Chapter 15

DISTRIBUTED CONCURRENCY CONTROL

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

1. INTRODUCTION

Distributed databases fit more naturally in the decentralized structures of many real-time database system (RTDBS) applications that are inherently distributed (e.g., the stock market, banking, command and control systems, and airline reservation systems). Distributed database systems provide shared data access capabilities to transactions; i.e., a transaction is allowed to access data items stored at remote sites. While scheduling transactions in a distributed RTDBS, besides observing the timing constraints, it should also be provided that the global consistency of the distributed database is preserved as well as the local consistency at each data site. To achieve this goal, it is required to exchange messages that carry scheduling information between the sites where the data items to be accessed by an executing transaction reside. The communication delay introduced by message exchanges constitutes a substantial overhead for the response time of a distributed transaction. Thus, guaranteeing the response times of transactions (i.e., satisfying the timing constraints), is more difficult in a distributed RTDBS than that in a single-site RTDBS.

The results of a considerable amount of research devoted to various issues in *distributed* RTDBSs have appeared in the literature. These issues include concurrency control (e.g., [10], [11], [12], [13], [14], [15], [17], [18], [21], [23], [24]), deadline assignment (e.g., [8]), replication (e.g., [25]), nested transaction execution ([4], [26]), and commitment (e.g., [5], [22]). Among these issues, concurrency control in distributed RTDBSs is the main theme of this chapter. *Concurrency control* in database systems is used to control the interaction among concurrently executing transactions in order to maintain the consistency of the database [20]. In a distributed database system, a scheduler at each site is responsible for controlling concurrent accesses to data items stored at that site. Access requests of both local and remote transactions are ordered together on the basis of the concurrency control protocol being executed. The schedulers at all sites together constitute a distributed scheduler. Implementation of concurrency control protocols in a distributed RTDBS environment is difficult due to the conflicting requirements of meeting timing constraints and maintaining data consistency. Research efforts in this field has basicly focused on development and evaluation of protocols that meet soft/firm deadline requirements with the aim of minimizing the number of missed deadlines.

This chapter provides a brief overview of the recent work that has addressed the concurrency control problem in distributed RTDBSs. It also includes the relative performance results of several real-time concurrency control protocols in a distributed database environment. The protocols aim to maximize the satisfaction of real-time requirements while maintaining data consistency via enforcing serializability. Concurrency control protocols are different in the way timing constraints of transactions are involved in controlling concurrent accesses to shared data.

2. RESEARCH EFFORTS

In this section, we describe examples of work that has addressed the concurrency control issue in distributed RTDBSs. The amount of work done in this field is much less compared to the work on concurrency control in single-site RTDBSs.

In an early work on distributed real-time concurrency control, Sha et al. proposed a protocol to overcome the so called *priority inversion* problem, which is the result of blocking high priority transactions by lower priority ones [17], [18]. The protocol which is called *priority ceiling,* bounds the blocking time of high priority transactions to no more than one transaction execution time, and also eliminates the deadlock problem of the locking protocols. The 'priority ceiling' of a data item is defined as the priority of the highest priority transaction that may have a lock on that item. In order to obtain a lock on a data item, the protocol requires that a transaction T must have a priority strictly higher than the highest priority ceiling of data items locked by the transactions other than T. Otherwise, transaction T is blocked by the transaction which holds the lock on the data item of the highest priority ceiling. Performance of the priority ceiling protocol was examined in [19] using simulation. The results obtained revealed that performance of the protocol is not satisfactory when the database is not memory resident. However, a significant improvement was observed in the performance when intention I/O was used to prefetch data items accessed by transactions. Son and Chang investigated the performance of several different versions of the priority ceiling protocol [21]. They developed and used a prototype of a distributed RTDBS for the performance evaluation. The priority inversion problem was investigated by Huang et al. as well in a particular real-time environment where a two-phase locking concurrency control protocol was employed [6], [7].

Lam and Hung introduced two locking-based concurrency control protocols for distributed RTDBSs [11]. The first protocol, which is based on dynamic locking, uses the concept of cautious waiting to resolve lock conflicts among transactions. The performance of this protocol was shown to be better than some other protocols with different conflict resolution strategies. The second protocol proposed by the authors is based on static locking, assuming that lock requirements of transactions are known prior to their execution. It was shown that this protocol performs well for the systems where the majority of lock requests of transactions are on the remote data items. The authors provided three more static locking protocols in a more recent paper [12]. These protocols resolve lock conflicts by reserving the locks for high priority transactions. The protocols are different in their methods of reducing the blocking time of high priority transactions. The results of an extensive simulation study of the protocols presented in the paper show that the relative performance of the protocols depends on the degree of data contention that exists in the system. It was also observed that performance of the protocols is at a higher level when a centralized scheduler is employed.

