Adjourn State Concurrency Control Avoiding Time-Out Problems in Atomic Commit Protocols

Sebastian Obermeier #1, Stefan Böttcher #2, Martin Hett #3, Panos K. Chrysanthis *4, George Samaras +5

#University of Paderborn, Faculty 5 - EIM
33102 Paderborn, Germany
1so@upb.de
2mh1108@upb.de
3stb@upb.de

*University of Pittsburgh, Department of Computer Science
Pittsburgh, PA 15260, USA
4panos@cs.pitt.edu

+University of Cyprus, Department of Computer Science
CY-1678 Nicosia, Cyprus
4cssamara@cs.ucy.ac.cy

***Abstract*— The use of atomic commit protocols in mobile ad-hoc networks involves difficulties in setting up reasonable time-outs for aborting a pending distributed transaction. This paper presents the non-blocking Adjourn State, a concurrency control modification which makes time-outs in an atomic commit protocol for aborting a transaction unnecessary. Further, it enhances concurrency among transactions performing conflicting accesses to resources used by completed distributed transactions waiting for the commit protocol to be initiated.**

I. Introduction

As mobile devices get ubiquitous and interact cooperatively, the management of their shared data also becomes increasingly important. Within fixed wired networks, atomic commit-protocols such as Two-phase Commit protocol (2PC) [6] or Three-phase Commit protocol (3PC) [11], and their variants (e.g. [7], [10]), ensure the atomic execution of distributed transactions. Most of these techniques and protocols rely on time-outs to detect and handle failures. In the context of mobile ad-hoc networks where disconnection times are unforeseeable, it is extremely difficult to set up reasonable time-outs, for example, for aborting a transaction on a mobile host when its commit coordinator does not respond immediately during the execution of a 2PC or 3PC. Hence, the use of standard lock-based concurrency control techniques and atomic commit protocols in mobile ad-hoc environments may lead to unbounded and unpredictable delays due to blocking. This observation has motivated our search for techniques and protocols that are more flexible and can more effectively deal with the much more enhanced failure model of mobile environments.

A. Contributions

We present the "Adjourn State", a non-blocking state that allows a transaction to execute operations which conflict with those of another distributed transaction waiting for its coordinator's voteRequest message to initiate commit preparation. In contrast to the traditional *wait state*, the Adjourn State is based on optimistic concurrency control and shows the following advantages:

- It does not require the set-up of transaction time-outs.
- It does not block concurrent transactions.
- It is compatible with [3] that unlike traditional atomic commit protocols does not require an abort of the global transaction if a conflict is detected, but only requires a partial redo of the local transaction.

II. System Model

We are assuming a set of mobile devices or hosts (MH) with local databases. Each MH shares its data via web-services. Thus, applications on one MH can access data on another MH by invoking a web-service. Each web-service request initiates one or more sub-transactions against the local database, or it can spawn another sub-transaction on another MH.

A. Transaction Model

We assume that the sub-transactions comprising a web-service obey the following transaction execution model (c.f. Figure 1), in which transactions are committed using an atomic commit protocol.

After a MH has received a (sub-)transaction T from the initiator, the MH processes the transaction's reads and performs all write operations within a private transaction space (read phase). During the execution of the read phase of a sub-transaction T_i, T_i may also invoke other sub-transactions T_j, but the MH must store the invocation parameters for T_j. After a (sub-)transaction has finished its read phase, it goes through a validation phase at its local MH. If the validation succeeds, the MH sends the (sub-)transaction's result along with a list of the invoked sub-transactions to its initiator. Otherwise, it sends an abort message. Details of the validation phase are explained in the following Section II-B. When the initiator of T receives all the results and must commit the distributed transaction it invokes an atomic commit protocol by notifying a commit coordinator instance about all participating sub-transactions.

In the case of 2PC, the commit coordinator sends a voteRequest message to each MH involved in the distributed

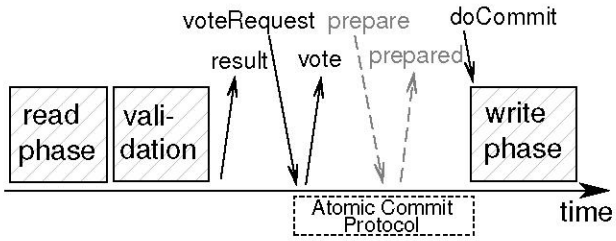

Fig. 1. Transaction Model

transaction T. Each MH that received this voteRequest replies by sending the vote message on the transaction. After the coordinator has received all the vote messages and none of them was for abort, the coordinator issues the doCommit command and each MH executes the write phase of the transaction. The coordinator may also abort the transaction whenever a participant does not respond in a given time interval or voted for abort.

When 3PC is used, additional prepare and prepared messages, illustrated by dashed gray lines in Figure 1, are exchanged before the coordinator sends the final commit command.

