In Search for an Efficient Real-Time Atomic Commit Protocol*

Yousef J. Al-Houmaily
Dept. of Electrical Engineering
University of Pittsburgh
Pittsburgh, PA 15261
yjast1+@pitt.edu

Panos K. Chrysanthis
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260
panos@cs.pitt.edu

Abstract

The purpose of this paper is to report on the first step in our quest for an efficient atomic commit protocol in real-time databases. This includes the development of *real-time implicit yes-vote* (RT-IYV), a new real-time atomic commit protocol. In contrast to other real-time commit protocols that provide for semantic atomicity, RT-IYV is designed to ensure the traditional notion of transaction atomicity. To illustrate its performance advantages, we compare RT-IYV with the recently proposed *optimistic commit protocol* which is also designed to support the standard transaction atomicity in real-time databases.

1 Introduction

An *atomic commit protocol* provides the only mean to ensure the traditional notion of *atomicity* of transactions in any distributed database system. The *two-phase commit* protocol [3] is the simplest and most widely used atomic commit protocol.

The two-phase commit protocol (2PC) consumes a substantial amount of a transaction's execution time due to the cost associated with its coordination messages and forced log writes to stable storage required for recovery. For this reason, a number of 2PC variants and optimizations have been proposed to reduce the cost of 2PC in different environments (e.g., [5, 4, 6, 7, 1]). One such variant is the *implicit yes–vote* protocol (IYV) that we have proposed in the context of gigabit–networked distributed databases [1, 2] which reduces transaction execution time by exploiting the network characteristics. Specifically, IYV (1) eliminates the voting phase from 2PC hence, reducing the number of sequential coordination messages and forced log writes during normal processing, and (2) supports transactions' forward recovery hence, enabling partially executed transactions to resume their execution after a failure.

In *real-time distributed database systems* (RT-DDBSs), semantics-based commit protocols that eliminate the uncertainty period to terminate a transaction and the blocking effects that might lead to the *priority inversion* problem [8], have been proposed as an alternative to 2PC for the commitment of real-time distributed transactions [9, 10]. In these protocols, a participant is allowed to unilaterally commit a real-time transaction and release the resources held by the transaction. If the final decision is to abort the transaction, *compensation* is used to semantically obliterate the effects of the aborted transaction. That is, these protocols do not ensure the traditional notion of transaction atomicity but they provide for *semantic atomicity* of the real-time transactions. Since compensation has limited applicability and in some real-time database applications it is necessary to ensure transaction atomicity, the *optimistic commit protocol* (OPT), a new real-time 2PC variant, has been recently proposed in [4]. In this paper, we investigate the applicability of IYV in the context of RT-DDBSs, due to its efficiency appealing characteristics, as part of our search for an efficient atomic commit protocol for real-time distributed transactions.

The rest of this paper is structured as follows. In the next section, we discuss the transaction model in RT-DDBSs and the priority inversion phenomenon. In Section 3, we present our RT-IYV while in Section 3.1, we compare the RT-IYV advantages over OPT. Section 4 concludes this paper with our future work in this direction.

2 Transactions in RT-DDBSs

A *real-time distributed transaction* (RTDT) is a traditional transaction [3] that is associated with a *firm deadline* and accesses data stored at different sites. In particular, in this paper, a RTDT adheres to the traditional transaction properties of *atomicity* and *serializability*. These properties are typically ensured by employing *strict two-phase locking* (S2PL) for concurrency control combined with *write-ahead logging* (WAL) for recovery [3].

In addition, each RTDT is associated with a *priority* that reflects its significance and is used to determine how resource conflicts between transactions should be resolved. Specifically, when a high priority RTDT re-

*Supported in part by the Saudi Arabian Government through a graduate student scholarship and National Science Foundation under grants IRI-9210588 and IRI-95020091.

quests a resource held by a low priority one, the holding RTDT is preempted to allow access to the resource and to avoid the, undesirable, *priority inversion* phenomenon [8]. Without preemption, this phenomenon causes high priority RTDTs to miss their deadlines and to abort due to their contention over the system's resources with low priority transactions. An alternative to preemption is *priority inheritance* [8] in which a low priority RTDT inherits a higher priority from its awaiting transaction in order to finish its execution faster, thereby, allowing the high priority RTDT to access the requested resources sooner and to commit by its deadline.

