Chapter 16

DATA REPLICATION AND AVAILABILITY

Özgür Ulusoy
Department of Computer Engineering
Bilkent University
06533 Bilkent, Ankara, TURKEY
oulusoy@cs.bilkent.edu.tr

1. INTRODUCTION

In a *replicated database system* copies of data can be stored redundantly at multiple sites. The potential of data replication for high data availability and improved read performance is crucial to distributed real-time database systems (RTDBSs). On the other hand, data replication introduces its own problems. Access to a data item is no longer controlled exclusively by a single site, instead the access control is distributed across the sites each storing a copy of the data item. It is necessary to ensure that *mutual consistency* of the replicated data is provided; in other words, replicated copies must behave like a single copy. This can be made possible by preventing conflicting accesses on the different copies of the same data item, and by making sure that all data sites eventually receive all updates [7]. Multiple copy updates lead to a considerable overhead due to the communication required among the data sites holding the copies.

In this chapter, a brief overview of the research efforts that has addressed the data replication problem in distributed RTDBSs is provided together with our work which investigates the impact of storing multiple copies of data on satisfying timing constraints of real-time transactions. In our work, a performance model of a distributed RTDBS is employed to evaluate the effects of various workload parameters and design alternatives on system performance. The primary performance issue considered is the satisfaction of transaction deadlines; more specifically, an answer to the following question is looked for: 'does replication of data always aid in satisfying timing constraints of transactions?'. Various experiments are conducted to identify the conditions under which data replication can help real-time transactions satisfy their timing constraints. Different application types are considered in evaluating the effects

of the degree of data replication. Each application is distinguished by the type (query vs update) and data access distribution (local vs remote) of the processed transactions.

2. RESEARCH EFFORTS

Although the majority of the previous works involving distributed database models assumed either *no-replication* or *full-replication,* some performance evaluation studies of *partially replicated* database systems were also provided (e.g., [1] , [4], [5], [9]). The impact of the level of data replication on the performance of conventional database systems was examined in those studies considering the average response time of the transactions and the system throughput to be the basic performance measures. It was found in those evaluations that increasing data replication usually leads to some performance degradation due to the overhead of update synchronization among replicated copies of data.

Very little amount of work has appeared in the literature exploring data replication and availability in RTDBSs. In an early work in this field, Lin and Lin proposed some techniques to enhance the availability of replicated real-time databases [8]. They suggested that a transaction characterized by a strict deadline should be able to execute even if the most up-to-date data copies are not available, so that the mutual consistency requirement can be relaxed for distributed RTDBSs that process hard deadline transactions. They also introduced the *user quorum* scheme to increase the availability in a partitioned RTDBS. The scheme is different from traditional quorum protocols in that it gives access rights to a partition with a majority of users rather than a partition with a majority of data copies.

Son and Kouloumbis proposed a replication control algorithm for distributed RTDBSs [12], [13], [14]. The algorithm integrates real-time scheduling with data replication control. It employs *epsilon serializability*[1] as the correctness criterion to provide more concurrency to real-time transactions. In the algorithm, real-time scheduling features are involved in responding to timing requirements of transactions, and a *token-based synchronization scheme* is used to control data replication. Performance issues of the algorithm were investigated using a prototyping environment [12], [14].

In a more recent paper, Son and Zhang provided another data replication control algorithm for distributed RTDBSs [15]. Similar to the algorithm provided in [14], the new algorithm also uses epsilon serializability as the correctness

[1]Epsilon serializability offers the possibility of maintaining mutual consistency of replicated data asynchronously [10]. It allows queries to execute without being blocked or aborted by update transactions. Inconsistent data may be seen by some queries, but data will eventually converge to a consistent state. Also, the degree of inconsistency in query results can be controlled.

criterion. However, unlike the previous algorithm which involves a token-based synchronization scheme, this algorithm controls data replication on the basis of the *majority consensus approach.* It was shown through simulation experiments that the real-time performance of the new algorithm, in terms of the percentage of satisfied transaction deadlines, is better than that of the token-based algorithm under low levels of data conflicts. The implementation overhead of the algorithm is lower, however, the degree of concurrency provided is not as high as that allowed by the token-based algorithm.

