Dynamic Adjustment of Serialization Order using Timestamp Interval in Real-Time Databases

Jan Lindström (jplindst@cs.helsinki.fi)

Helsinki January 25, 1999

University of Helsinki
Department of Computer Science
Seminar on Transaction Processing

Contents

1 Introduction

Real-Time Database Systems (RTDBSs) differ from conventional database systems in a number of ways. In RTDBS transactions have timing constraints and the primary performance criterion is timeliness level not average response time or throughput. Scheduling of transactions is driven by priority considerations rather than fairness considerations. Given these significant differences, considerable research has been recently devoted to designing concurrency control algorithms for RTDBSs and to evaluating their performance [AGM88, HCL90b, HSTR89, HSRT91, LLH95a, LLH95b, LSH97, LS96, KLR97, MN82]. Most of these algorithms are based on one of the two basic concurrency control mechanisms: *locking* [EGLT76] or *optimistic concurrency control* (OCC) [KR81]. Some of them use priority information in the resolution of conflicts, that is, conflicts are resolved in favor of the higher priority transactions.

Optimistic concurrency control protocols have the nice properties of being non-blocking and deadlock-free. Those properties 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 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. Thus, the major concern in the design of real-time optimistic concurrency control protocols is not only to incorporate priority information for conflict resolution but also to design methods to minimize the number of transaction restarts.

In this paper, we present a method to reduce the number of transaction restarts and we propose a new optimistic concurrency control protocol, called OCC-DATI. OCC-DATI is fully optimistic protocol and uses forward adjustment. With the new protocol, the number of transaction restarts is smaller than with OCC-BC, OPT-WAIT, or WAIT-50 [HCL90b, HCL90a, HSRT91], because the serialization order of the conflicting transactions is adjusted dynamically. On the other hand, the overhead for supporting dynamic adjustment is much smaller than the one in OCC-DA [LLH95a, LLH95b] or modified OCC-TI [KLR97].

Remaining parts of the paper is organized as follows. Section 2 introduces the basic mechanisms of our new optimistic CC protocol. In Section 3 contains the performance study of the new protocol as compared with other optimistic methods. Finally, the conclusion of the paper is presented in

Section 4.

2 A New Real-Time OCC Protocol

OCC-DATI is based on forward validation [Här84]. The number of transaction restarts is reduced by dynamic adjustment of serialization order which is supported with timestamp intervals. Unlike the OCC-TI protocol [LS96, KLR97], all checking is performed at the validation phase of each transaction. There is no need to check for conflicts while a transaction is still in its read phase. In OCC-DATI, when a transaction enters the validation phase, it's serialization order relative to its conflicting transactions will be adjusted either backward or forward based on the conflict type. A *serious conflict* occurs when a conflicting active transaction has to be both backward and forward adjusted with respect to the validating transaction. The validating transaction is allowed to commit if the validity of all its accessed data are still sound and there is no serious conflict.

2.1 Dynamic Adjustment of Serialization Order

The major performance problem with OCC protocols is the late restart. Thus, one important way to improve the performance of OCC protocols is to reduce the number of transaction restarts. Actually, in the conventional OCC protocols many transaction restarts are not necessary. For example, let $r_i[x]$ and $w_i[x]$ denote read and write operation, respectively, on data object x by transaction T_i, and let v_i and c_i denote the validation and commit of transaction T_i. Consider transactions T_1 and T_2:

$T_1 : r_1[x]w_1[x]v_1c_1$
$T_2 : r_2[x]w_2[y]v_2c_2$ and suppose they execute as follows:
$H_1 = r_2[x]w_2[y]r_1[x]w_1[x]v_1$

Based on the OCC-FV [Här84], T_2 has to be restarted. However, it is not necessary. Because if T_2 is allowed to commit such as:

$H_2 = r_2[x]w_2[y]r_1[x]w_1[x]v_1c_1\mathbf{v_2c_2},$

the schedule of H_2 is equivalent to the serialization order $T_2 \rightarrow T_1$ as the actual write of T_1 is performed after its validation and after the read of T_2. There is no cycle in their serialization graph and H_2 is serializable [BHG87].