A MH itself may abort a transaction as long as the vote message has not been sent to the coordinator, e.g., if the coordinator does not request the vote within a certain period of time.[1]

B. Local Concurrency Control by Backward Validation

MHs use optimistic concurrency control [8], more precisely *backward oriented optimistic concurrency control* with parallel validation. In detail, a local sub-transaction T_O is older than a local sub-transaction T_V running on the same database, if T_O starts its *validation phase* before T_V does.

A transaction T_V validates to *true*, if one of the following three conditions holds for each older transaction T_O:

1) T_O has completed its write phase before T_V has started.
2) T_O has completed its write phase before T_V started its validation phase, and $\mathsf{readset}(T_V) \cap \mathsf{writeset}(T_O) = \varnothing$.
3) T_O has not finished its write phase before T_V has started the validation, and $(\mathsf{readset}(T_V) \cup \mathsf{writeset}(T_V)) \cap \mathsf{writeset}(T_O) = \varnothing$.

C. Problem Description

Regardless of which concrete atomic commit protocol is used, the following problem occurs when the MH still waits for a voteRequest on a transaction T_O, but the commit coordinator is not reachable anymore, i.e., whenever a transaction T_O has successfully validated and the result message was sent, but the MH does not receive the voteRequest message. Then, the last validation condition (3) will be checked by each newer parallel transaction T_V, and this condition prevents conflicting transactions T_V from being successfully validated for the following reason. Every transaction T_V that is started while T_O waits for its coordinator and that wants to access an object that T_O intends to write, validates to false and will be aborted. In other words, any delay in the commit phase of T_O has a blocking effect on concurrent conflicting transactions T_V. To solve this problem, [11] has introduced time-outs after which the MH aborts the transaction T_O if it is still allowed to do so, i.e., if it has not sent its vote message.

However, especially in mobile networks, the question arises: "What is a reasonable time-out after which the MH should abort the transaction T_O if it is still allowed to do so?". If the time-out is too large, it prevents concurrent and conflicting transactions T_N from a successful validation, since T_N will not pass the validation phase successfully due to the pending transaction T_O. If the time-out is too short, T_O may be unnecessarily aborted, e.g., when the delay is caused by the network or when the duration of the validation phase differs for the MHs participating in the global transaction. Determining a reasonable time-out is difficult since it involves not only knowledge about the network conditions, e.g., device movement, message delivery times, message loss rates, etc., it must also consider the device's computing power and CPU utilization, and the varying duration of the validation phase for each mobile device. Therefore, our solution, which does not rely on such a time-out, is much easier to set up and more effective.

III. Solution

In order to avoid setting up time-outs for aborting a transaction, our solution distinguishes between two states in which a MH can wait for the coordinator's voteRequest message: the *blocking state* and the non-blocking *Adjourn State*. A MH is allowed to switch unilaterally from the blocking to the Adjourn State as long as the vote has not been sent. However, the MH must perform a *second adjourn specific validation phase* before a transaction is allowed to leave the Adjourn State. Both states, the blocking state and the non-blocking Adjourn State differ in the way the validation phase for a concurrent transaction is executed, and therefore show a different blocking behavior.

A. The Blocking State

While a successfully validated transaction T_V is in the blocking state, the validation of a newer transaction T_N against the older transaction T_V is done by T_N as described in Section II-B. This means, transaction T_N is validated against T_V with the effect that whenever transaction T_N is in conflict with T_V, T_N is aborted.

B. The Non-Blocking Adjourn State

A successfully validated transaction T_V may enter the non-blocking Adjourn State, after T_V has sent the result message to the initiator. However, T_V must switch from Adjourn State to blocking state before it may send the vote to the coordinator.

While T_V is in the non-blocking Adjourn State, the validation of a concurrent transaction T_N is done as follows: T_N is validated against all older transactions *except* those being in the Adjourn State. This means, T_V, which is in the Adjourn State, has no blocking effect on concurrent transactions T_N.

[1]Note that in 3PC, a database is not allowed to unilaterally abort a transaction after the first vote message has been sent, cf. [11].

When T_V must leave the Adjourn State, e.g., when the commit coordinator demands a binding vote on T_V, T_V must be validated again in a second adjourn specific validation phase. However, the scope of this second validation is different from the first validation phase: This second validation of a transaction T_V is successful, if and only if the following condition holds for each transaction T_N that has started its validation while T_V has been in the Adjourn State:

$$(\mathsf{readset}(T_N) \cup \mathsf{writeset}(T_N)) \cap \mathsf{writeset}(T_V) = \varnothing.$$

If this validation fails, T_V must either be aborted or can be locally restarted.

The reason for this concurrency check is the following: although T_V entered its validation phase before T_N, i.e., T_V is older, T_N has not been validated against T_V. Since T_N may have already been committed, the validation of T_V against T_N must be either successful, or T_V must be aborted.

Note that the Adjourn State only delays the validation of T_N against T_V and lets T_V validate against T_N instead of T_N against T_V. However, the number of validation tests is exactly the same as with other commit protocols that use backward oriented concurrent validation.

