Concurrent Control for Replicated Data in Distributed Real-Time Systems

Sang H. Son, Fengjie Zhang and Buhyun Hwang
University of Virginia

The design and implementation of time-critical schedulers for distributed replicated database systems must satisfy two major requirements: transactions must be able to meet the timing constraints associated with them, and mutual and internal consistency of replicated data must be preserved. In this paper, we present a replication control algorithm, which integrates real-time scheduling and replication control. The algorithm adopts a majority consensus scheme for replication control and attempts to balance the criticality of real-time transactions with the conflict resolution policies of that scheme. The algorithm employs epsilon-serializability (ESR), a correctness criterion which is less stringent than conventional one-copy-serializability, to guarantee the robustness of the scheme. The performance of the algorithm is evaluated and compared with a real-time token-based algorithm. The algorithm is extended to use the notion of quorum consensus, and the effects of read quorum on the performance is investigated.

In Real-time Distributed Database Systems (RTDDBS), transactions must be scheduled in such a way that they complete before their corresponding deadlines expire. In other words, the timeliness of results can be as important as their correctness (Yu et al, 1994). The main motivations behind the design and implementation of distributed systems include increased availability of data, improved fault tolerance, enhanced performance, and distributed workload (Son, 1987). However, the problems related to replication control in those systems, such as the preservation of mutual and internal consistency of replicated data, the concurrency control with inherent communication delays, become even more pressing and hard to solve when timing constraints are imposed on transactions.

The goal of scheduling in RTDDBS is twofold; to meet the timing constraints and to ensure that the replicas remain mutually consistent (Lin, 1989). Real-time task scheduling can be used to enforce timing constraints on transactions, while concurrency control is employed to maintain data consistency. Unfortunately, the integration of the two mechanisms is non trivial because of the trade-offs involved. Serializability may be too strong as a correctness criterion for concurrency control in database systems with timing constraints, for serializability severely limits concurrency. As a consequence, data consistency might be compromised to satisfy timing constraints.

In replication control methods, on the other hand (Pu & Leff, 1991a; Herlihy, 1986; Thomas, 1979), the objective is to provide a high degree of concurrency and thus faster average response time without violating data consistency (Bernstein & Goodman, 1984). Two different policies can be employed in order to synchronize concurrent data access of transactions and to ensure identical replica values: blocking transactions or aborting transactions. However, blocking may cause priority inversion when a high priority transaction is blocked by lower priority transactions. Aborting the very same lower priority transactions, though, wastes the work done by them. Thus, both policies have a negative effect on time-critical scheduling.

A less stringent, general-purpose consistency criterion is necessary. The new criterion should allow more real-time transactions to satisfy their timing constraints by temporarily sacrificing database consistency to some small degree. Epsilon-serializability (ESR) is such a correctness criterion, offering the possibility of maintaining mutual consistency of replicated data asynchronously (Pu & Leff, 1991b). Inconsistent data may be seen by certain query transactions, but data will eventually converge to a consistent state. In addition, the degree of inconsistency can be controlled so that the amount of the accumulated error (departure from consistency) in a
query can possibly be reduced to within a specified margin.

In this paper, we present a replication control algorithm that allows as many transactions as possible to meet their deadlines and at the same time maintain the consistency of the replicated data. The algorithm is based on majority consensus approach. Epsilon-serializability is employed as the correctness criterion to guarantee the consistency of the replicated database. Real-time scheduling features are then developed on top of this platform. We also present results from the performance study of our real-time majority consensus algorithm compared to a different real-time replication control algorithm based on token-based synchronization scheme (Son & Kouloumbis, 1993). Some conclusions were made as to which approach performs better under certain circumstances. In addition, we have extended the real-time majority consensus algorithm to be a more general replication control algorithm, namely the real-time majority quorum consensus algorithm. This quorum consensus algorithm can control the number of transactions meeting their deadlines by modifying their read and write quorums.

Database Model

Before presenting the theoretical background and feasible implementation of our real-time replication schemes, we first describe the organization of the underlying distributed database system. A simplified model of a distributed system is presented, and a short description of transaction processed in a replicated system is given.

Distributed System Environment

A distributed system consists of multiple autonomous computer systems (sites) connected via a communication network. Each site maintains a local database system. In order for transactions to be managed properly and for the results of their execution to be applied consistently to all replicas, a special process called transaction manager runs at each site. A scheduler process at each site implements the divergence control (as opposed to concurrency control) mechanism presented in the following section. Given the distributed nature and the increased communication burden of such a database system, a message server process runs at each site and take care of the communication protocols between its site and all others. Data managers are low-level processes, running one per site, that manage the local database (Son, 1992).

The smallest unit of data accessible to the user is called data object. In distributed database systems with replicated data objects, a logical data object is represented by a set of one or more replicated physical data objects. In a particular system, a physical data object might be a file, a page or a record. We assume that the database is fully replicated at all sites.

For the majority consensus algorithm, each data object copy is a read-write copy. It consists of a data value and a timestamp. The time-stamp of a data object reflects the time-stamp of the transaction that last updates the data object. Time-stamps are used in the voting procedure and the application of updates to the databases. They are also critical for insuring mutual consistency of all the databases and the internal consistency of each local database. We do not assume a global clock in the system. Local clocks can be synchronized using Lamport’s clock rules (Lamport, 1978).

Transactions

A transaction is a sequence of operations that takes the database from a consistent state to another consistent state. It represents a complete and correct computation. Two types of transactions are allowed in our environment: query transactions and update transactions. Query transactions consist only of read operations that access data objects and return their values to the user. Thus, query transactions do not modify the database state. Update transactions consist of both read and write operations.

Transactions have their time-stamps constructed by adding 1 to the greater of either the current time or the highest time-stamp of their base variables. Base variables are a joint set of their read-sets and write-sets. We assume that for each transaction, the write-set is a subset of the read-set. Therefore a transaction must first read a data object before it writes to it.

In majority consensus approach (Thomas, 1979), transactions first read all the data objects in the read-set from its local database, and then go through the voting procedure. Having received a vote request from a transaction, the site manager first checks to see if all the base variables of the transaction are up-to-date. If all the base variables are current, it will vote OK to accept the request. Otherwise, it will reject the request. A transaction can only abort after it has been rejected at one site. The currency of a data object is checked by comparing the time-stamp of the data object in the request’s read-set with the time-stamp of the data object in the local database. If these two time-stamps are the same, it means that the transaction has read the current data. If the object in the local database has a larger time-stamp than the one supplied by the request, the site manager can conclude that the transaction has read an outdated value, therefore it must be aborted. If the time-stamp supplied by the request is more current, the request is made to wait because this means that the update of the data object has not yet been applied at this location.

Two transactions conflict if the read-set of one transaction intersects with the write-set of the other transaction. During the voting process, if the transaction asking for the vote conflicts with a pending transaction, the scheduler will vote either PASS or defer the request based on the transaction’s priority. Each transaction is assigned a priority, possibly based on its time-stamp. The transaction will be deferred if it conflicts with a lower priority transaction.