One way to reduce the number of transaction restarts is to dynamically adjust the serialization order of the conflicting transactions. When some data conflict with the validating transaction is detected, there is no need to restart the conflicting transaction immediately. Instead, a serialization order can be dynamically defined. To preserve serializability with OCC protocols, if validating transaction T_v has to be serialized before active transaction T_j, the following two conditions must be satisfied:

1) No overwriting. The writes of T_v should not overwrite the writes of T_j.
2) No read dependency. The writes of T_v should not affect the read phase of T_j.

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 induce serialization order between T_v and T_j:

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

If an active transaction has to be both backward and forward adjusted with respect to the validating transaction, serious conflicts is said to be occurred and some of the conflicting transactions 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.

2.2 Procedural Description

The OCC-DATI protocol resolves conflicts using time intervals [LS93] of the transactions. Every transaction must be executed within a specific time slot. When an access conflict occurs, it is resolved using the read and write sets of the conflicting transactions together with the allocated time slot. Time slots are adjusted when a transaction commits. In this protocol, every transaction in the read phase is assigned a timestamp interval (TI). This interval is used to record a temporary serialization order induced during the execution of the transaction. At the start of the execution, the timestamp interval of the transaction is initialized as $[0, \infty[$, i.e., the entire range of timestamp space. Whenever the serialization order of the transaction is induced by its data operation or the validation of other transactions, its timestamp interval is adjusted to represent the dependencies.

At the beginning of the validation (Figure 1), the final timestamp of the validating transaction $TS(T_v)$ is determined from the timestamp interval allocated to the transaction T_v. The timestamp intervals of all other concurrently running and conflicting transactions must be adjusted to reflect the serialization order. We set $TS(T_v)$ to the validation time if it belongs to the time interval of T_v or to maximum value from the time interval otherwise. $RTS(D_i)$ and $WTS(D_i)$ denote the largest timestamp of commited transactions that have read and written, respectively, data object D_i. The adjustment of timestamp intervals of a active transactions iterates through the readset (RS) and writeset (WS) of validating transaction. When access has been made to the same objects both in validating transaction and in the active transaction, the time interval of the active transaction is adjusted. Non-serializable execution is detected when the timestamp interval of an active transaction shuts out and transaction is restarted.

```
occdati_validate(T_v)
{
    if (validation_time ∈ TI(T_v))
        TS(T_v) = validation_time;
    else
        TS(T_v) = max(TI(T_v));

    for ( ∀ D_i ∈ (RS(T_v) ∪ WS(T_v) ))
    {

        if (D_i ∈ RS(T_v))
            TI(T_v) = TI(T_v) ∩ [WTS(D_i), ∞[ ;

        if (D_i ∈ WS(T_v))
            TI(T_v) = TI(T_v) ∩ [WTS(D_i), ∞[ ∩ [RTS(D_i), ∞[ ;

        if (TI(T_v) = []) restart(T_v);

        for ( ∀ T_a ∈ active_conflicting_transactions() )
        {
            if (D_i ∈ (WS(T_a) ∩ RS(T_v)))
                TI(T_a) = TI(T_a) ∩ [TS(T_v), ∞[ ;

            if (D_i ∈ (RS(T_a) ∩ WS(T_v)))
                TI(T_a) = TI(T_a) ∩ [0, TS(T_v) - 1];

            if (D_i ∈ (WS(T_a) ∩ WS(T_v)))
                TI(T_a) = TI(T_a) ∩ [TS(T_v), ∞[ ;

            if (TI(T_a) = []) restart(T_a);
        }

        if (D_i ∈ RS(T_v)) RTS(D_i) = max(RTS(D_i), TS(T_v));
        if (D_i ∈ WS(T_v)) WTS(D_i) = max(WTS(D_i), TS(T_v));
    }

    commit WS(T_v) to database;
}
```

Figure 1: Validation algorithm for OCC-DATI.

