Relaxed Correctness for Firm Real-Time Databases

Jan Lindström
University of Helsinki, Department of Computer Science,
P.O. Box 68, 00014 Helsingin yliopisto, FINLAND
jan.lindstrom@cs.helsinki.fi

Abstract

Real-time database system must meet time constraints in addition to the integrity constraints. Concurrency control is one of the main issues in the studies of real-time database systems. Traditional concurrency control methods use serializability as the correctness criterion when transactions are executed concurrently. However, strict serializability as the correctness criterion is not always suitable in real-time databases. Instead, correctness requirements vary from one type of transactions to another and from one data type to another. In this paper we propose a concurrency control method witch is based on optimistic methods with a extension to relaxed serializability and semantic conflict resolution so that a special purpose concurrency control scheme can be applied. Proposed method is evaluated and tested in prototype implementation of real-time database system for telecommunications.

1. Introduction

Many real-word applications involve time-constrained access to data and access to data that has temporal validity. Consider for example a telephone switching system, network management, navigation systems, stock trading, and command and control systems. Additionally, consider the following operations within these environments: looking up the "800 directory", obstacle detection and avoidance, radar tracking and recognition of objects. These involve gathering data from the environment, processing of information, and contributing *timely* response. Additionally, these examples contain processing both temporal data, which loses its validity after a certain time intervals, as well as historical data.

Traditional databases, hereafter referred to as databases, deal with persistent data. Transactions access this data while maintaining its consistency. The overall goal of transaction and query processing in databases is to get a good throughput or response time. In *real-time systems* can also deal with temporal data, i.e., data that becomes outdated after a certain time. Therefore, tasks in real-time systems have deadlines. The important difference is that the goal of real-time systems is to meet the time constraints of the tasks [27].

Real-time does not just mean fast [27] and timing constraints that are in nanoseconds or microseconds. Instead, real-time means the need to manage *explicit* time constraints in a predictable fashion, that is, to use time-cognizant protocols to deal with deadlines or periodicity constraints associated with tasks [23].

Concurrency control is one of the main issues in the studies of real-time database systems. With a strict consistency requirement defined by serializability [3], most real-time concurrency control schemes considered in the literature are based on two-phase locking (2PL) [6]. 2PL has been studied extensively in traditional database systems and is being widely used in commercial databases. In recent years, various real-time concurrency control protocols have been proposed for single-site RTDBS by modifying 2PL [12, 22, 16].

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 [24].

Optimistic concurrency control protocols [13, 8] have the nice properties of being non-blocking and deadlock-free. These properties make them especially attractive for real-time database systems. Because conflict resolution between the transactions is delayed until a transaction is near to its completion, there will be more information available in making the conflict resolution. Although optimistic approaches have been shown to be better than locking protocols for RTDBSs [10, 9], they have the problem of unnecessary restarts and heavy restart overhead. This is due to the late conflict detection that increases the restart overhead since some near-to-complete transactions have to be restarted. Therefore in recent years numerous optimistic

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

concurrency control algorithms have been proposed for real-time databases [5, 4, 18, 15].

Telecommunication is an example of an application area, which has database requirements that require a real-time database or at least time-cognizant database. A telecommunication database, especially one designed for Intelligent Network (IN) services [1], must support access times less than 50 milliseconds. Most database requests are simple reads, which access few items and return some value based on the content in the database. In this paper two transaction types have been studied: service provision (IN) and service management (TMN) [2] transactions. In transaction scheduling, IN transactions are expressed as firm deadline transactions and TMN transactions are expressed as soft deadline transactions.

The rest of the paper is organized as follows. Section 2 presents recent related work in relaxing serializability and proposed method is presented. Section 3 presents a experimental setup and results. Finally, the conclusion of the paper is presented in Section 4.

2 Relaxing Serializability

Traditional concurrency control methods use serializability [6] as the correctness criterion when transactions are concurrently executed. However, in real-time database systems strict serializability is not always needed because it restricts simultaneous access and induces unnecessary overhead to the system. Real-time data is often temporal and neither serializing concurrency control nor full support for failure recovery is required because of the overhead [28]. This has led to ideas such as *atomic set-wise serializability* [26], *external versus internal consistency* [19], *epsilon serializability* [25], and *similarity* [14]. Graham [7] has argued that none are as obviously correct, nor as obviously implementable, as serializability.

