Using Logs to Increase Availability in Real-Time Main-Memory Database

Tiina Niklander and Kimmo Raatikainen

University of Helsinki, Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki,Finland
{tiina.niklander,kimmo.raatikainen}@cs.Helsinki.FI

Abstract. Real-time main-memory databases are useful in real-time environments. They are often faster and provide more predictable execution of transactions than disk-based databases do. The most reprehensible feature is the volatility of the memory. In the RODAIN Database Architecture we solve this problem by maintaining a remote copy of the database in a stand-by node. We use logs to update the database copy on the hot stand-by. The log writing is often the most dominating factor in the transaction commit phase. With hot stand-by we can completely omit the disk update from the critical path of the transaction, thus providing more predictable commit phase execution, which is important when the transactions need to be finished within their deadlines.

1 Introduction

Real-time databases will be an important part of the future telecommunications infrastructure. They will hold the information needed in operations and management of telecommunication services and networks. The performance, reliability, and availability requirements of data access operations are demanding. Thousands of retrievals must be executed in a second. The allowed unscheduled down time is only a few minutes per year. The requirements originate in the following areas: real-time access to data, fault tolerance, distribution, object orientation, efficiency, flexibility, multiple interfaces, and compatibility [13, 14]. Telecommunication requirements and real-time database concepts are studied in the literature [1–3, 7].

The RODAIN[1] database architecture is a *real-time*, *object-oriented*, *fault-tolerant*, and *distributed* database management system, which is designed to fulfill the requirements of a modern telecommunications database system. It offers simultaneous execution of firm and soft deadline transactions as well as transactions that do not have deadlines at all. It supports high availability of the data using a hot stand-by, which maintains a copy of the operational database. The hot stand-by is ready to switch to the database server at any time, if the primary server fails. Related systems include ClustRa [4], Dalí[5], and StarBase [6].

[1] RODAIN is the acronym of the project name *Real-Time Object-Oriented Database Architecture for Intelligent Networks* funded by Nokia Networks, Solid Information Technology, and the National Technology Agency of Finland.

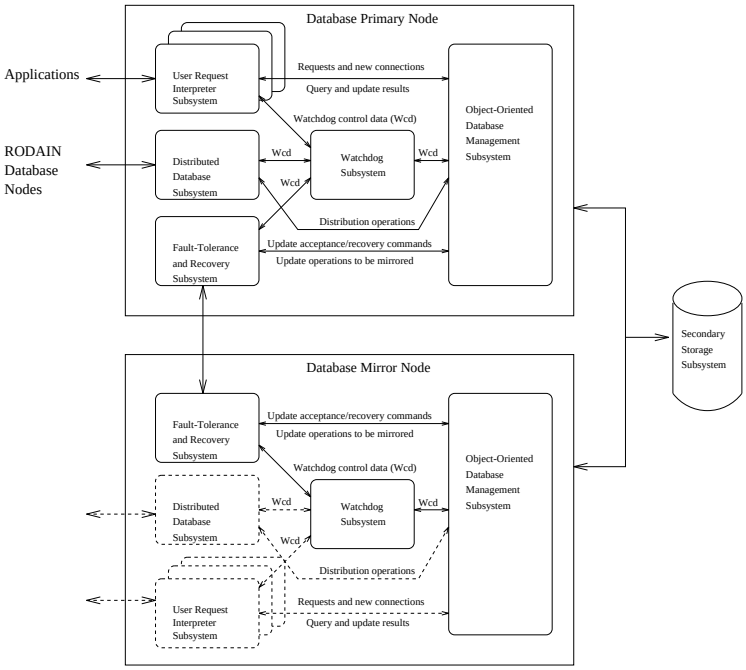


Fig. 1. The Architecture of RODAIN Database Node.

The rest of the paper is organized as follows. The architecture of the RODAIN Database Management System is presented in section 2. The logging mechanism is presented in detail in section 3. Finally, in section 4 we will summarize the results of our experiments, based on a prototype implementation of the RODAIN database system.