In a real-time distributed database, when a RTDT finishes its execution before its deadline has expired, its *coordinator* (i.e., the site where the transaction has been submitted) initiates 2PC which consists of a *voting* phase and a *decision* phase. When a participant votes `Yes` during the voting phase, the transaction enters its *prepared to commit* state at the participant's site. While a RTDT is in a *prepared to commit* state (uncertainty period), the participant can neither commit nor abort the transaction and it has to wait until it receives the final decision from the coordinator. If a higher priority RTDT requests a resource held by a *prepared to commit* low priority RTDT, the high priority transaction has to wait until the low priority RTDT is finally committed or aborted by its coordinator and in this case, priority inversion is inevitable.

The recently proposed OPT is a 2PC variant that attempts to alleviate priority inversion in 2PC. In OPT, a high priority RTDT is allowed to *borrow* (i.e., access) data items held by a *prepared to commit* low priority RTDT under the assumption that the latter transaction will most probably going to commit, and taking the risk of aborting both transactions if the low priority RTDT aborts.

3 Real-Time Implicit Yes-Vote

The *real-time Implicit Yes-Vote* protocol (RT-IYV) is based on the assumptions that each site employs (1) a *strict two-phase locking* protocol (S2PL) for concurrency control that takes into consideration the priority of transactions and (2) *physical page–level write–ahead logging* (WAL) [3].

Based on the first assumption, it is not possible for a participant in a transaction's execution to abort the transaction due to a deadlock or serializability violation once all the operations received by the participant have been successfully executed and acknowledged. Hence, instead of initiating commit processing at the end of transactions, which is the case in 2PC and OPT, in RT-IYV, commit processing is overlapped with the execution of the transactions' operations.

Specifically, when the coordinator of a RTDT receives

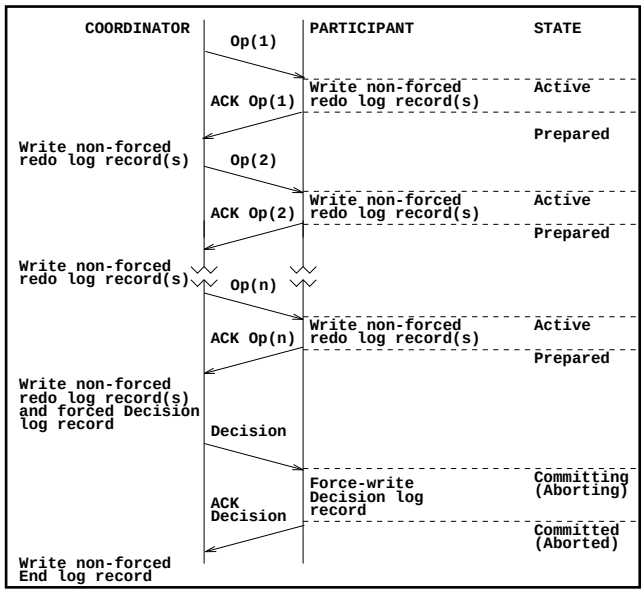

Figure 1: The RT-IYV protocol.

an acknowledgment message (ACK) from a participant pertaining to the transaction's operation, the ACK is *implicitly* interpreted to mean that the RTDT is in a *prepared to commit* state at the participant as shown in Figure 1. When the participant receives a new operation for execution, the RTDT becomes active again. While the RTDT is active, it can be aborted, for example, if it causes a deadlock or priority inversion. If the RTDT is aborted, the participant responds with a *negative acknowledgment* message (NACK). When all the operations pertaining to the RTDT are executed and acknowledged by their perspective participants, the coordinator commits the transaction if its deadline has not expired. Otherwise, the coordinator aborts the transaction. In either case, the coordinator propagates its decision to the participants and waits for their acknowledgments as it is the case in 2PC. Thus, in RT-IYV the (explicit) *voting* phase of 2PC, which polls the votes of the participants, is eliminated by overlapping it with the execution of operations while the *decision* phase remains the same as in 2PC.

