Exploiting Main Memory DBMS Features to Improve Real-Time Concurrency Control Protocols

Özgür Ulusoy*
Department of Computer Engineering
Bilkent University
Ankara, TURKEY

Alejandro Buchmann
Department of Computer Science
Technical University Darmstadt
Darmstadt, GERMANY

1 Introduction

The Real-Time Database Systems (RTDBSs) Project at Bilkent University is concerned with various aspects of transaction scheduling in RTDBSs. Within the REACH (REal-time, Active and Heterogeneous Systems) Project at Darmstadt we have been working on engineering real-time DBMSs on top of the Chorus RTOS with a special emphasis on providing predictability and the ability to map high-level RT-protocols to basic RTOS schedulers. We are also working on combining active and real-time functionality [2]. Recently, both groups started to cooperate. The work described in this paper is joint work that resulted from this cooperation.

Predictability of transaction execution is a basic issue in RTDBSs. Because of the unpredictability of page faults and the time required for I/O (3-4 orders of magnitude higher than memory accesses) the work on RTDBS concurrency control has made one of two possible assumptions: either the performance metric used is simply the percentage of transactions finishing their execution within their deadline, without giving any guarantees for an individual transaction, or the underlying database is memory-resident. Main memory databases become increasingly feasible due to the drastically falling memory prices and growing memory sizes. However, the concurrency control protocols developed so far for RTDBSs are derived from disk-resident DBMS concurrency control algorithms and do not exploit the inherent properties of main memory databases.

An assumption underlying the recently published concurrency control protocols for RTDBSs is that predeclaration of locks is impractical because it is impossible to predict which instance of a relation will actually be accessed by a transaction. Therefore, the model of dynamic resource acquisition is assumed. Those protocols spend considerable CPU time for the detection and resolution of data conflicts and fine-grained lock management. This overhead reduces the effective CPU utilization, and leads to an increase in the number of missed transaction deadlines.

It has been demonstrated that very large lock granularities (e.g., relations) are most appropriate in main memory databases [3], [4]. When I/O is eliminated, the primary advantage of small lock granularities is effectively removed. Under the considerations for lock granularity in main-memory databases, we believe that concurrency control and scheduling with predeclaration of resources should be revisited for RTDBSs. It is simple to determine by purely syntactic means during transaction compilation the read and write sets of the transaction at the relation level. If the lock granularity is the relation, it becomes feasible to use simple but efficient concurrency control and scheduling mechanisms based on resource predeclaration.

In this paper, we propose a simple, predeclaration-based concurrency control protocol for main-memory RTDBSs. Predeclaration protocols require the knowledge of what resources will be used ahead of time and the granularity at which resources are locked. A transaction, when submitted, is always parsed and compiled into an internal, optimizable form. During parsing it is possible to detect with simple syntactical means which relations will be accessed by a given transaction. It is further possible to detect in the same parsing step whether the proposed access is for reading or writing purposes. Under these conditions the read and write sets for a transaction can be established a priori, and the transaction can be scheduled in a conflict-free manner by preacquiring the necessary resources. By doing this, a transaction will execute without blocking and will minimize its time in the system. Data resources will be locked only the time required for the actual computation of a transaction. The CPU as the limiting resource will be fully used and less time will be wasted on lock management. We claim that our protocol is much more efficient than previously proposed protocols with dynamic resource acquisition. It also offers the possibility of determining execution times without the effects of blocking thereby allowing us to give guarantees for the execution of high-priority transactions.

We also describe a performance model designed for studying various issues in main-memory RTDBSs. We are using this model to compare our protocol against other concurrency control protocols from the literature. The performance results

*The work of Ulusoy was supported by the Research Council of Turkey (TÜBİTAK) grant no. EEEAG-137.