Xiong et al. proposed a concurrency control protocol, specially designed for firm-deadline applications operating on replicated real-time databases [17]. The proposed protocol augments a non-real time concurrency control protocol with a novel state-based conflict resolution method to fine-tune the real-time performance. Compared to locking-based real-time concurrency control protocols, the performance of the new protocol was shown to be the best in replicated environments for real-time applications. It was also shown that the relative performance of replica concurrency control algorithms in real-time environments can be significantly different from that in non-real-time environments.

Our work in this field involves investigation of the impact of storing multiple copies of data on satisfying timing constraints of RTDBS transactions [16]. A brief description of this work is provided in the following.

3. A REPLICATED REAL-TIME DATABASE SYSTEM MODEL

This section provides the model of a RTDBS model that we used in investigation of the effects of data replication on real-time performance of the system. In this model, a number of data sites are interconnected by a local communication network. Each data site contains a transaction manager, a resource manager, a message server, a scheduler, and a recovery manager.

Each transaction is characterized by a *criticalness* and a *deadline.* The criticalness of a transaction is an indication of its level of importance [3]. The most critical transactions are assigned the highest level. Transaction deadlines are *soft;* i.e., each transaction is executed to completion even if it misses its deadline. The transaction manager at the originating site of a transaction T assigns a real-time priority to transaction T based on its criticalness (C_T), deadline (D_T), and arrival time (AT). The priority of transaction T is determined by the following formula:

$$P_T = \frac{C_T}{D_T - A_T}$$

There is no globally shared memory in the system, and all sites communicate via message exchanges over the communication network. A message server at each site is responsible for sending/receiving messages to/from other sites.

Access requests for data items are ordered by the scheduler on the basis of the concurrency control protocol executed.

I/O and CPU services at each site are provided by the resource manager. I/O service is required for reading or updating data items, while CPU service is necessary for processing data items and communication messages. Both CPU and I/O queues are organized on the basis of real-time priorities, and preemptive-resume priority scheduling is used by the CPUs at each site.

3.1 DISTRIBUTED TRANSACTION EXECUTION

Each distributed transaction exists in the system in the form of a master process that executes at the originating site of the transaction and a number of cohort processes that execute at various sites where the the copies of required data items reside. The transaction manager is responsible for creating a master process for each new transaction and specifying the appropriate sites for the execution of the cohort processes of the transaction. A transaction can have at most one cohort at each data site. For each operation executed, a global data dictionary is referred to find out the locations of the data item referenced by the operation. The coordination of the execution of remote cohorts is provided by the master process through communicating with the transaction manager of each cohort's site.

When a cohort completes its data access and processing requirements, it waits for the master process to initiate two-phase commit. The master process commits a transaction only if all the cohort processes of the transaction run to completion successfully, otherwise it aborts and later restarts the transaction. A restarted transaction accesses the same data items as before, and is executed with its original priority. The cohorts of the transaction are reinitialized at relevant data sites.

One-copy serializability in replicated database systems can be achieved by providing both concurrency control for the processed transactions and mutual consistency for the copies of a data item. In our model, mutual consistency of replicated data is achieved by using the *read-one, write-all-available* scheme [2]. Based on this scheme, a read operation on a data item can be performed on any available copy of the data. In order to execute a write operation of a transaction on a data item, each transaction cohort executing at an operational data site storing a copy the item is activated to perform the update on that copy.

3.2 DATA DISTRIBUTION

We used a data distribution model which provides a partial replication of the distributed database. The model enables us to execute the system at precisely specified levels[2] of data replication. Each data item has exactly N copies in the distributed system, where $1 \leq N \leq n$. Each data site can have at most one copy of a data item. The remote copies of a data item are uniformly distributed over the remote data sites; in other words, the remote sites for the copies of a data item are chosen randomly. If the average database size at a site is specified by *db_size,*

$$db\_size = N * local\_db\_size$$

where *local_db_size* represents the database size originated at each site. Note that $N = 1$ and $N = n$ correspond to the no-replication and full-replication cases, respectively.

3.3 RELIABILITY ISSUES

The distributed RTDBS model assumes that the data sites fail in accordance with an exponential distribution of inter-failure times. After a failure, a site stays halted during a repair period, again chosen from an exponential distribution. The means of the distributions are determined by the parameters *mtbf* (mean time between failures) and *mttr* (mean time to repair). The recovery manager at each site is responsible for handling site failures and maintaining the necessary information for that purpose. The communication network, on the other hand, is assumed to provide reliable message delivery and is free of partitions. It is also assumed that the network has enough capacity to carry any number of messages at a given time, and each message is delivered within a finite amount of time.