Using the example below we show how our algorithm works. Consider transactions T_1, T_2 and T_3 and history H_1:

$T_1 : r_1[x]w_1[x]r_1[y]w_1[y]v_1$
$T_2 : r_2[x]w_2[x]...v_2$
$T_3 : r_3[y]...v_3$
$H_1 : r_1[x]w_1[x]r_2[x]r_3[y]w_2[x]r_1[y]w_1[y]v_1$

In this example T_1 reaches validation phase first and has *reconcilable conflict* with T_3 and a *irreconcilable conflict* with T_2 [LS93]. Therefore, it is required that transaction T_2 be restarted. Similary, T_3 must precede T_1 in the serialization history in order to avoid an unnecessary restart. Let $RTS(x)$, $WTS(x)$, $RTS(y)$, and $WTS(y)$ be initialized as 100. Assume that transaction T_1 arrives to its validation phase at time 1000. Our OCC-DATI algorithm sets $TS(T_1)$ as 1000. The validating transaction T_1 is first forward adjusted to $[100, \infty[$. Transaction T_2 has written object x and T_2's time interval is adjusted to $[1000, \infty[$ using forward adjustment, and T_2 has also read object x thus time interfal is adjusted to $[1000, \infty[\ \cap \ [0, 999] = []$ using backward adjustment. Thus the time interval of T_2 is shut out and the transaction is restarted. Transaction T_3 has read object y and it's time interval is adjusted to $[0, 999]$ using backward adjustment. Transaction T_1 updates object timestamps and commits. Thus our OCC-DATI algorithm produces a serializable history as well as avoids the unnecessary restart problem $\Box$.

Lemma 1: Let T_1 and T_2 be committed transactions in a history H produced by the proposed algorithm. If there is an edge $T_1 \rightarrow T_2$ in $SG(H)$, then $TS(T_1) < TS(T_2)$.

Proof: Since there is an edge, $T_1 \rightarrow T_2$ in $SG(H)$, there must exist one or more conflicting operations whose type is one of the following three:

1) $r_1[x] \rightarrow w_2[x]$: This case implies that T_1 commits before T_2 reaches its validation phase since $r_1[x]$ is not affected by $w_2[x]$. For $w_2[x]$, OCC-DATI adjusts $TI(T_2)$ to follow $RTS(x)$ that is equal to or greater than $TS(T_1)$. That is, $TS(T_1) \leq RTS(x) < TS(T_2)$. Therefore, $TS(T_1) < TS(T_2)$.
2) $w_1[x] \rightarrow r_2[x]$: This case implies that the write phase of T_1 finishes before $r_2[x]$ executes in T_2's read phase. For $r_2[x]$, OCC-DATI adjusts $TI(T_2)$ to follow $WTS(x)$, which is equal to or greater than $TS(T_1)$. That is, $TS(T_1) \leq WTS(x) < TS(T_2)$. Therefore, $TS(T_1) < TS(T_2)$.

3) $w_1[x] \rightarrow w_2[x]$: This case implies that write phase of T_1 finishes before $w_2[x]$ executes in T_2's write phase. For $w_2[x]$, OCC-DATI adjusts $TI(T_2)$ to follow $WTS(x)$, which is equal to or greater than $TS(T_1)$. That is, $TS(T_1) \leq WTS(x) < TS(T_2)$. Therefore, $TS(T_1) < TS(T_2)$. $\Box$

Theorem 1: Every history generated by OCC-DATI algorithm is serializable.

Proof: Let H denote any history generated by the OCC-DATI algorithm. Suppose, by way of contradiction, that $SG(H)$ contains a cycle $T_1 \rightarrow T_2 \rightarrow ... \rightarrow T_n \rightarrow T_1$, where $n > 1$. By Lemma 1, we have $TS(T_1) < TS(T_2) < ... < TS(T_n) < TS(T_1)$. This is a contradiction. Therefore no cycle can exist in $SG(H)$ and thus the algorithm produces only serializable histories. $\Box$

2.3 Implementation issues

We use a locking mechanism to implement an optimistic protocol as proposed in [HSRT91]. In the selected mechanism the system maintains a system-wide lock table to take care of book keeping data access by all concurrently executing transactions. Generally, the validation process is carried out by checking the lock compatibility with the lock table. Such locking-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) [HSRT91]. We selected the parallel validation-write because our implementation is based on a multiprocess server.