An optimistic concurrency control protocol with dynamic adjustment of serialization order was adapted to distributed RTDBSs by Lam et al. [10]. With this protocol, it is aimed to take the advantage of dynamic adjustment of serialization order, while trying to minimize the overhead of that adjustment in a distributed environment. Among the factors that can lead to a substantial amount of overhead are the increased complexity of the dynamic adjustment, high communication message volume, and the possibility of distributed deadlocks during the validation of transaction executions. The performance results presented in the paper show that the proposed protocol is effective in reducing the overhead of dynamic adjustment of serialization order. The protocol involves a new deadlock-free distributed validation scheme and a new conflict resolution scheme.

Lee et al. proposed new priority assignment policies for distributed RTDBSs and investigated their impact on some typical real-time concurrency control protocols [16]. It was presented through performance experiments that optimistic concurrency control protocols are more affected by the priority assignment policies compared to locking-based protocols. It was also shown that considering both transaction deadlines and current data contention together in assigning transaction priorities provides the best performance among a variety of priority assignment techniques.

In a recent paper, Lam et al. introduced a protocol for resolving data conflicts among executing and committing real-time transactions in a distributed database system [14]. The proposed protocol integrates concurrency control and transaction commitment management. The protocol aims to reduce the impact of a committing transaction on the executing transactions which depend on it. A deadline-driven approach is used to overcome the dependency problem by reversing the the dependencies if the committing transaction has experienced a long commitment delay.

Lam et al. also studied the concurrency control problem of real-time database transactions in a mobile environment [13], [15]. It was argued by the authors that due to limited bandwidth and unpredictable behavior of mobile networks, results of the past research on distributed real-time concurrency control cannot be directly applicable to mobile distributed real-time database systems. The authors proposed a distributed real-time locking protocol which is based on the High Priority scheme [1] and the concept of similarity [9]. The restrictions introduced by mobile networks were considered in developing the new protocol.

Chen and Gruenwald [4], and Ulusoy [26] provided locking-based concurrency control protocols for distributed RTDBSs where transaction scheduling is based on the nested transaction model. Also provided in references [4] and [26] are the performance evaluation results of the protocols under different loading conditions.

In the rest of the chapter, we provide a presentation of our work on concurrency control in distributed RTDBSs, that was previously published in [23] and [24].

3. A DISTRIBUTED REAL-TIME DATABASE SYSTEM MODEL

This section provides the model of a distributed RTDBS that we used in evaluating real-time concurrency control protocols. In the distributed system model, a number of data sites are interconnected by a local communication network. There exists exactly one copy of each data item in the system. Each site contains a transaction manager, a scheduler, a resource manager, and a message server.

The transaction manager is responsible for generating transaction identifiers and assigning real-time priorities to transactions. Each transaction submitted to the system is associated with a real-time constraint in the form of a deadline. Transaction deadlines are *soft*; i.e., each transaction is executed to completion even if it misses its deadline. Each transaction is assigned a real-time priority based on the *earliest deadline first* priority assignment policy. A distributed transaction is modeled as a master process that executes at the originating site of the transaction and a collection of cohorts that execute at various sites where the required data items reside. The master process initiates the execution of each cohort process. Remote cohorts are initiated by sending a message to the appropriate sites. Each cohort is executed with its transaction's priority.

Each cohort performs one or more database operations on specified data items. 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.

Concurrent data access requests of the cohort processes at a site are controlled by the scheduler at that site. The scheduler orders the data accesses based on the concurrency control protocol executed. When a cohort completes all its accesses and processing requirements, it waits for the master process to initiate two-phase commit. Following the successful commitment of the distributed transaction, the cohort writes its updates, if any, into the local database.

Each site's resource manager is responsible for providing I/O service for reading/updating data items, and CPU service for processing data items, performing various concurrency control operations (e.g. conflict check, locking, etc.) and processing communication messages. Both CPU and I/O queues are organized on the basis of the cohorts' real-time priorities. Preemgtive-resume priority scheduling is used by the CPU at each site; a higher-priority process preempts a lower-priority process, and the lower-priority process can resume when there exists no higher-priority process waiting for the CPU. Communication messages are given higher priority at the CPU than other processing requests. Besides preemption, the CPU can be released by a cohort process as a result of lock conflict, for I/O, or communication to other data sites.