The following subsections details the reliability issues considered in our distributed system model.

3.3.1 Availability. *Availability* of a system specifies when transactions can be executed [6]. It is intimately related to the replica control strategy used by the system. For the read-one, write-all-available strategy, availability can be defined as the fraction of time (or probability) for which at least one copy of a data item is available to be accessed by an operation [9]. This strategy provides a high level of availability, since the system can continue to operate when all but one site have failed. A transaction that issues an operation that fails is blocked until a copy of the requested data item becomes available.

[2]The level of replication corresponds to the number of copies that exist for each data item.

One method to measure the availability of an executing system is to keep track of the total number of attempted and failed operations experienced over a long period of time. Availability in our model is defined by the following formula:

$$Availability = \frac{Total\ no.\ of\ successful\ (read\ and\ write)\ operations}{Total\ no.\ of\ (read\ and\ write)\ operation\ requests}$$

This formula is a convenient one to use in RTDBSs since both read and write availabilities are equally crucial to such systems, and thus they can be treated together.

3.3.2 Site Failure. At a given time a site in our distributed system can be in any of three states: *operating, halted,* or *recovering.* A site is in the halted state if it has ceased to function due to a hardware or software failure. A site failure is modeled in a fail-stop manner; i.e., the site simply halts when it fails [11]. Following its repair, the site is transformed from the halted state to the recovering state and the recovery manager executes a predefined recovery procedure. A site that is operational or has been repaired is said to be in the operating state. Data items stored at a site are available only when the site is in the operating state.

A list of operating sites is maintained by the recovery manager at each site. The list is kept current by 'up-state' messages received from remote sites. An 'up-state' message is transmitted periodically by each operating site to all other sites. When the message has not been received from a site for a certain timeout period, the site is assumed to be down.

3.3.3 Site Recovery Strategy. The recovery procedure at a site restores the database of that site to a consistent and up-to-date state. Our work does not simulate the details of site recovery; instead, it includes a simplified site recovery model which is sufficient for the purpose of estimating the impact of site failures on system performance.

The recovery manager at each site maintains a log for recovery purposes, which records the chronological sequence of write operations executed at that site. Whenever a write operation is performed by a transaction, a log record for that write is created before the database is modified. At the commit time of a transaction, a commit record is written in the log at each participating data site. In the case of a transaction abort, the log records stored for that transaction are simply discarded. The recovery manager of a recovering site first performs local recovery by using the local log. Then, it obtains the logs kept at operating sites to check whether any of its replicated data items were updated while the site was in the halted state. It then refreshes the values of updated items using the current values of the copies stored at operational sites.

4. SUMMARY OF PERFORMANCE EVALUATION RESULTS

Details of the replicated database system model described in the preceding section were captured in a simulation program. The performance metric used in the experiments was *success-ratio;* i.e., the fraction of transactions that satisfy their deadlines.

Different application types were considered in evaluating the effects of level of data replication on satisfying transaction deadlines. Each type considered is characterized by the fraction of update transactions processed and the distribution of accessed data items (local vs remote). In execution environments where queries predominate and the data items at all sites are accessed uniformly, increasing the level of replication helped transactions meet their deadlines. For the application types where the majority of processed transactions are of update type, having many data copies was not attractive. This result was due to the fact that the overhead of update synchronization among the multiple copies of updated data increases with each additional data copy. Concurrency control protocols which employ restarts in scheduling, exhibited better performance than blocking-based protocols in query-based application environments when the level of data replication was low.

With update-oriented applications blocking-based protocols outperformed restart-based protocols, leading to the result that the overhead of executing a blocking-based concurrency control protocol is less than that of a restart-based one when the update transactions dominate in the system. Blocking-based protocols become more preferable as the level of data replication increases, since the performance of restart-based protocols is affected more negatively by the increased overhead of multiple copy updates. Aborting a transaction becomes more expensive as the number of copies of the data items updated by the transaction increases. It was shown that aborting a lock-holding transaction should be considered only in the case that the transaction is at the early stages of its execution.

