scheduling. In *Proc. of the Real-Time System Symposium*, 1988.

[2] A. Datta and S. H. Son. A study of concurrency control in real-time active database systems. Tech. report, Department of MIS, University of Arizona, Tucson, 1996.

[3] A. Datta, I. R. Viguier, S. H Son, and V. Kumar. A study of priority cognizance in conflict resolution for firm real time database systems. In *Proc. of the Second International Workshop on Real-Time Databases: Issues and Applications*, March 7-8, 1997.

[4] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19(11):624–633, November 1976.

[5] T. Härder. Observations on optimistic concurrency control schemes. *Information Systems*, 9(2):111–120, 1984.

[6] J. R. Haritsa, M. J. Carey, and M. Livny. Dynamic real-time optimistic concurrency control. In *Proc. of the 11th Real-Time Symposium*, pages 94–103, 1990.

[7] J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley. Experimental evaluation of real-time optimistic concurrency control schemes. In G. M. Lohman, A. Sernadas, and R. Camps, editors, *Proc. of the 17th VLDB Conference*, pages 35–46, September 1991.

[8] J. Kiviniemi, T. Niklander, P. Porkka, and K. Raatikainen. Transaction processing in the RODAIN real-time database system. In A. Bestavros and V. Fay-Wolfe, editors, *Real-Time Database and Information Systems*, pages 355–375, 1997.

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

[10] K.-W. Lam, K.-Y. Lam, and S. Hung. An efficient real-time optimistic concurrency control protocol. In *Proc. of the First International Workshop on Active and Real-Time Database Systems*, pages 209–225, June 1995.

[11] K.-W. Lam, K.-Y. Lam, and S. Hung. Real-time optimistic concurrency control protocol with dynamic adjustment of serialization order. In *Proc. of IEEE Real-Time Technology and Application Symposium*, pages 174–179, May 1995.

[12] J. Lee and S. H. Son. Using dynamic adjustment of serialization order for real-time database systems. In *Proc. of the 14th IEEE Real-Time Systems Symposium*, pages 66–75, 1993.

[13] J. Lee and S. H. Son. Performance of concurrency control algorithms for real-time database systems. In V. Kumar, editor, *Performance of Concurrency Control Mechanisms in Centralized Database Systems*, pages 429–460. 1996.

[14] J. Lindström, T. Niklander, P. Porkka, and K. Raatikainen. A distributed real-time main-memory database for telecommunication. In *Databases in Telecommunications*, Lecture Notes in Computer Science, 1819, pages 158–173, 1999.

[15] J. Lindström and K. Raatikainen. Dynamic adjustment of serialization order using timestamp intervals in real-time databases. In *Proc. of 6th International Conference on Real-Time Computing Systems and Applications*, 1999.

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

[17] T. Niklander, J. Kiviniemi, and K. Raatikainen. A real-time database for future telecommunication services. In D. Ga¨ti, editor, *Intelligent Networks and Intelligence in Networks*, pages 413–430, 1997. Chapman & Hall.

[18] Dick Pountain. The Chorus microkernel. *Byte*, pages 131–138, January 1994.

[19] K. E. E. Raatikainen. Information aspects of services and service features in intelligent network capability set 1. Report C-1994-45, University of Helsinki, Dept. of Computer Science, Helsinki, Finland, September 1994.

[20] K. Ramamritham. Real-time databases. *Distributed and Parallel Databases*, 1:199–226, 1993.

[21] P. S. Yu, K.-L. Wu, K.-J. Lin, and S. H. Son. On real-time databases: Concurrency control and scheduling. *Proc. of the IEEE*, 82(1):140–157, January 1994.