Due to the semantic properties of telecommunications applications, the correctness criterion can be relaxed to so called *semantic based serializability*. We have decomposed the semantic based serializability into two parts. Firstly, we have defined a temporal serializability criterion called *τ-serializability*, which allows old data to be read unless the data is too old. Secondly, we have used a semantic conflict resolution model that introduces explicit rules, which are then used to relax transaction serializability. The first method reduces the number of read-write conflicts whereas the second one reduces the number of write-write conflicts.

We use the τ-serializability as a correctness criterion to reduce read-write conflicts. Suppose that transaction T_A updates the data item x at time t_a. Later transaction T_B wants to read the data item x. Let t_b be the time when T_B requests the read on x. In τ-serializability the two operations do not conflict if $t_a + \min(\tau_b, \tau_x) > t_b$. The tolerance $\min(\tau_b, \tau_x)$ specifies how long the old value is useful, which may depend both on data semantics (τ_x) and on application semantics (τ_b).

We have adopted the model of semantic-based concurrency control presented in [20]. We have used two semantic levels for write operations: *update* and *replace* semantics. Update semantics is like a write operation in traditional databases. Replace semantics is used when the new value of an object does not depend on any value of any object in the database. This semantics is like a blind write operation but the transaction can perform a read (blind read) operation before replacing the object.

We use *Hoare logic* [11] in the semantic-based concurrency control where formulas are called *triples* and have form: $\{P\}S\{Q\}$, where P and Q are well-formed formulas of predicate logic, which we refer to as *preassertions* and *postassertions*, and S is a syntactically correct statement or sequence of statements in some imperative programming language. To overcome the limit of serializability, to increase performance, and extend this method to real-time database systems we propose a new correctness criterion called *real-time semantic correctness*. We assume that the execution of each statement of a transaction is atomic and isolated. A schedule of transactions is real-time semantically correct if

$$\{I\}Schedule\{I \wedge Q_{Schedule} \wedge T_{Schedule}\} \qquad (1)$$

is true. A real-time semantically correct schedule must maintain the consistency of the database, as indicated by the fact that I is a pre and postassertion of schedule. A real-time semantically correct schedule must also transform the database that reflects the cumulative results of all the transactions in schedule. We denote the assertion that describes set of states by $Q_{Shedule}$. Finally, real-time semantically correct schedule must also maintain time constraints of all the transactions in schedule and all temporal constraints of data items accessed by the transactions. We denote the assertion that describes set of time constraints by $T_{Schedule}$.

To motivate semantic correctness in real-time database setting, consider network database that mimics a Home Location Register (HLR) [30] which is used to store information about users of the network. Operators use HLR databases to store subscriber data, location data, network access data, and about network services data, for example call forwarding. To simplify presentations schema presented below does not contain all the operations of the HLR. Instead database and transactions are from The Telecom One (TM1) benchmark designed for telecommunication applications [29]. Database consists three tables: Subscriber, SpecialFacility, and CallForwarding. The basic data, such as the location data, of all subscribers using the network is found in the Subscriber table. Network services accessible

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

to a subscriber are stored in the SpecialFacility table. Each of those services might have a number of call forwardings, which are stored in the CallForwarding table.

- SUBSCRIBER($\underline{s\_id}$, cf_active, vlr_location, hlr_location)
- SPECIALFACILITY($\underline{s\_id}$, sf_type, is_active)
- CALLFORWARDING($\underline{s\_id}$, cf_number, sf_type, start_time, end_time)

Now consider a transaction GetNewDestination which retrieves a new call destination for a call. It joins the Subscriber, SpecialFacility and CallForwarding tables and retrieves one row with a given subscriber and a special facility type:

```
GetNewDestination(s_id, sf_type)
  BEGIN TRANSACTION
    {true}

    SELECT cf.cf_number
    INTO cf_number
    FROM Subscriber AS s, SpecialFacility AS sf,
         CallForwarding AS cf
    WHERE s.s_id = :s_id AND
          s.s_id = sf.s_id AND
          sf.sf_type = :sf_type AND
          sf.is_active = 1 AND
          sf.s_id = cf.s_id AND
          sf.sf_type = cf.sf_type AND
          current_time BETWEEN
            cf.start_time AND cf.end_time;

    {SQLSUCCESS ∧ :cf_number ∧
        current_time < deadline }
    COMMIT
  END TRANSACTION
```

Figure 1. GetNewDestination transaction.