The message server at each site is responsible for sending/receiving messages to/from other sites. It listens on a well-known port, waiting for remote messages.

Reliability and recovery issues were not addressed in our work. We assumed a reliable system, in which no site failures or communication network failures occur.

3.1 MODEL PARAMETERS

The following parameters were used to specify the system configuration and workload. Parameter *nr_sites* represents the number of data sites in the distributed system. *db_size* specifies the number of data items stored in the database of a site, and *mem_size* specifies the number of data items that can be held in the main memory of a site. The mean interarrival time of transactions to each of the sites is determined by the parameter *iat.* The times between the arrival of transactions are exponentially distributed. The transaction workload consists of both query and update transactions. *tr_type_prob* specifies the update type probability. *access_mean* corresponds to the mean number of data items to be accessed by a transaction. Accesses are uniformly distributed among data sites. For each data access of an update transaction, the probability that the accessed data item will be updated is specified by the parameter *data_update_prob.* The CPU time for processing a data item is determined by the parameter *cpu_time,* while the time to access a data item on the disk is specified by *io_time.* It is assumed that each site has one CPU and one disk. For each new transaction, there exists an initial CPU cost of assigning a unique real-time priority to the transaction. This processing overhead is simulated by the parameter *pri_assign_cost.* Parameter *mes_proc_time* corresponds to the CPU time required to process a communication message prior to sending or after receiving the message. The communication delay of messages between the sites is assumed to be constant and specified by the parameter *comm_delay.*

slack_rate is the parameter used in assigning deadlines to new transactions. The slack time of a transaction is chosen randomly from an exponential distribution with a mean of *slack_rate* times the estimated processing time of the transaction.

4. REAL-TIME CONCURRENCY CONTROL PROTOCOLS

In this section, we briefly describe the concurrency control protocols whose relative performance was studied by using the distributed RTDBS model presented in the preceding section.

4.1 PRIORITY INHERITANCE PROTOCOL (PI)

'Priority inheritance' is one method proposed to overcome the problem of uncontrolled priority inversion [17]. This method makes sure that when a transaction blocks higher priority transactions, it is executed at the highest priority of the blocked transactions; in other words, it inherits the highest priority.

We implemented a distributed version of the priority inheritance (PI) protocol in our distributed RTDBS model as follows: When a cohort is blocked by a lower priority cohort, the latter inherits the priority of the former. Whenever a cohort of a transaction inherits a priority, the scheduler at the cohort's site notifies the transaction master process by sending a priority inheritance message, which contains the inherited priority. The master process then propagates this message to the sites of other cohorts belonging to the same transaction, so that the priority of the cohorts can be adjusted. When a cohort in the blocked state inherits a priority, that priority is also inherited by the blocking cohort (and its siblings) if it is higher than that of the blocking cohort.

4.2 HIGH PRIORITY PROTOCOL (HP)

This protocol prevents priority inversion by aborting low priority transactions whenever necessary [1]. High priority (HP) protocol was adapted to our model as follows: In the case of a data lock conflict, if the lock-holding cohort has higher priority than the priority of the cohort that is requesting the lock, the latter cohort is blocked. Otherwise, the lock-holding cohort is aborted and the lock is granted to the high priority lock-requesting cohort. Upon the abort of a cohort, a message is sent to the master process of the aborted cohort to restart the whole transaction. The master process notifies the schedulers at all relevant sites to cause the cohorts belonging to that transaction to abort. Since a high priority transaction is never blocked by a lower priority transaction, this protocol is deadlock-free.

4.3 DATA-PRIORITY-BASED LOCKING PROTOCOL (DP)

This subsection presents the concurrency control protocol that we introduced as an improvement over the 'priority ceiling' protocol [23]. Similar to the priority ceiling protocol (see Section 2), data-priority-based (DP) locking protocol is based on prioritizing data items; however, in ordering the access requests of the transactions on a data item *D,* it considers only the priority of *D* without requiring a knowledge of the priorities of all locked items.

Each data item carries a priority which is equal to the highest priority of all transactions in the system that include the data item in their access lists. When a new transaction arrives at the system, the priority of each data item to be accessed is updated if the item has a priority lower than that of the transaction. When a transaction commits and leaves the system, each data item that carries the priority of that transaction has its priority adjusted to that of the highest priority active transaction that is going to access that data item. Protocol DP assumes that transaction priorities are distinct.