IV. Experimental Evaluation and Results

In order to evaluate our proposed Adjourn State, we have developed a simulator of a mobile environment in which MHs participating in the execution of a transaction may disconnect and reconnect after a specified time and/or a number of messages are dropped during a specified period. We have compared the Adjourn State and the "traditional" blocking state with different time-out values by measuring transaction throughput and blocking behavior.

Our experiments have shown that concurrency control using our Adjourn State enhancement blocks significantly less transactions compared to the one using the traditional blocking state. Hence, using Adjourn State significantly decreases the number of aborted transactions. Additionally, the Adjourn State achieves a significantly higher transaction throughput in unreliable networks with many, short disconnections.

Furthermore, our experiments have confirmed the difficulty in setting up the right time-out that increases the transaction throughput and reduces the amount of blocking. This justifies the use of the Adjourn State even in mobile networks with moderate reliability, since Adjourn State protocols do not expose the user to the risk of setting up a "wrong" time-out that leads to performance degradation.

V. Related Work

To avoid locking, concurrency control mechanisms like multiversion concurrency control [2], timestamp-based concurrency control [9], or optimistic concurrency control [8] have been proposed. However, these approaches do not solve the problem of setting up time-outs when the database has to abort a transaction. Our proposed Adjourn State does not rely on such time-outs, and merges nicely with these concurrency control mechanisms since it is an "on demand" strategy for giving concurrent transactions access to resources used by transactions waiting for the commit protocol to be invoked.

The suspend state, which is proposed by [5], relates to our concept. However, this approach uses locking instead of validation and is intended for the use within an environment consisting of several mobile cells and a fixed-wired network, in which disconnections are detectable and even foreseeable, and therefore transactions are considered to be compensatable. In contrast, our solution does not rely on the concept of compensation.

Compared to our previous contribution [3], the Adjourn State proposed in this paper is developed for the combination of optimistic concurrency control and atomic commit protocols. Since the Adjourn State can be combined with a dynamic transactional model, the coordinator must get to know the participating sub-transactions. An approach that allows the coordinator to keep track of all dynamically invoked sub-transactions is described in [4].

Our approach is based on the same optimistic principle as [1], but differs as the Adjourn State does not block resources after the read phase's result has been sent.

VI. Conclusion

In this paper, we have presented Adjourn State, which is a concurrency control enhancement for atomic commit protocols that use optimistic concurrency control. A benefit of Adjourn State is the omission of setting up time-outs for aborting a transaction in case of network or coordinator failures, which makes the Adjourn State particularly applicable in unreliable environments and environments in which disconnections are unpredictable, such as in a mobile ad-hoc environment.

References

[1] Y. Al-Houmaily, P. K. Chrysanthis, and S. P. Levitan. An argument in favor of the presumed commit protocol. In *Proc. of the 13th Int'l Conference on Data Engineering*, pages 255–265, April 1997.

[2] P. A. Bernstein and N. Goodman. Multiversion concurrency control - theory and algorithms. *ACM Trans. Database Syst.*, 8(4):465–483, 1983.

[3] S. Böttcher, L. Gruenwald, and S. Obermeier. Reducing sub-transaction aborts and blocking time within atomic commit protocols. In *23rd British National Conference on Databases (BNCOD), Belfast, Northern Ireland, UK*, pages 59–72, 2006.

[4] S. Böttcher and S. Obermeier. Dynamic commit tree management for service oriented architectures. In *Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS), Funchal, Madeira - Portugal*, 2007.

[5] R. A. Dirckze and L. Gruenwald. A toggle transact. management technique for mobile multidatabases. In *CIKM '98*, pages 371–377, New York, USA, 1998. ACM Press.

[6] J. Gray. Notes on data base operating systems. In M. J. Flynn, J. Gray, A. K. Jones, et al., editors, *Advanced Course: Operating Systems*, volume 60 of *Lecture Notes in Computer Science*, pages 393–481. Springer, 1978.

[7] V. Kumar, N. Prabhu, M. H. Dunham, and A. Y. Seydim. Tcot-a timeout-based mobile transaction commitment protocol. *IEEE Trans. Com.*, 51(10):1212–1218, 2002.

[8] H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. *ACM Trans. Database Syst.*, 6(2):213–226, 1981.

[9] P.-J. Leu and B. K. Bhargava. Multidimensional timestamp protocols for concurrency control. In *Proceedings of the Second International Conference on Data Engineering*, pages 482–489, Washington, DC, USA, 1986. IEEE Computer Society.

[10] P. K. Reddy and M. Kitsuregawa. Reducing the blocking in two-phase commit with backup sites. *Inf. Process. Lett.*, 86(1):39–47, 2003.

[11] D. Skeen. Nonblocking commit protocols. In Y. E. Lien, editor, *Proceedings of the 1981 ACM SIGMOD Intl. Conference on Management of Data, Ann Arbor, Michigan*, pages 133–142. ACM Press, 1981.