2 RODAIN Database

A database designed to be used as a part of telecommunication services must give quick and timely responses to requests. In the RODAIN Database System (see Fig. 1) this is achieved by keeping time-critical data in the main-memory database and using real-time transactions. Real-time transactions have attributes like criticality and deadline that are used in their scheduling. Data availability is increased using a hot stand-by node to maintain a copy of the main-memory database. The hot stand-by, which we call the Database Mirror Node, can replace the main database server, called the Database Primary Node, in the case of failure.

Our main goal in the database design was to avoid as much of the overhead of rollbacks during transaction abort as possible. This was achieved using the deferred write mechanism. In a deferred write mechanism the transaction is allowed to write the modified data to the database area only after it is accepted to

commit by the concurrency control mechanism. This way the aborted transaction can simply discard its modified copies of the data without rollbacking. An aborted transaction is either discarded or restarted depending on its properties.

For concurrency control, we chose to use an optimistic concurrency control protocol. Such a protocol seems appropriate to our environment with main-memory data and mainly short, read-only transactions with firm deadlines. We combined the features of OCC-DA [8] and OCC-TI [9], thus creating our own concurrency control protocol called OCC-DATI [11] which reduces the number of unnecessary restarts.

A modified version of the traditional Earliest Deadline First (EDF) scheduling is used for transaction scheduling. The modification is needed to support a small number of non-realtime transactions that are executed simultaneously with the real-time transactions. Without deadlines the non-realtime transactions get the execution turn only when the system has no real-time transaction ready for execution. Hence, they are likely to suffer from starvation. We avoid this by reserving a fixed fraction of execution time for the non-realtime transactions. The reservation is made on a demand basis.

To handle occasional system overload situations the scheduler can limit the number of active transactions in the database system. We use the number of transactions that have missed their deadlines within the observation period as the indication of the current system load level.

The synchronization between Primary and Mirror Nodes within the RODAIN Database Node is done by transaction logs and it is the base for the high availability of the main-memory database. Transactions are executed only on the Primary Node. For each write and commit operation a transaction redo log record is created. This log is passed to the Mirror Node before the transaction is committed. The Mirror Node updates its database copy accordingly and stores the log records to the disk. The transaction is allowed to commit as soon as the log records are on the Mirror Node removing the actual disk write from the critical path. It is like the log handling done in [10], except that our processors do not share memory. Thus, the commit time needed for a transaction contains one message round-trip time instead of a disk write.

The database durability is trusted on the assumption that both nodes do not fail simultaneously. If this fails, our model might loose some committed data. This data loss comes from the main idea of using the Mirror Node as the stable storage for the transactions. The data storing to the disk is not synchronized with the transaction commits. Instead, the disk updates are made after the transaction is committed. A sequential failure of both nodes does not lose data, if the time difference between the failures is large enough for the Mirror Node to store the buffered logs to the disk.

The risk of loosing committed data decreases when the time between node failures increases. As soon as the remaining node has had enough time to store the remaining logs to the disk, no data will be lost. In telecommunication the minor risk of loosing committed data seems to be acceptable, since most updates

handle data that has some temporal nature. The loss of temporal data is not catastrophic, it will be updated again at a later time.

During node failure the remaining node, called the Transient Node, will function as Primary Node, but it must store the transaction logs directly to the disk before allowing the transaction to commit. The failed node will always become a Mirror Node when it recovers. This solution avoids the need to switch the database processing responsibilities from the currently running node to another. The switch is only done when the current server fails and can no longer serve any requests.

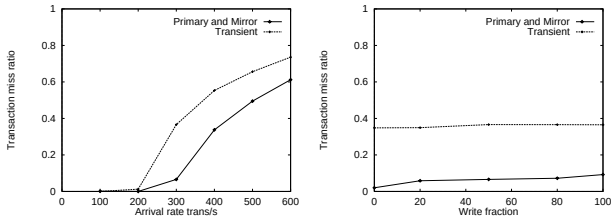
3 Log Handling in the RODAIN Database Node

Log records are used for two different purposes in the RODAIN Database Node. Firstly, they are used to maintain an up-to-date copy of the main-memory database on a separate Mirror Node in order to recover quickly from failures of the Primary Node. Secondly, the logs are stored in a secondary media in the same way as in a traditional database system. These logs are used to maintain the database content even if both nodes fail simultaneously, but they can be also used for, for example, off-line analysis of the database usage.