Lock requests of transactions are handled by this protocol as follows: In order to obtain a lock on a data item *D,* the priority of a cohort *C* must be equal to the priority of *D.* Otherwise (if the priority of *C* is less than that of *D*), cohort *C* is blocked by the cohort that determines the current priority of *D.* Suppose that *C* has the same priority as *D,* but *D* has already been locked by a lower priority cohort *C'* before *C* has adjusted the priority of *D.* *C'* is aborted at the time *C* needs to lock *D.* When a cohort is aborted due to data conflict, the aborted cohort's master is notified to restart the whole transaction.

This protocol is deadlock-free since a high priority transaction is never blocked by lower priority transactions and no two transactions have the same priority.

5. SUMMARY OF PERFORMANCE EVALUATION RESULTS

Performance of the real-time concurrency control protocols was evaluated by simulating them on the distributed RTDBS model described in Section 3. Values of the system parameters were selected to provide a transaction load and data contention high enough to bring out the differences between the real-time performances of concurrency control protocols. The primary performance metric used in the evaluation of the protocols was *success_ratio;* i.e., the fraction of transactions that satisfy their deadlines.

When the performance of protocols was examined under various levels of transaction load by varying parameter *iat,* it was observed that the high priority (HP) protocol provides consistently better performance than the priority inheritance (PI) protocol under all load conditions tested. Resolving data conflicts by aborting low priority transactions seems to be more desirable in distributed RTDBSs than allowing the low-priority transactions to execute with inherited high priorities. Another observation was that data-priority-based (DP) locking protocol provides a substantial improvement in real-time performance over the other protocols, and the improvement becomes more as the transaction load in the system increases. Similar to HP, protocol DP does not allow the situation in which a high priority transaction can be blocked by a lower priority transaction. The number of transaction restarts is much lower than that of protocol HP since protocol DP restricts the possibility of transaction abort only to the following case. Between any two conflicting transactions, if the lower priority transaction accesses the data item before the higher priority transaction is submitted to the system (before the priority of the data item is adjusted by the entry of the higher priority transaction) the lower priority transaction is aborted when and if the higher priority transaction accesses the data item before the commitment of the lower priority transaction.

Some other experiments were conducted to evaluate the effect of various other system parameters on the protocols' performance. These parameters included *nr_sites* (number of data sites), *comm_delay* (message transfer delay), and *access_mean* (average number of data items accessed by a transaction). The results of each of these experiments can be found in [24]. The comparative performance of the protocols was not sensitive to varying the values of these parameters.

We also conducted various experiments by using a distributed transaction execution model which is different than that described in Section 3. In the modified model, the master process of a transaction spawns cohorts all together, and the cohorts are executed in parallel. The master process sends to each remote site a message containing an (implicit) request to spawn a cohort, and the list of all operations of the transaction to be executed at that site. The assumption here is that the operations performed by one cohort are independent of the results of the operations performed at the other sites. The sibling cohorts do not have to transfer information to each other. A cohort is said to be completed at a site when it has performed all its operations. The master process can start the two-phase commit protocol when it has received ‘completed’ messages from all its cohorts. Results of the experiments conducted by using this model can also be found in [24].

6. SUMMARY AND FUTURE WORK

Crucial to the concurrency control problem in real-time database systems (RTDBSs) is processing transactions within their deadlines, as well as maintaining the consistency of data accessed by transactions. It is difficult to implement concurrency control protocols in a RTDBS due to the conflicting requirements of meeting timing constraints and maintaining data consistency. The communication delay and other constraints of distributed systems make it more difficult to develop distributed concurrency control protocols that can offer good performance in terms of the satisfaction rate of timing constraints.

In this chapter, various approaches to the concurrency control problem in distributed RTDBSs are briefly described. As summarized in the chapter, research efforts in this field has basicly focused on development and evaluation of concurrency control protocols. Our work in this direction was chosen as an example to be presented in the chapter.

We believe that the amount of work done in this field is much less compared to the work on concurrency control in single-site RTDBSs and there is much room for further research and experimentation.

References

[1] Abbott R., Garcia-Molina H., “Scheduling Real-Time Transactions: A Performance Evaluation", *International Conference on Very Large Data Bases,* 1–12, 1988.

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

[3] Ceri S., Pelagatti G., *Distributed Databases: Principles and Systems,* McGraw-Hill, 1984.

[4] Chen, Y. W., Gruenwald, L., “Effects of Deadline Propagation on Scheduling Nested Transactions in Distributed Real-Time Database Systems", *Information Systems,* vol.21, 103–124, 1996.