We also studied the impact of site failures on system performance under different system reliability levels. Investigating the effectiveness of data replication in preventing the effects of site failures, we observed that replication turns out to be more desirable as site failures become more frequent.

The results of our performance experiments led to the following general observation: the optimum number of data replicas to provide the best performance in RTDBSs depends upon the fraction of update operations required by the application, the distribution of accessed data items, and the reliability of data sites. In general, as few as 2 or 3 copies appeared to be a good choice under the parameter ranges explored.

5. SUMMARY AND FUTURE WORK

It might be desirable to replicate data in a distributed real-time database system (RTDBS) due to certain benefits provided, such as high data availability and potentially improved read performance. However, under certain conditions, the overhead of synchronizing multiple copy updates can become a considerable factor in determining performance. Our work on replication in RTDBSs aimed to identify the conditions under which data replication can help real-time transactions satisfy their timing constraints. This chapter provides a summary of this work as well as a brief overview of the recent research in the field. There is a requirement for the development of new techniques to control data replication and to enhance the availability of replicated RTDBSs.

References

[1] Barbara, D., Garcia-Molina, H., "How Expensive is Data Replication? An Example", *International Conference on Distributed Computing Systems,* 263–268, 1982.

[2] Bernstein, P. A., Goodman, N., "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases", *ACM Transactions on Database Systems,* vol.9, 596–615, 1984.

[3] Biyabani, S. R., Stankovic, J. A., Ramamritham, K., "The Integration of Deadline and Criticalness in Hard Real-Time Scheduling", *Real-Time Systems Symposium,* 152–160, 1988.

[4] Carey, M. J., Livny, M., "Conflict Detection Tradeoffs for Replicated Data", ACM *Transactions on Database Systems,* vol. 16, 703–746, 1991.

[5] Ciciani, B., Dias, D. M., Yu, P. S., "Analysis of Replication in Distributed Database Systems", *IEEE Transactions on Knowledge and Data Engineering,* vol.2, 247–261, 1990.

[6] Garcia-Molina, H., "The Future of Data Replication", *Symposium on Reliable Distributed Systems,* 13–19, 1986.

[7] Garcia-Molina, H., Abbott, R. K., "Reliable Distributed Database Management", *Proceedings of the IEEE,* vol.75, 601–620, 1987.

[8] Lin, K. J., Lin, M. J., "Enhancing Availability in Distributed Real-Time Databases", *ACM SIGMOD Record,* vol.17, 34–43, 1988.

[9] Noe, J. D., Andreassian, A., "Effectiveness of Replication in Distributed Computer Networks", *International Conference on Distributed Computing Systems,* 508–513, 1987.

[10] Pu, C., Leff, A., "Replica Control in Distributed Systems: An Asynchronous Approach", *ACM SIGMOD International Conference on the Management of Data,* 377–386, 1991.

[11] Schlicting, R. D., Schneider, F. B., “Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems", *ACM Transactions on Computer Systems,* vol.1, 222–238, 1983.

[12] Son, S. H., Kouloumbis, S., *A Real-Time Synchronization Scheme for Replicated Data in Distributed Database Systems,* Technical Report, CS-TR-91-12, Department of Computer Science, University of Virginia, 1991.

[13] Son, S. H., Kouloumbis, S., “Replication Control for Distributed Real-Time Database Systems", *International Conference on Distributed Computing Systems,* 144–151, 1992.

[14] Son, S. H., Kouloumbis, S., “A Token-Based Synchronization Scheme for Distributed Real-Time Databases", *Information Systems,* vol. 18, 375–389, 1993.

[15] Son, S. H., Zhang, F., “Real-Time Replication Control for Distributed Database Systems: Algorithms and Their Performance", *International Conference on Database Systems for Advanced Applications,* 214–221, 1995.

[16] Ulusoy, Ö., “Processing Real-Time Transactions in a Replicated Database System", *Distributed and Parallel Databases,* vol.2, 405–436, 1994.

[17] Xiong, M., Ramamritham, K., Haritsa, J., Stankovic, J. A., “MIRROR: A State-Conscious Concurrency Control Protocol for Replicated Real-Time Databases", *IEEE International Workshop on E-Commerce and Web-based Information Systems,* 1999.