Non-Blocking One-Phase Commit Made Possible for Distributed Transactions over Replicated Data

Yuqing ZHU
Institute of Computing Technology, Chinese Academy of Sciences
Beijing 100190, China
Email: zhuyuqing@ict.ac.cn

***Abstract*—Distributed transaction can enable not only large-scale transacting business in highly scalable datastores, but also fast information extraction from big data through materialized view maintenance and incremental processing. Atomic commitment is key to the correctness of transaction processing. While two-phase commit (2PC) is widely used for distributed transaction commitment even in modern large-scale datastores, it is costly in performance. Even though variants of 2PC can improve performance, they are blocking on server failures, impairing data availability. Existent non-blocking atomic commitment protocols are too costly for practical usage. This work presents Pronto, a non-blocking one-phase commit protocol for distributed transactions over replicated data. Pronto removes the prepare phase of 2PC, reducing communication and logging costs. Without transaction client failure, Pronto can commit a transaction in one communication roundtrip; and, client failures cannot block transaction processing. Pronto can also tolerate server failures. Pronto is compared to state-of-art commitment protocols. Pronto outperforms other commitment protocols in workloads with varying numbers of participants. When a transaction has many participants, Pronto can commit in one fifth of the time 2PC does.**

***Keywords*-atomic commitment; coordination; transaction processing;**

I. Introduction

Distributed transaction can enable not only large-scale transacting business in highly scalable datastores, but also fast information extraction from big data through incremental processing and materialized view maintenance [1], [2]. It can facilitate the development of big data applications. Hence, recent years have witnessed the resurgent interests in database transaction, which was once given up in order to guarantee high availability of large-scale datastores. It is highly desired that general distributed transactions are supported in large-scale datastores with high availability and performance. One key obstacle to this goal is the atomic commitment of distributed transactions [3].

Committing a distributed transaction requires the running of an atomic agreement protocol that typically involves two phases of processing. The two-phase commit (2PC) protocol is still widely used in modern databases and large-scale datastores, e.g., Spanner [4] and H-Store [5]. While optimizations proposed for 2PC, e.g., 1PC protocols [6], can improve performance, 2PC and its variants are blocking protocols, impairing the availability of large-scale systems. Existent non-blocking atomic commitment protocols such as Paxos Commit [7] and three-phase commit [8] are too costly for practical usage.

This work presents Pronto, a non-blocking one-phase commit protocol for distributed transactions over replicated data. Thanks to the fact that all data are replicated, Pronto removes one processing phase of 2PC, eliminating synchronous logging and reducing communication roundtrips. Without transaction client failure, Pronto can commit a transaction in one communication roundtrip; and, client failures cannot block transaction processing. Pronto can also tolerate server failures. Pronto can commit in fewer communication roundtrips and messages than the classic protocols of 2PC and Faster Paxos Commit [7], as well as the recent works of replicated commit [9] for multi-datacenter scenarios.

In the following, Section II overviews the design of Pronto and Section III illustrates the evaluation of Pronto towards other state-of-art commitment protocols. Section IV draws the conclusion.

II. Design Overview

Pronto targets at supporting the commitment of general distributed transactions in highly scalable datastores, which shard and replicate data. A general distributed transaction first processes reads and writes, applying concurrency control rules, and then the commitment. A transaction client can commit the transaction only if it successfully reads or writes data and the commitment is allowed by the concurrency control scheme. Pronto improves the commitment process.

Pronto reduces communication roundtrips and eliminates synchronous logging by removing the prepare phase of 2PC, which involves two processing phases, i.e., the prepare phase and the commit phase. A 2PC coordinator coordinates the transaction participants to commit a transaction, following the 2PC protocol. Though a transaction actually commits in the commit phase of 2PC, the prepare phase is introduced for two demands. First, the coordinator needs to decide the transaction outcome, because some factors can force an abort. Second, the coordinator and participants need to log