[5] Gupta, R., Haritsa, J., Ramamritham, K., Seshadri, S., “Commit Processing in Distributed Real-Time Database Systems", *IEEE Real-Time Systems Symposium,* 1996.

[6] Huang, J., Stankovic, J.A., Ramamritham, K., Towsley, D., “On Using Priority Inheritance In Real-Time Databases", *Proceedings of the 12th Real-Time Systems Symposium,* 210–221, 1991.

[7] Huang, J., Stankovic, J.A., Ramamritham, K., Towsley, D., Purimetla, B., “Priority Inheritance in Soft Real-Time Databases", *Real-Time Systems,* vol.4, 243–268, 1992.

[8] Kao, B., Garcia-Molina, H., “Deadline Assignment in a Distributed Soft Real-Time System", *International Conference on Distributed Computing Systems,* 428–437, 1993.

[9] Kuo, T. W., Mok, A. K., “Application Semantics and Concurrency Control of Real-Time Data Intensive Applications", *IEEE Real-Time Systems Symposium,* 1992.

[10] Lam, K. W., Lee, V. C. S., Lam, K. Y., Hung, S. L., “Distributed Real-Time Optimistic Concurrency Control Protocol", *International Workshop on Parallel and Distributed Real-Time Systems,* 1996.

[11] Lam, K. Y., Hung, S. L., “Concurrency Control for Time-Constrained Transactions in Distributed Database Systems", *The Computer Journal,* vol.38, 704–716, 1995.

[12] Lam, K. Y., Hung, S. L., Son, S.H., “On Using Real-Time Static Locking Protocols for Distributed Real-Time Databases", *Real-Time Systems,* vol.13, 141–166, 1997.

[13] Lam, K. Y., Kuo, T. W., Law, G. C. K., Tsang, W. H., “A Similarity-Based Protocol for Concurrency Control in Mobile Distributed Real-Time Database Systems“, *International Workshop in Parallel and Distributed Real-time Systems,* 1999.

[14] Lam, K. Y., Pang, C. L., Son, S. H., Cao, J., “Resolving Executing-Committing Conflicts in Distributed Real-Time Database Systems", to appear in *The Computer Journal,* 2000.

[15] Lam, K. Y., Kuo, T. W., Tsang, W. H., Law, G. C. K., “Concurrency Control in Mobile Distributed Real-time Database Systems", to appear in *Information Systems,* 2000.

[16] Lee, V. C. S., Lam, K. Y., Kao, B. C. M., “Priority Scheduling of Transactions in Distributed Real-Time Databases", *Real-Time Systems,* vol. 16, 31–62, 1999.

[17] Sha, L., Rajkumar, R., Lehoczky, J., “Concurrency Control for Distributed Real-Time Databases” *ACM SZGMOD Record,* vol. 17, 82–98, 1988.

[18] Sha, L., Rajkumar, R., Lehoczky, J., “Priority Inheritance Protocols: An Approach to Real-Time Synchronization" *IEEE Transaction on Computers,* vol.39, 1175–1185, 1990.

[19] Sha, L., Rajkumar, R., Son, S.H., Chang, C.H., “A Real-Time Locking Protocol", *IEEE Transactions on Computers,* vol.40, 793–800, 1991.

[20] Silberschatz, A., Korth, H. F., Sudarshan, S., *Database System Concepts.* Third Edition, McGraw-Hill Computer Science Series, 1997.

[21] Son, S. H., Chang, C. H., “Performance Evaluation of Real-Time Locking Protocols Using a Distributed Software Prototyping Environment", *International Conference on Distributed Computing Systems,* 124–131, 1990.

[22] Soparkar, N., Levy, E., Korth, H. F., Silberschatz, A., *Adaptive Commitment for Real-Time Distributed Transactions,* Technical Report TR-92-15, Department of Computer Science, University of Texas at Austin, 1992.

[23] Ulusoy, Ö., Belford, G. G., “Real-Time Lock Based Concurrency Control in a Distributed Database System", *International Conference on Distributed Computing Systems,* 136–143, 1992.

[24] Ulusoy, Ö., “Lock-Based Concurrency Control in Distributed Real-Time Database Systems”, *Journal of Database Management* vol.4, 3–16, 1993.

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

[26] Ulusoy, Ö., “Transaction Processing in Distributed Active Real-Time Database Systems", *Journal of Systems and Software,* vol.42, 247–262, 1998.