For each transaction T submitted to the system,
OFF-LINE:
Parse transaction T, identify the relations to be accessed, and construct *read_set[T]* and *write_set[T]*.
ON-LINE:
Set *read_blocked[T]* and *write_blocked[T]* to empty.
For each transaction T' that is either executing or in the *ready-queue*,
read_blocked[T] = *read_blocked[T]* $\cup$ (*read_set[T]* $\cap$ *write_set[T']*)
write_blocked[T] = *write_blocked[T]* $\cup$ (*write_set[T]* $\cap$ *read_set[T']*) $\cup$ (*write_set[T]* $\cap$ *write_set[T']*)
EndFor.
If both *read_blocked[T]* and *write_blocked[T]* are empty
insert T into *ready-queue*.
else,
insert T into *wait-queue*.
EndIf.

When a transaction T commits,
The locks on the relations in *read_set[T]* and *write_set[T]* are released.
For each transaction T' in the *wait-queue*,
read_blocked[T'] = *read_blocked[T']* - *write_set[T]*
write_blocked[T'] = *write_blocked[T']* - *write_set[T]* - *read_set[T]*
If both *read_blocked[T']* and *write_blocked[T']* become empty
T' is transferred to the *ready-queue*.
EndIf.
EndFor.
The highest priority transaction in the *ready-queue* is started to execute after granting its locks.

Figure 1: The concurrency control protocol.

will be presented in the full version of the paper.

2 The Concurrency Control Protocol

For each transaction submitted to the system, the list of relations to be accessed by the transaction, and the mode of each access (i.e., either read or write) are determined. Then, a conflict check is performed between the relations to be accessed by the new transaction and the relations in the access list of already scheduled transactions. If no conflict is detected, the transaction is inserted into the ready-queue. Otherwise, the transaction is inserted into the wait-queue, and two sets of blocked relations, one for read access and one for write access, are established for the transaction. Both the ready-queue and the wait-queue are organized on the basis of transaction priorities. A variety of criteria can be used to determine a transaction's priority.

When a transaction is committed, its read and write locks are released. The set of relations which were write-accessed (read/write-accessed) by the committed transaction is intersected with the read-blocked (write-blocked) relation set of each transaction in the wait-queue. The intersecting elements are eliminated from the blocked relations sets. If, for any transaction in the wait-queue, both the read-blocked and write-blocked relations sets become empty, that transaction is transferred to the ready-queue. Following each commitment, the first transaction in the ready queue is started to execute.

2.1 Formal Description of the Protocol

The data structures associated with the protocol are:

- *read_set[T]*: The set of relations to be read by transaction T.
- *write_set[T]*: The set of relations to be updated by transaction T.
- *read_blocked[T]*: The set of relations in *read_set[T]*, but currently write-locked by some other transactions.
- *write_blocked[T]*: The set of relations in *write_set[T]*, but currently read or write-locked by some other transactions.
- *ready-queue*: The list of transactions that are ready to execute.
- *wait-queue*: The list of transactions that have some access requests conflicting with the scheduled transactions.

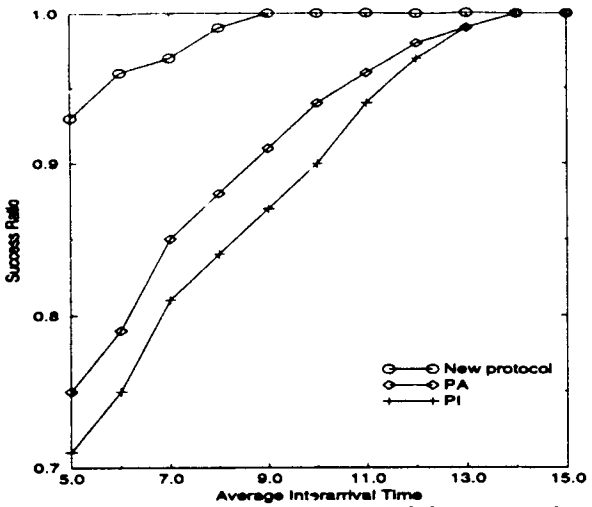

Figure 2: Real-time performance of the protocols as a function of the average interarrival time of transactions.