2.1 Conflict Detection and Resolution

This section presents conflict detection and resolution using τ-serializability. We will call this method OCC-τDATI. Conflict detection is based on forward validation [8]. The number of transaction restarts is reduced by dynamic adjustment of the serialization order which is supported by similar timestamp intervals as in OCC-DATI [21]. Only difference between OCC-DATI and OCC-τDATI conflict detection is that OCC-τDATI allows transaction to read also from the old committed transaction (see Figure 2) for a τ-period based on transactions and the data item.

```
if (D_i ∈ RS(T_v))
  TI(T_v) = TI(T_v) ∩
      [WTS(D_i) + min(τ_v, τ_i, τ_{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);
```

Figure 2. Reading from committed transactions.

Second change between OCC-DATI and OCC-τDATI is in the conflict resolution where OCC-τDATI allows replacing a data item without a conflict with a read transaction (see Figure 3).

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

  if ( T_v.priority < T_a.priority )
      if ( TI == ∅ )
          restart(T_v); /* Validation ends here /*

  TI = TI ∩ [TS(T_v) + 1, ∞[ ;
  adjusted.push({(T_a, TI)});
}

backward_adjustment(T_a, T_v, adjusted, x)
{
  if (semantics(D_i, T_a) == replace )
      return; /* No conflict i.e allow replace */
  if (T_a ∈ adjusted)
    TI = adjusted.pop(T_a);
  else
    TI = TI(T_a);

  if ( T_v.priority < T_a.priority )
      if ( TI == ∅ )
          restart(T_v); /* Validation ends here /*