The log records containing database updates, the after images of the updated data items, are generated during the transaction's write phase. At the write phase the transaction is already accepted for commit and it just updates the data items it has modified during its execution. Each update also generates a log record containing transaction identification, data item identification and an after image of the data item. All transactions that have entered their write phases will eventually commit, unless the primary database system fails. When the Primary Node fails, all transactions that are not yet committed are considered aborted, and their modifications to the database are not performed on the database copy in the Mirror Node.

The communication between the committing transaction and the Log writer is synchronous. The Log Writer on the Primary Node sends the log records to the Mirror Node as soon as they are generated. When the Mirror Node receives a commit record, it immediately sends an acknowledgment back. This acknowledgment is used as an indication that the logs of this specific transaction have arrived to the Mirror Node. The Log writer then allows the transaction to proceed to the final commit step. If a Mirror Node does not exist, then the Log writer (on Transient Node) must store the logs directly to the disk.

The logs are reordered based on transactions before the Mirror Node updates its database copy and stores the logs on disk. The true validation order of the transactions is used for the reordering. This reordering simplifies the recovery process. With logs already ordered, the recovery can simply pass the log once from the beginning to the end omitting only the transactions that do not have a commit record in the log. Likewise, The Mirror Node performs the logged updates to its database only when it has also received the commit record. This way it can be sure that it never needs to undo any changes based on logs.



(a) write ratio 50 %

(b) arrival rate 300 trp/s

Fig. 2. Comparison of normal mode, both Primary and Mirror Node present, and transient mode, only Transient Node, using true log writes.

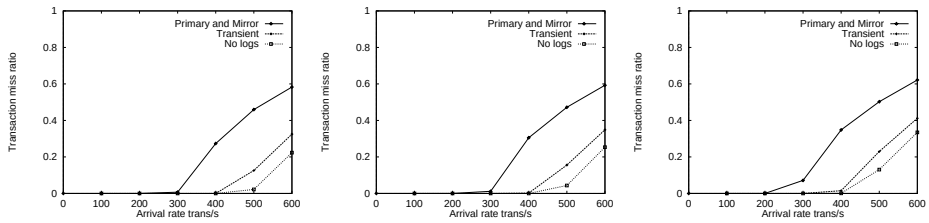
4 Experimental Study

The current implementation of the RODAIN Database Prototype runs on a Chorus/ClassiX operating system [12]. The measurements were done on computers with a Pentium Pro 200MHz processor and 64 MB of main memory. All transactions arrive at the RODAIN Database Prototype through a specific interface process, that reads the load descriptions 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. The test database, containing 30 000 data objects, represents a number translation service. The number of concurrently running transactions is limited to 50. If the limit is reached, an arriving lower priority 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 a variable mix of two transactions, one simple read-only transaction and the other a simple write transaction. The read-only service provision transaction reads a few objects and commits. The write transaction is an update service provision transaction that reads a few objects, updates them and then commits. The relative firm deadline of all real-time transactions is 50ms and the deadline of all write transactions is 150ms.

We measured the transaction miss ratio, which represents the fraction of transactions that were aborted. The aborts can be either due to the exceeding of a transaction deadline, a concurrency control conflict, or an acceptance denial due to the load limit. In the experiments, the failures in transaction executions were mainly due to system overload. Occasionally a transaction also exceeded its deadline and was, therefore, aborted.

We compared the performance of our logging mechanism in its normal use with both the Primary and the Mirror Node to a situation where only a single Transient node is running (see Fig. 2). When both nodes are up and running the logs are passed from the Primary to the Mirror Node. When the Transient Node is running alone, it stores the logs directly to the log storage. The experiment



(a) write ratio 0% (b) write ratio 20% (c) write ratio 80%

Fig. 3. Comparison of optimal (marked as No logs), single node (Transient), and two node systems (Primary and Mirror).

shows clearly that the use of a remote node instead of direct disk writes increases the system performance.