To ensure correct recovery after a failure, each participant in RT-IYV is required to include the *redo* log records generated during the execution of an operation with their corresponding *log sequence numbers* (LSNs) in the operation's ACK. Each participant also includes the *read* locks acquired during the execution of an operation in the ACK in order to support *forward recovery*. In this way, each coordinator has a partial image of the state of each of its participants in the system. After a crash, a participant repairs its log and reconstructs the state of its database, that includes its lock table, using

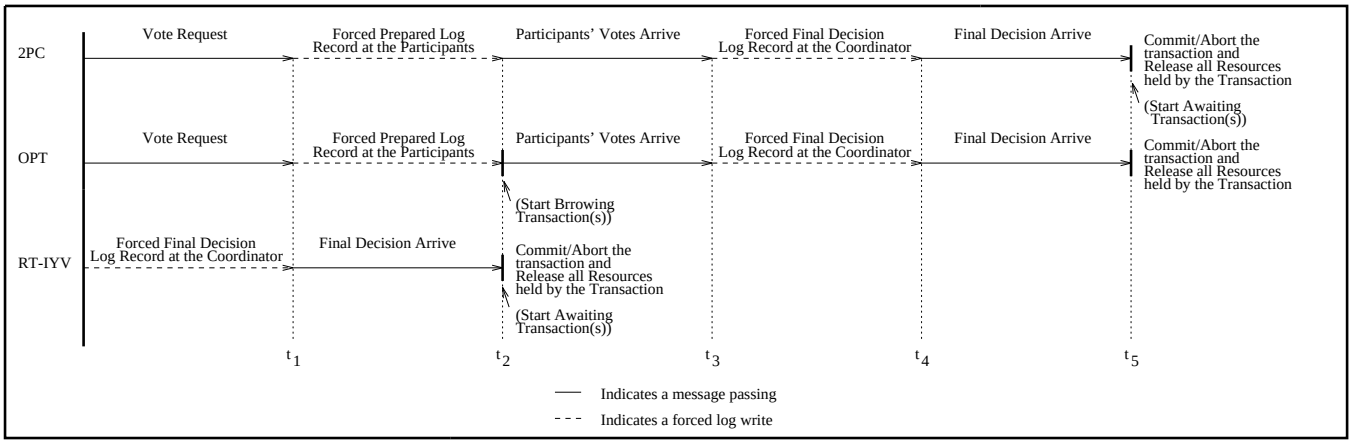

Figure 2: The sequential coordination messages and forced log writes during normal processing.

the partial images of its state stored at the coordinators. Specifically, a failed participant redoes the effects of committed RTDTs and those RTDTs that are still active in the system and their deadlines have not expired yet. Thus, in RT-IYV, a participant failure does not necessarily leads to the abort of those RTDTs that are still in-progress at other participants by the time the participant has recovered, as opposed to 2PC and OPT. This also has the implication that a coordinator, in RT-IYV, can commit a RTDT despite participants' site and communication failures as long as the transaction's deadline has not expired by the time it has finished its execution.

3.1 RT-IYV Performance Advantages

Consider Figure 2 which graphically illustrates the sequence of coordination messages and forced log writes that are involved in 2PC, OPT and RT-IYV to reach a decision point and to release the resources held at the participants for the commit as well as the abort case.

2PC, OPT and RT-IYV have different decision points. The decision point from a coordinator's perspective is at t_1 in RT-IYV whereas it is at t_4 in 2PC and OPT. On the other hand, the decision point from a participant's perspective is at t_2 in RT-IYV whereas it is at t_5 in 2PC and OPT.

Although RT-IYV and OPT have different decision points, both RT-IYV and OPT allow high priority RTDTs to access data items held by low priority RTDTs at the participant sites within the same amount of time (i.e., at t_2 in Figure 2). However, OPT supports early access to data at the risk of cascading aborts. In OPT, high priority RTDTs can borrow data from low priority ones while the low priority RTDTs are in their prepared states and are aborted in the event that the lending RTDTs are finally aborted. On the other hand, in RT-IYV, a high priority RTDT is never aborted due to an aborting low priority RTDT. In RT-IYV, data is released, rather than lent, to a high priority RTDT when the lower priority RTDT reaches its commit point at the participant which is the same point at which data is lent in OPT. For the same reason, RT-IYV does not have to employ deferred updates to prevent overwriting of uncommitted data whereas OPT has to support deferred updates if a RTDT is allowed to perform write operations on borrowed data.