  TI = TI ∩ [0, TS(T_v) − 1 + min(τ_v, τ_a), τ_x] ;
  adjusted.push({(T_a, TI)});
}
```

Figure 3. Backward and Forward adjustment for the OCC-τDATI method.

3 Experimental results

We have carried out a set of experiments in order to examine the feasibility of our prototype implementation, specifically the concurrency control mechanism. All experiments were executed in the prototype database running on

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

a Intel Pentium 2400 GHz processor containing 512 MB of main memory with the Linux operating system 2.6.12.

All transactions arrive to the prototype through a specific user request interpreter subsystem, that gets the arriving transactions from an off-line generated test file. Every test session contains 20 000 transactions and is repeated at least 20 times. Reported values are the means of the replications. In particular, we examined how well our OCC-τDATI algorithm performs when compared to the OCC-TI [17] and OCC-DA [15] algorithm.

The test database represents a typical network database that mimics a Home Location Register (HLR) [30] which is used to store information about users of the network. Operators use HLR databases to store subscriber data, location data, network access data, and about network services data, for example call forwarding. To simplify presentations schema presented below does not contain all the operations of the HLR. Instead database and transactions are from The Telecom One (TM1) benchmark designed for telecommunication applications [29]. Transaction parameters used in these test are listed in the Table 1.

Table 1. Transaction parameters.

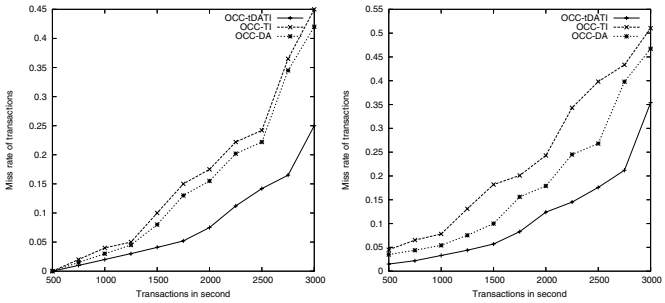
Transaction name	Test # #1	#2	Deadline	Mode
GetNewDestination	90 %	85 %	50 ms	read-only
UpdateDestination	10 %	10 %	100 ms	update
UpdateLocation	0 %	5 %	150 ms	replace

Transactions are validated atomically. If the deadline of a transaction expires, the transaction is always aborted. Other test parameters include the exponentially-distributed arrival rate. The final deadline is calculated using EDF method which is uses as a scheduling method.

3.1 Results of our experiments

In the first series of experiment tests we have used miss-ratio metric, which represents the fraction of transactions which miss their deadlines. In Figure 4(a) we have fixed fraction of write transactions to 10% (UpdateDestination) and varied the arrival rate of the transactions in second. We can clearly see that OCC-τDATI offers the best overall performance, OCC-DA is second best, and OCC-TI is clearly the worst algorithm. This confirms that the overhead for supporting dynamic adjustment in OCC-τDATI is smaller than the one in OCC-DA. This confirms also that the number of transaction restarts is smaller in OCC-τDATI compared to OCC-TI and OCC-DA. Our experiments also confirm the results in [15] that OCC-DA outperform OCC-TI.

A high data contention introduced by massive service management transactions often increases the number of concurrency control aborts in traditional optimistic concurrency control protocols. In addition, the validation time of transaction must be bounded in order to guarantee that real-time transactions will meet their deadlines. In the second series of experiments (Figure 4(b)), we examined how the execution of non-real time service management transactions affects the database throughput. Thus we have used all tree transactions presented earlier. Fraction of UpdateLocation transaction is 5% in the test. Experiment results confirm that OCC-τDATI outperforms OCC-DA and OCC-TI.



(a) 10 % writes (b) 15 % writes

Figure 4. OCC methods compared.

4 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 presented an optimistic concurrency control protocol called OCC-τDATI that takes into account the priority of transactions and uses a relaxed serializability. 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.

Compared to other OCC protocols that use dynamic serialization order adjustment, the proposed method OCC-

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

τDATI is much more efficient. Performance studies presented here confirm that OCC-τDATI outperforms both OCC-TI and OCC-DA. Experiment results using real-time database system for telecommunications clearly indicate that the proposed method is able to dynamically adapt changing workload situations.

References

[1] I. Ahn. Database issues in telecommunications network management. *ACM SIGMOD Record*, 23(2):37–43, June 1994.

[2] M. Appeldorn, R. Kung, and R. Saracco. TMN + IN = TINA. *IEEE Communications Magazine*, 31(3):78–85, Mar. 1993.

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

[4] 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.

[5] 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 *Proceedings of the Second International Workshop on Real-Time Databases: Issues and Applications*, pages 167–180. Kluwer Academic Publishers, 1997.

[6] 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, Nov. 1976.

[7] M. H. Graham. How to get serializability for real-time transactions without having to pay for it. In *Proceedings of the 14th IEEE Real-time Systems Symposium*, pages 56–65, 1993.

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

[9] 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, 1990.

[10] 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 Press, 1990.

[11] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–585, 1969.

[12] S.-L. Hung and K.-Y. Lam. Locking protocols for concurrency control in real-time database systems. *ACM SIGMOD Record*, 21(4):22–27, Dec. 1992.

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

[14] T. Kuo and A. K. Mok. Application semantics and concurrency control of real-time data-intensive applications. In *Proceedings of Real-Time System Symposium*, pages 76–86, 1993.

[15] 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. Springer, 1995.

[16] K.-Y. Lam, S.-L. Hung, and S. H. Son. On using real-time static locking protocols for distributed real-time databases. *The Journal of Real-Time Systems*, 13(2):141–166, Sept. 1997.

[17] 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. IEEE Computer Society Press, 1993.

[18] 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. Prentice-Hall, 1996.

[19] K.-J. Lin. Consistency issues in real-time database systems. In *Proceedings of the 22nd Hawaii Int. Conf. on System Sciences*, pages 654–661, 1989.

[20] K.-J. Lin and C. Peng. Enhancing external consistency in real-time transactions. *ACM SIGMOD Record*, 25(1):26–28, Mar. 1996.

[21] 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*, pages 13–20. IEEE Computer Society Press, 1999.

[22] K. Marzullo. Concurrency control for transactions with priorities. Tech. Report TR 89-996, Department of Computer Science, Cornell University, Ithaca, NY, May 1989.

[23] B. Purimetla, R. M. Sivasankaran, K. Ramamritham, and J. A. Stankovic. Real-time databases: Issues and applications. In S. H. Son, editor, *Advances in Real-Time Systems*, pages 487–507. Prentice Hall, 1996.

[24] K. Ramamritham. Real-time databases. *Distributed and Parallel Databases*, 1(2):199–226, Apr. 1993.

[25] K. Ramamritham and C. Pu. A formal characterization of epsilon serializability. *IEEE Transactions on Knowledge and Data Engineering*, 7(6), Dec. 1996.

[26] L. Sha, J. P. Lehoczky, and E. D. Jensen. Modular concurrency control and failure recovery. *IEEE Transactions on Computers*, 37(2):146–159, Feb. 1988.

[27] J. A. Stankovic, S. H. Son, and J. Hansson. Misconceptions about real-time databases. *IEEE Computer*, 32(6):29–36, June 1999.

[28] J. A. Stankovic and W. Zhao. On real-time transactions. *ACM SIGMOD Record*, 17(1):4–18, Mar. 1988.

[29] T. Strandell. Open source database systems: Systems study, performance and scalability. Series of publications C, master of science thesis, University of Helsinki, 2003.

[30] Wikipedia. Network switching subsystem. Technical report, http://en.wikipedia.org/wiki/Home_Location_Register.