3 Experimental Results

We have carried out a set of experiments in order to examine feasibility of our algorithms in practise. *Real-Time Object-Oriented Database Architecture for Intelligent Networks* (RODAIN) [TR96, NKR97, KNPR97, Raa97] is an architecture for a real-time, object-oriented, fault-tolerant, and distributed database management system. RODAIN consists of main-memory database, priority based scheduling and optimistic concurrency control. All experiments were executed in the RODAIN prototype database running on a Pentium Pro 200Mzh with 64 MB of main memory with the Chorus/ClassiX operating system. The test environment is a reduced subset of the RODAIN architecture.

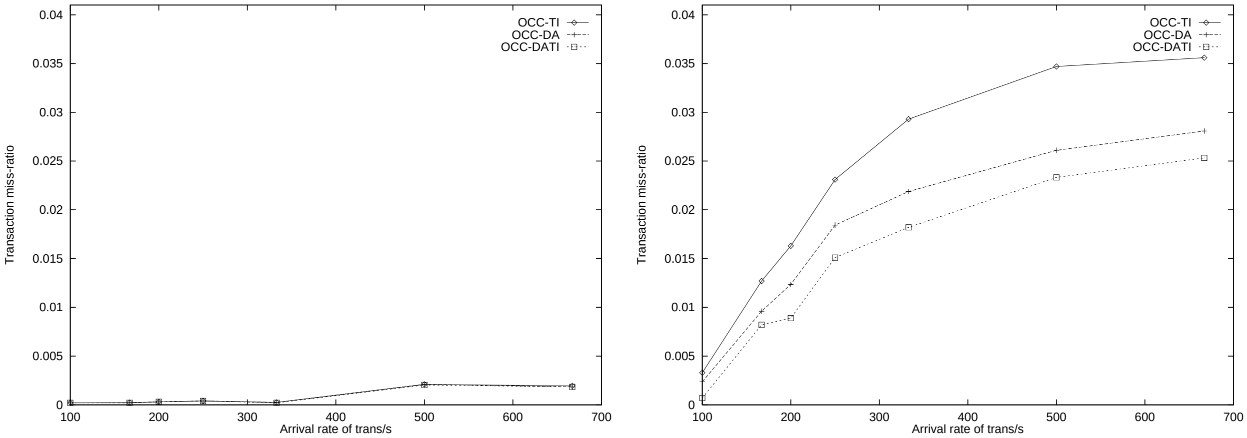
All transactions arrive to the RODAIN prototype through a specific user subsystem. In these tests we have used a special user subsystem that gets the arriving transactions from a off-line generated test file. Every test session contains $10\,000$ transactions and is repeated at least 20 times. Reported values are means of the replications. In particular, we examined how well our OCC-DATI algorithm performs when compared to the OCC-TI [LS93] and OCC-DA [LLH95a] algorithm.

The test database represents a typical Intelligent Network (IN) service, the database schema of which was provided by Sonera. The size of the database is $30\,000$ objects. We used three different transactions named as R1, W1 and T1. Transaction R1 is a read-only service provision transaction that reads few objects and commits. Transaction W1 is an update service provision transaction that reads few objects, updates them and commits. Transaction T1 is a service management transaction that reads part of database, updates 50% of the read objects and commits.

New transactions are accepted up to a prespecified limit, which is the number of installed Transaction Processes. If there is no Transaction Process available when a new transaction arrives, the transaction is aborted. Transactions are validated atomically. If the deadline of a transaction expires, the transaction is always aborted. The workload in a test session consists of variable mix of transactions. Fractions of each transaction type is a test parameter. Other test parameters include the arrival rate assumed to be exponentially distributed. Relative deadline of all firm real-time transactions is 100ms. Final deadline is calculated adding relative deadline to arrival time. Thus deadlines are also exponentially distributed.