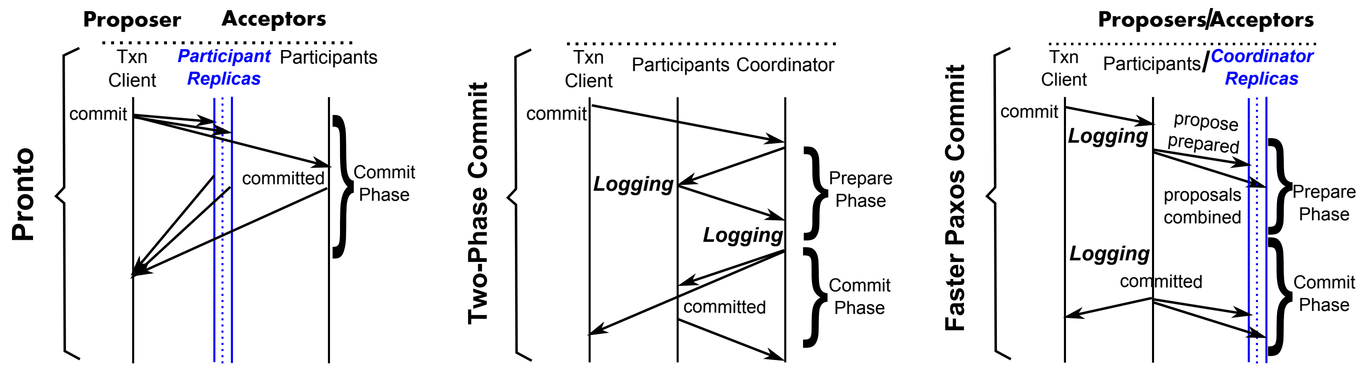

Figure 1. Figure 1. Comparing message flows of Pronto, Two-Phase Commit and Faster Paxos Commit in the fail-free case.

enough information for a correct recovery on failures [3]. As consistency and isolation checks can be made and acknowledged in the last operation of a transaction, the prepare phase of 2PC is unnecessary, when (1) participant failures cannot force an abort; (2) failure recovery is not based on logging by the coordinator and participants.

For distributed transactions over replicated data, each server holding any replica of any data item in a transaction is a participant. Participant failures cannot force an abort, as long as a quorum of replicas are available for each data item involved in a transaction; and, participants do not have to log for a correct recovery on failures. The reason is that failed participants can recover by copying from other correct participants. As committing a distributed transaction is to reach a consensus among participants, the non-blocking consensus algorithm Paxos [10] can be exploited to reach the commit consensus. The proposer acts as the coordinator and the commitment information can be kept by participants. As proposers are replaceable, a substituting proposer can correctly end the transaction on proposer failures, using the information kept at the correct participants. So neither coordinator logging is needed for transaction failure recovery. Therefore, Pronto removes the prepare phase of 2PC.

Furthermore, to reduce the number of communication roundtrips, Pronto initiates a Paxos instance for each transaction commit and assigns the unique transaction client as the only and the initial proposer of the instance. With Paxos, a proposer can directly propose any value to be accepted as the consensus, if the proposer is the only and the initial proposer in the instance (i.e., a run of the Paxos algorithm), incurring one single communication roundtrip. That is, Pronto can commit a transaction in one communication roundtrip on fail-free cases.

With its basis on Paxos, Pronto can tolerate at most $\ulcorner n/2 \urcorner$ failures, given that each transactional data item has n replicas stored at n different servers. On client failures, Pronto falls back to the classic Paxos algorithm, thus requiring at least two more communication roundtrips to settle the commitment. One of the servers involved in the transaction will be selected as the new proposer on the client failure. The selection can exploit some leader selection service to guarantee the protocol liveness. Such service is widely used in many fault-tolerant protocols. In sum, Pronto is non-blocking under server or client failures.

A. llustration by Comparison

Figure 1 compares Pronto to Two-Phase Commit (2PC) and Faster Paxos Commit (FPC). Compared to 2PC, FPC incurs fewer communication roundtrips but no fewer messages. The state-of-art practice for committing distributed transactions over replicated data either layers 2PC over Paxos-based replication [4] or vice versa [9], incurring no fewer costs than 2PC.

The three protocols in Figure 1 have different assumptions about replication. While 2PC does not assume any replication, Pronto assumes data replication and FPC assumes the replication of coordinator. Besides, although both FPC and Pronto exploit Paxos, their exploitations are different. With n participants, FPC initiates n Paxos instances for a commit, while Pronto initiates only one Paxos instance. In FPC, transaction participants act as proposers and acceptors in Paxos instances, while they take the acceptor role in Pronto. The logging for 2PC and FPC as demonstrated in the figure is indispensable. These differences are factors that lead to their performance differences.

III. Evaluation

In the evaluation, Pronto is compared to 2PC and RCommit [9]. 2PC is still considered the standard protocol for committing distributed transactions. It assumes no replication and is not resilient to single node failures. RCommit uses replication as Pronto does. RCommit layers the Paxos algorithm over 2PC. It has better performance than the approach that layers 2PC over the Paxos algorithm [4]. All commit protocols are evaluated using the YCSB benchmark [11], which is extended to support the evaluation of transaction commitment.