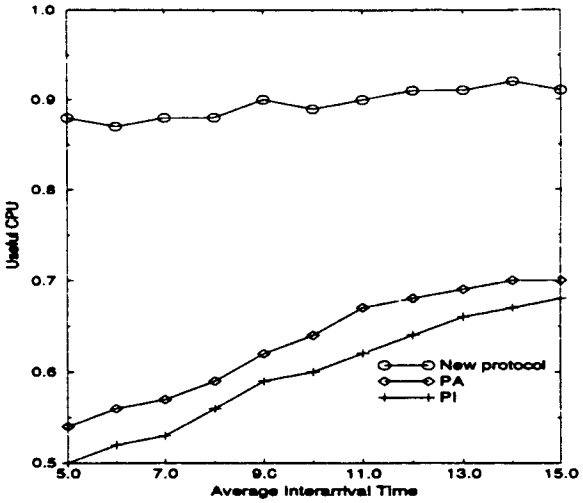

Figure 3: Useful CPU time results.

Figure 1 provides a formal description of our RTDB concurrency control protocol.

3 A Real-Time Database System Model and Preliminary Results

In this section, we sketch the RTDBS simulation model used to evaluate the performance of the proposed concurrency control protocol and give some preliminary results. The model is based on an open queuing model of a multiprocessor, memory-resident database system.

The entire database is kept in main memory, while a stable copy, possibly out of date, is kept on disk. For each transaction the disk is only accessed once to write the log record onto disk and guarantee write-ahead logging. The disk-resident copy of the database can be updated asynchronously by applying the log (possibly on a separate processor). In the case of system failure, the database can be recovered from the stable copy and the log. Since application of the log records to the disk-resident version of the database can be done off-line, it is safe to assume that this process does not interfere with regular transaction processing. Therefore, the only I/O cost paid by a transaction is the writing of the log. A feasible alternative for writing to the disk is broadcasting the log to other machines, thus reducing the delay by almost an order of magnitude.

Transaction arrivals are assumed to be Poisson. Each transaction is associated with a real-time constraint in the form of a deadline. The transactions are prioritized based on the *earliest deadline first* policy; i.e., a transaction with an earlier deadline has higher priority than a transaction with a later deadline.

The details of the main-memory RTDBS model were captured in a simulation program. The performance of our protocol was studied and compared against the performance of some other concurrency control protocols proposed recently for RTDBSs. Two protocols selected for comparison are the Priority Abort protocol [1] and the Priority Inheritance protocol [5]. The initial results we obtained under different levels of transaction load are quite encouraging for the performance of our protocol. Figure 2 presents the preliminary results obtained for the relative performance of the protocols in terms of the fraction of transactions that satisfy their deadlines. The new protocol provides considerably better performance than the others. It enables the system to spend more useful CPU time on transaction processing, and as a result, even under very high loads only a few transactions miss their deadlines. Figure 3 presents the results obtained for *useful CPU time* which specifies the fraction of CPU time that is not wasted (i.e., used for processing the operations of committed transactions).

References

[1] R. Abbott, H. Garcia-Molina, 'Scheduling Real-Time Transactions: A Performance Evaluation', *ACM Transactions on Database Systems*, vol.17, no.3, pp.513-560, 1992.

[2] H. Branding, A. Buchmann, 'On Providing Soft and Hard Real-Time Capabilities in an Active DBMS', *International Workshop on Active and Real-Time Database Systems*, Skovde, Sweden, 1995.

[3] H. Garcia-Molina, K. Salem, 'Main-Memory Database Systems', *IEEE Transactions on Knowledge and Data Engineering*, vol.4, no.6, pp.509-516, 1992.

[4] T.J. Lehman, M.J. Carey, 'A Recovery Algorithm for a High-Performance Memory-Resident Database System', *ACM SIGMOD Conference*, pp.104-117, 1987.

[5] L.Sha, R.Rajkumar, S.H.Son, C.H.Chang, 'A Real-Time Locking Protocol', *IEEE Transactions on Computers*, vol.40, no.7, pp.793-800, 1991.