RT-IYV, in general, involves larger size of messages and extra logging activities at the coordinators. However, given the characteristics of modern high speed networks, the message size is not an issue. Similarly, given the availability of main memory in modern computers and that the extra logging does not involve access to stable storage, the overhead incurred by the extra logging is negligible.

In 2PC and OPT, when a participating site fails, all the RTDT executing at the failed participant are aborted by their coordinators. As discussed in the previous section, this is not necessarily the case in RT-IYV. In RT-IYV, a coordinator has the flexibility to decide between aborting, committing or forward recovering and continuing a RTDT on a failed participant. Thus, RT-IYV provides the potential of more RTDTs to commit by their deadlines.

OPT attempts to increase the transactions success ratio by incorporating two optimizations. The first one, called *active abort*, is an unsolicited `No` vote optimization in which a participant asynchronously votes `No` once the participant decides to abort a RTDT and without waiting for the prepare request from the coordinator of the RTDT. In RT-IYV, there is no need for active abort since its semantics are captured by means of the negative acknowledgment (NACK). In RT-IYV, a participant can abort a RTDT for any reason including priority inversion and the expiration of the transaction's deadline only if the participant has a pending operation's acknowledg-

ment. If a participant aborts a RTDT, it synchronously notifies the transaction's coordinator about the abort with a NACK.

The second optimization used in OPT is the *deadline silence* in which the coordinator before the initiation of OPT, and the participants before entering the *prepared to commit* state, can silently abort a RTDT without any further communication in the event that the RTDT's deadline expires. This optimization has limited applicability in RT-IYV with no obvious significant benefits. In RT-IYV, a RTDT at a participant is either in its *prepared to commit* state or has a pending acknowledgment for the coordinator. Only in the latter case, if a participant decides to abort the RTDT because its deadline has expired, it can choose not to send a NACK. However, still the coordinator, not knowing which participants are in a prepared to commit state, has to send an abort message to all the participants as soon as it detects the expiration of the RTDT's deadline.

4 Conclusion

The purpose of this paper was to present the first step in our quest for an efficient atomic commit protocol in real-time databases. This included the development of RT-IYV (*real-time implicit yes-vote*), a new real-time atomic commit protocol. To illustrate its performance advantages, we compared RT-IYV with the recently proposed OPT (*optimistic commit protocol*) for real-time databases.

Our next step is to extend our simulator, developed for the empirical evaluation of the best 2PC variants in conjunction with the read-only optimization, to include the modeling of real-time transactions. This will allow us to better establish the relative performance advantages of RT-IYV compared with OPT.

References

[1] Y. J. Al-Houmaily and P. K. Chrysanthis, "Two-Phase Commit in Gigabit-Networked Distributed Databases," *Proc. of the 8th Int'l Conf. on Parallel and Distributed Computing Systems*, pp. 554–560, Sept. 1995.

[2] Y. J. Al-Houmaily and P. K. Chrysanthis, "The Implicit Yes-Vote Commit Protocol with Delegation of Commitment," *Proc. of the 9th Int'l Conf. on Parallel and Distributed Computing Systems*, pp. 804–810, Sept. 1996.

[3] P. A. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Reading, MA, 1987.

[4] R. Gupta, J. Haritsa, K. Ramamritham and S. Seshadri, "Commit Processing in Distributed Real-Time Database Systems," *Proc. of the 17th IEEE Real-Time Systems Symposium*, Dec. 1996.

[5] C. Mohan, B. Lindsay and R. Obermarck, "Transaction Management in the R^* Distributed Database Management System," *ACM TODS*, 11(4):378–596, Dec. 1986.

[6] G. Samaras, K. Britton, A. Citron and C. Mohan, "Two-Phase Commit Optimizations and Tradeoffs in the Commercial Environment," *Proc. of the 9th Int'l Conf. on Data Engineering*, pp. 520-529, Feb. 1993.

[7] G. Samaras, K. Britton, A. Citron and C. Mohan, "Two-Phase Commit Optimizations in a Commercial Distributed Environment," *Distributed and Parallel Databases*, 3(4):325–360, Oct. 1995.

[8] L. Sha, R. Rajkumar and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *Technical Report CMU-CS-87-181*, Carnegie Mellon University.

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

[10] Y. Yoon, "Transaction Scheduling and Commit Processing for Real-Time Distributed Database Systems," *Ph.D. Thesis*, Korea Advanced Institute of Science and Technology, May 1994.