Since our experiments with disk writing showed that the log storing to the disk can easily become the bottleneck in the log handling, we ran more tests with the disk writing turned off. This scenario is feasible, if the probability of simultaneous failure of both nodes is acceptable and the system can be trusted to run without any other backups. The omission of the disk writes also emphasizes the overhead from our log handling algorithms with the two nodes. If the log storing to the disk system is slower than the median log generation rate, then the system gets trashed from the buffered logs and must reduce the incoming ratio of the transactions to the pace of disk storing. This would then remove most of the benefit of the Mirror Node use. For comparison, we also ran tests on a Transient Node where the logging feature was completely turned off. The results from this optimal situation do not differ much from the results of Transient Node with logging turned off.

From Fig. 3 we can see that the most effective feature of the system performance is the transaction arrival time. At the arrival rate of 200 to 300 transactions per second depending on the ratio of update transactions, the system becomes saturated and most of the unsuccessfully executed (=missed) transactions are due to abortions by overload manager. The effect of the ratio of update transactions is relatively small. There are two reasons for this behavior. First, the update transactions modify only a few items. Thus, the number of log records per transaction is not large either. Secondly, the system generates a commit log record also for read-only transactions, thus forcing the commit times of both transaction types to be quite close.

The benefits of the use of the hot stand-by are actually seen when the primary database system fails. When that happens, the Mirror Node can almost instantaneously serve incoming requests. If, however, the Primary Node was alone and had to recover from the backup on the disk or in the stable memory, like Flash, the database would be down much longer. Such down-times are not allowed in certain application areas such as telecommunication.

5 Conclusion

The RODAIN database architecture is designed to meet the challenge of future telecommunication systems. In order to fulfill the requirements of the next generation of telecommunications systems, the database architecture must be fault-tolerant and support real-time transactions with explicit deadlines. The internals of the RODAIN DBMS described are designed to meet the requirements of telecommunications applications. The high availability of the RODAIN Database is achieved through using a database mirror. The mirror is also used for log processing, which reduces the load at the primary database node and shortens the commit times of transactions allowing more transactions to be executed within their deadlines.

References

1. I. Ahn. Database issues in telecommunications network management. *ACM SIGMOD Record*, 23(2):37–43, 1994.
2. R. Aranha et al. Implementation of a real-time database system. *Information Systems*, 21(1):55–74, 1996.
3. T. Bowen et al. A scale database architecture for network services. *IEEE Communications Magazine*, 29(1):52–59, January 1991.
4. S. Hvasshovd et al. The ClustRa telecom database: High availability, high throughput, and real-time response. In *Proc. of the 21th VLDB Conf.*, pp. 469–477, 1995.
5. H. Jagadish et al. Dalí: A high performance main memory storage manager. In *Proc. of the 20th VLDB Conf.*, pp. 48–59, 1994.
6. Y. Kim and S. Son. Developing a real-time database: The StarBase experience. In A. Bestavros, K. Lin, and S. Son, editors, *Real-Time Database Systems: Issues and Applications*, pp. 305–324. Kluwer, 1997.
7. Y. Kiriha. Real-time database experiences in network management application. Tech. Report CS-TR-95-1555, Stanford University, USA, 1995.
8. K. Lam, K. Lam, and S. Hung. An efficient real-time optimistic concurrency control protocol. In *Proc. of the 1st Int. Workshop on Active and Real-Time Database Systems*, pp. 209–225. Springer, 1995.
9. J. 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*, pp. 429–460. Prentice-Hall, 1996.
10. T. Lehman and M. Carey. A recovery algorithm for a high-performance memory-resident database system. In U. Dayal and I. Trager, editors, *Proc. of ACM SIGMOD 1987 Ann. Conf.*, pp. 104–117, 1987.
11. J. Lindström and K. Raatikainen. Dynamic adjustment of serialization order using timestamp intervals in real-time databases. In *Proc. of 6th Int. Conf. on Real-Time Computing Systems and Applications*, 1999.
12. D. Pountain. The Chorus microkernel. *Byte*, pp. 131–138, January 1994.
13. K. Raatikainen. Real-time databases in telecommunications. In A. Bestavros, K. Lin, and S. Son, editors, *Real-Time Database Systems: Issues and Applications*, pp. 93–98. Kluwer, 1997.
14. J. Taina and K. Raatikainen. Experimental real-time object-oriented database architecture for intelligent networks. *Engineering Intelligent Systems*, 4(3):57–63, September 1996.