The evaluation focuses on the actual costs of the commitment process. The number of participants and data

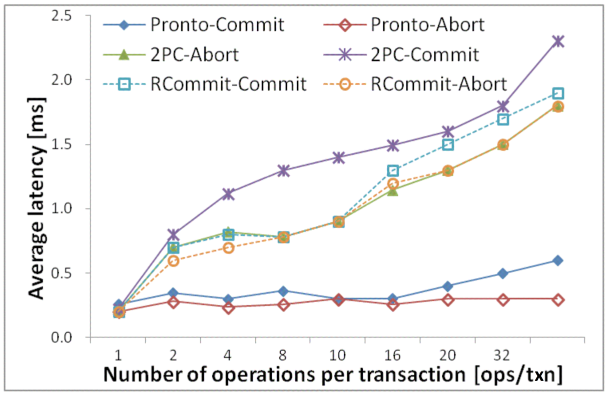

Figure 2. Commit latencies when increasing the number of operations per transaction.

items involved in a transaction is the key factor affecting the performance of commitment protocols, thus Pronto, 2PC and RCommit are evaluated on various numbers of operations per transaction. Each operation operates on a different data item. All protocols use the same locking-based concurrency control scheme. The duration of the commitment process is studied.

Pronto outperforms 2PC and RCommit in all workloads. Figure 2 shows the latencies of commit and abort. The number of operations per transaction is varied from 1 to 64. The advantage of Pronto increases as the number of operations per transaction increases. When a transaction has 64 operations, Pronto can commit in one fifth of the time 2PC does.

The abort latency of Pronto is almost constant, while Pronto's commit latency increases slightly as the number of operations increases to 20. On committing a transaction, the system must apply all writes and release all locks. When the number of operations is small, applying writes and releasing locks in the in-memory database cost a small amount of time, as compared to the network communication roundtrip time (RTT). As the number of operations increases, the time needed increases slightly for applying all writes in the in-memory database. Thus, the commit latency of Pronto increases, but the abort latency is unaffected.

2PC and RCommit have increased latencies for commit and abort as there are more operations per transaction. They need to log writes and the old values of data items, thus the time needed for the prepare phase increases as the number of writes goes up, leading to a longer commitment process. The aborts in the experiments are due to the failure of lock acquisition. Thus, the aborts also involve a prepare phase, leading to a considerable cost. 2PC has a higher commit latency than RCommit, because 2PC must log in-memory data and RCommit relies on replication for fault tolerance.

In sum, Pronto has smaller commit latency than 2PC and RCommit. As the number of participants per transaction increases, Pronto's advantage increases accordingly.

IV. Conclusion

This work presents the design of Pronto, a non-blocking commitment protocol for distributed transactions over replicated data. In the evaluation based on an extended YCSB benchmark, Pronto outperforms state-of-art commitment protocols and demonstrates stable commit performance as the number of transaction participants increases.

Acknowledgment

This work is supported in part by the State Key Development Program for Basic Research of China (Grant No. 2014CB340402) and the National Natural Science Foundation of China (Grant No. 61303054 and No. 61432006).

References

[1] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications." in *OSDI*, vol. 10, 2010, pp. 1–15.

[2] H. V. Jagadish, I. S. Mumick, and A. Silberschatz, "View maintenance issues for the chronicle data model (extended abstract)," in *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, ser. PODS '95. New York, NY, USA: ACM, 1995, pp. 113–124.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Addison-wesley New York, 1987, vol. 370.

[4] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild *et al.*, "Spanner: Google's globally-distributed database," *Proceedings of OSDI*, p. 1, 2012.

[5] E. P. Jones, D. J. Abadi, and S. Madden, "Low overhead concurrency control for partitioned main memory databases," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 603–614.

[6] M. Abdallah, R. Guerraoui, and P. Pucheral, "One-phase commit: does it make sense?" in *Parallel and Distributed Systems, 1998. Proceedings. 1998 International Conference on*. IEEE, 1998, pp. 182–192.

[7] J. Gray and L. Lamport, "Consensus on transaction commit," *ACM Trans. Database Syst.*, vol. 31, no. 1, pp. 133–160, Mar. 2006.

[8] D. Skeen, "Nonblocking commit protocols," in *Proceedings of the 1981 ACM SIGMOD international conference on Management of data*. ACM, 1981, pp. 133–142.

[9] H. A. Mahmoud, A. Pucher, F. Nawab, D. Agrawal, and A. E. Abbadi, "Low latency multi-datacenter databases using replicated commits," in *Proceedings of the VLDB Endowment*, 2013.

[10] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 2, pp. 133–169, 1998.

[11] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st SoCC*. ACM, 2010.