3.1 Experimental results

In Figure 2(a) fraction of write transactions is 40%. We can see that there is only small differences between the algorithms. When we used even smaller fractions of write transactions the difference between algorithms was even smaller. In Figure 2(b) fraction of write transactions is 60%. We can clearly see that OCC-DATI offers best overal performance, OCC-DA is second best, and OCC-TI is clearly worst algorithm. This confirms that overhead for supporting dynamic adjustment in OCC-DATI is smaller than one in OCC-DA. This confirms also that number of transaction restarts is smaller in OCC-DATI compared to OCC-TI and OCC-DA. Our experimental results also confirm experimental results in [LLH95a] that OCC-DA outperform OCC-TI. Experimental results in [LS93] have already comfirmed that OCC-TI outperform OCC-BC, OPT-WAIT and WAIT-50 algorithms.

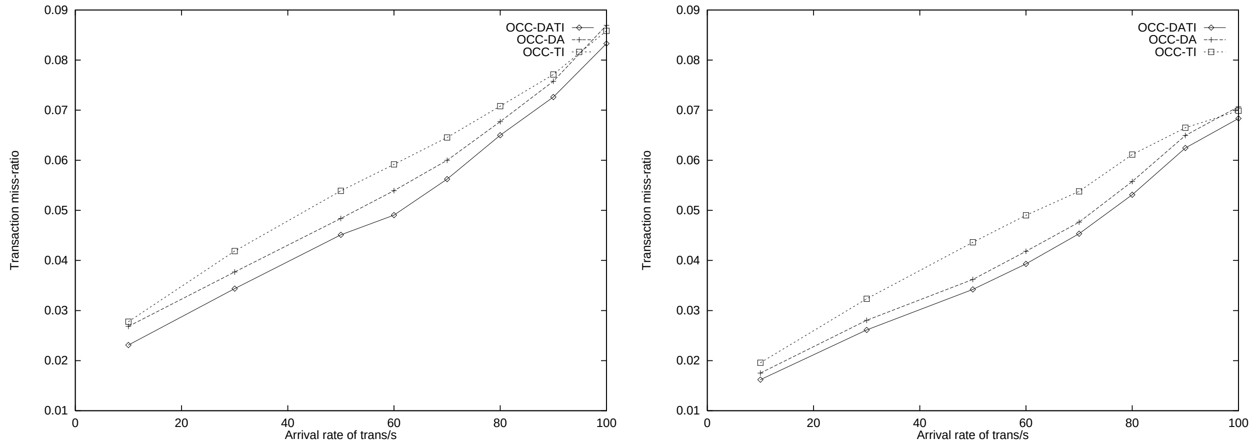


(a) 60% R1 40% W1

(b) 40% R1 60% W1

Figure 2: OCC algorithms compared.

In the second series of experiment we have used fixed arrival rate of transactions and varied the fraction of write transactions (W1) from 10% to 100%. In Figure 3(a) arrival rate of transactions is 200 transactions in second. We can see that OCC-DATI offers best overal performance compared to OCC-DA and OCC-TI. Again OCC-DA is second best and OCC-TI is worst algorithm. In Figure 3(b) arrival rate of transactions is 333 transactions in second. We can see that OCC-DATI offers best overal performance compared to OCC-DA and OCC-TI. Again OCC-DA is second best and OCC-TI is worst algorithm. This confirms that OCC-DATI outperforms OCC-DA and OCC-TI also when arrival rate of transactions is increased.



(a) 200 transactions/s

(b) 333 transactions/s

Figure 3: OCC algorithms compared.

4 Conclusions

Although optimistic approach has been shown to be better than locking protocols for RTDBS, it has the problems of unnecessary restarts and heavy restart overhead. In this paper, we propose a new optimistic concurrency control protocol called OCC-DATI. It has several advantages over the other concurrency control protocols. The protocol maintains all the nice properties with forward validation, high degree of concurrency, free from deadlock, early detection and resolution of conflicts, resulting in both less wasted resources and smaller number of restarts. All of these are important to the performance of RTDBS and contribute to greater chances of meeting transaction deadlines.

