Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks

Robert N. Watson
robert@fledge.watson.org

March 2, 1999

Abstract

Adya95: Adya, et al. *Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks.* ACM Int. Conf. on Management of Data (SIGMOD), May 1995.

1 Introduction

Adya95 describes an algorithm for *efficient optimistic concurrency control* for use in distributed databases with caching clients. The algorithm provides *serializability* and *external consistency.* Their goals include a low memory overhead (no per-object overhead), and high performance. They assume the availability of *loosely synchronized clocks*, and low to moderate contention.

For the purposes of description and implementation, they choose the *Thor*, a distributed object-oriented database system. In Thor, persistent storage of objects occurs on a home server or *owner.* The clients, or *front ends*, may cache whole objects along with associated meta-data. The servers track cached copies in an object's *cached set.* Servers may then send *invalidation messages* to clients indicating that their cached copies are no longer valid. On receiving an invalidation, the client aborts any in-progress transactions. This lowers the load on servers by preventing later aborts on the server; clients ACK invalidations so that the related server can update its cached set.

When a transaction is ready to commit on a client, it must be submitted to the server pool for validation. For transactions using objects only on a single server, the transaction is submitted to that server. For transactions using objects on multiple servers, a single server is selected as *coordinator.* A *2-phase commit protocol* is then used to other servers or *participants.* When a transaction is submitted by a front end, both *validation information*, the identities of objects as well as the access types, and *installation information*, the modified data, are sent.

Time synchronization allows Thor to determine an *a priori* total ordering for transactions. This allows it to meet the goals, serializability and external consistency. NTP is used for synchronization; each transaction is assigned a time-stamp in the form $T.ts =< time, ServerID >$; for ordering, first the time field is used, and then a unique server identification number (such as an IP address) for tie-breaking. The time-stamp is assigned by the transaction coordinator in the event of a multi-server transaction.

During the validation phase of the 2-phase commit, a newly submitted transaction is compared against all transactions that have already been successfully validated: this is referred to as *backwards validation*. A transaction *conflicts* with another transaction if either one has modified an object that the other reads or modifies. Each participant in the validation must check to ensure that any of its objects in the transaction pair can be serialized in the time-stamp order. Because any existing transactions that have already been validated cannot be aborted, the new transaction must be aborted in the event that a conflict cannot be resolved; this will prevent the new transaction from violating the total ordering based on time-stamps. In a multi-server transaction, this is performed by returning a NACK to the coordinator, who then aborts the transaction.

On a successful validation, the relevant information is stored in the *validation queue*, or VQ. Checks for a new transaction T occur against all transactions S in the VQ. Because of loose synchronization or delays, it is possible that T might be aborted even if it has an earlier time-stamp than S; however, this is infrequent with time synchronization and low contention. The explicit checks include the *later-conflict* check, where T must not be committed if $T.ts < S.ts$ and T read any object that S modified, and the *version check*, which aborts any transaction T where $S.ts < T.ts$, where S modified x and T read x and either 1. S validated but not committed: abort T as it cannot have seen S's results yet, or 2. S committed: abort T if it is not based on the current version of x. While these checks are conservative, they result in few spurious aborts.

Because validation queues grow with each transaction, but storage is finite, validation records cannot be retained indefinitely. However, they are required for transactions in progress. The approach taken is to maintain a *threshold* for how long old records may be retained. Records are retained for all uncommitted read-write transactions, and for all transactions with a time-stamp above a threshold. If information is not available to validate a new transaction because it has been truncated from the queue, then the transaction is aborted and may be retried. A number of approaches to the setting the time threshold are mentioned, including basing it on message delay and maximum clock skew, and discarding data associated with a transaction once it commits.

In the event of a crash, the server must ensure that transactions validated after the crash are serializable with those before. To this end, validation information is logged. However, as read-only transactions are not kept in the VQ, they do not appear in the logs. Instead, a *stable threshold* is maintained, which is always $> T.ts$ where T is the latest transaction ever validated on the server, intermittently updated. No matter how this is handled, failures may cause some transactions to abort after recovery, especially for fast recovery. This is OK because this is optimistic concurrency control.

Adya95 compares their algorithm with existing algorithms, which largely involve atomic multicast (expensive) or logical close (complex). Unlike existing systems, the algorithm cannot cause deadlocks, and does not consume much space. In a performance comparison with ACBL (Adaptive Callback Locking, another optimistic scheme), the new algorithm performs very favorably in almost all loads. In comparison with non-optimistic schemes, their algorithm causes problems only in extremely high contention; they suggest a hybrid scheme switching between the two as contention increases.

Slides for the associated presentation available at
http://www.watson.org/~robert/15-712/