Chapter LXXXI
A Survey of Approaches to Database Replication

F. D. Muñoz-Escoí
Universidad Politécnica de Valencia, Spain

H. Decker
Universidad Politécnica de Valencia, Spain

J. E. Armendáriz
Universidad Politécnica de Valencia, Spain

J. R. González de Mendívil
Universidad Politécnica de Valencia, Spain

INTRODUCTION

Databases are replicated in order to get two complementary features: performance improvement and *high availability*. Performance can be improved when a database is replicated since each replica can serve read-only accesses without requiring any coordination with the rest of replicas. Thus, when most of the application accesses to the data are read-only, they can be served locally and without preventing accesses in the same or other replicas. Moreover, with a careful management, the failure of one or more replicas does not compromise the availability of the database.

Initially, database replication management was decomposed into two tasks: concurrency control and replica control, usually solved by different protocols. The solutions of the non-replicated domain were evolved into distributed concurrency control protocols (Bernstein & Goodman, 1981), taking as their base either the two-phase-locking (2PL) or some timestamp-ordering protocol. Replica con-

trol management was based on *voting techniques* (Gifford, 1979). These voting techniques assign a given number of votes to each replica, usually one, and require that each read access collects a read *quorum* (i.e., "r" votes) and each write access a write *quorum* (i.e., "w" votes). The database must assign version numbers to the items being replicated, and the values of "r" and "w" must ensure that r+w is greater than the total number of votes, and that "w" is greater than a half of the amount of votes. Thus, it can be guaranteed that each access to the data reaches at least one copy with the latest version number for each item. This ensured consistency, but the communication costs introduced by these techniques were high.

Voting replica-control protocols were still used in the next decade, boosting their features and including also management for system partition handling when dynamic voting approaches were included (Jajodia & Mutchler, 1990).

However, replication management is not so easy to achieve when both concurrency and replica control are merged, since what the replica control protocols do for ensuring consistency has to be accepted by the concurrency control being used. Deadlocks and transaction abortions are common when both protocols are joined. So, it seems adequate to find better solutions for this global management, i.e., replication protocols that consider both concurrency and replica controls. A first example of this combined technique is (Thomas, 1979), where a special kind of *voting technique* is combined with timestamp-based concurrency control. However, his solution still relies on simple communication primitives, and is not efficient enough, both in terms of response time and abortion rate. Note that efficient total order broadcast protocols were not produced until the middle eighties (Birman & Joseph, 1987), and they could not be used in these first stages.

So, new replication techniques were introduced for databases, as an evolution of the process replication approaches found in distributed systems. Thus, depending on the criteria being used, several classifications are possible (Gray et al., 1996; Wiesmann et al., 2000). The proposal of Wiesmann & Schiper (2005), distinguishing five different techniques (active, weak-voting, certification-based, primary copy and lazy replication) will be followed here. In all these techniques, the protocols need a reliable total order broadcast in order to propagate the transaction updates. This is the regular way for achieving replica consistency in any distributed application.

ACTIVE REPLICATION

In the *active replication* technique, the client initially submits a transaction request to one of the replicas. Such *delegate replica* broadcasts the request to all replicas and all of them process the transaction from its start. Note that different transactions can use different *delegate replicas*. This technique requires complete determinism in the execution of a transaction, since otherwise the resulting state could be different among replicas. In order to easily develop this model, transactions can be implemented as stored procedures. Note that all transaction operations or parameters should be known before such transaction is started, being propagated in its starting broadcast.

The main advantage of this technique is that the support for executing transactions can be the same in both *delegate* and non-delegate *replicas*. On the other hand, its disadvantages consist of (1) requiring determinism in the transaction code, and (2) compelling read operations to be executed in all replicas, losing the possibility of balancing reads among replicas, and thus compromising one of the best performance improvements of database replication.

This technique is a direct translation of the *active replication* model for distributed systems. It has not been directly used for database replication, but there are some adaptations that have eliminated several of its intrinsic problems, improving its performance and behavior. One of