An efficient method is designed to adjust the serialization order dynamically amongst the conflicting transactions to reduce the number of transaction restarts. Comparing with other OCC protocols that use dynamic serialization order adjustment, our method OCC-DATI is much more efficient and the overhead is smaller. There is no need to check for conflicts while a transaction is still in its read phase. All the checking is performed in the validation phase. Our performance studies presented here and other performance test results we have made confirm that OCC-DATI outperforms OCC-TI and OCC-DA.

References

[AGM88] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions: A performance evaluation. In F. Bancilhon and D. J. DeWitt, editors, *Proceedings of the 14th VLDB Conference*, pages 1–12, San Mateo, Calif., 1988. Morgan Kaufmann.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[EGLT76] 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.

[Här84] T. Härder. Observations on optimistic concurrency control schemes. *Information Systems*, 9(2):111–120, 1984.

[HCL90a] J. R. Haritsa, M. J. Carey, and M. Livny. Dynamic real-time optimistic concurrency control. In *Proceedings of the 11th Real-Time Symposium*, pages 94–103, Los Alamitos, Calif., 1990. IEEE, IEEE Computer Society Press.

[HCL90b] 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. ACM, ACM Press, 1990.

[HSRT91] 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, *Proceedings of the 17th VLDB Conference*, pages 35–46, San Mateo, Calif., September 1991. Morgan Kaufmann.

[HSTR89] J. Huang, J. A. Stankovic, D. Towsley, and K. Ramamritham. Experimental evaluation of real-time transaction processing. In *Proceedings of the 10th Real-Time Systems Symposium*, pages 144–153, Los Alamitos, Calif., December 1989. IEEE, IEEE Computer Society Press.

[KLR97] P. Konana, J. Lee, and S. Ram. Updating timestamp interval for dynamic adjustment of serialization order in optimistic concurrency control-time interval (occti) protocol. *Information Processing Letters*, 63:189–193, 1997.

[KNPR97] 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, London, 1997. Kluwer Academic Publishers.

[KR81] H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems*, 6(2):213–226, June 1981.

[LLH95a] K. Lam, K. 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, Skövde, Sweden, June 1995. Springer.

[LLH95b] K. Lam, K. Lam, and S. Hung. Real-time optimistic concurrency control protocol with dynamic adjustment of serialization order. In *Proceedings of IEEE Real-Time Technology and Application Symposium*, pages 174–179, Chigago, Illinois, May 1995.

[LS93] 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, Los Alamitos, Calif., 1993. IEEE, IEEE Computer Society Press.

[LS96] Y. Lee and S. 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. Prentice-Hall, 1996.

[LSH97] K. Lam, S. Son, and S. Hung. A priority ceiling protocol with dynamic adjustment of serialization order. In *13th IEEE Conf. on Data Engineering (ICDE'97)*, Birmingham, UK, April 1997.

[MN82] D. Menasce and T. Nakanishi. Optimistic versus pessimistic concurrency control mechanisms in database management systems. *Information Systems*, 7(1):13–27, 1982.

[NKR97] T. Niklander, J. Kiviniemi, and K. Raatikainen. A real-time database for future telecommunication services. In D. Gaïti, editor, *Intelligent Networks and Intelligence in Networks*, pages 413–430, Paris, France, 1997. Chapman & Hall.

[Raa97] K. Raatikainen. Real-time databases in telecommunications. In Azer Bestavros, Kwei-Jay Lin, and Sang Hyuk Son, editors, *Real-Time Database Systems: Issues and Applications*, pages 93–98. Kluwer, 1997.

[TR96] J. Taina and K. Raatikainen. Experimental real-time object-oriented database architecture for intelligent networks. *Engineering Intelligent Systems*, 4(3):57–63, September 1996.