ABSTRACT
Whereas the two-phase commit protocol guarantees global atomicity, its biggest drawback is that it is a blocking protocol - whenever the coordinator fails, cohort sites will have to wait for its recovery. Although three-phase commit protocol was devised to reduce blocking situation, it increases latency. In this paper we propose a proxy based two-phase atomic commitment protocol to improve the performance of two-phase commit protocol by reducing the number of forced-writes and extending the protocol to a non-blocking one. Another aspect of our protocol is to save the transactions from unnecessary restarts and thus reduces the execution time.

KEY WORDS
Atomic commit, blocking and non-blocking, forced write, stable memory, recovery.

1. Introduction
A major problem in a distributed data management system is how to achieve robust execution of distributed programs in spite of failures [2], [4]. A common approach to this problem is to define an atomic transaction [7], [8], [11] as a unit that preserves consistency in spite of failures (e.g., processor crashes) and parallel executions.

A transaction is a set of invocations on data objects, characterized by the concurrency atomicity and failure atomicity properties [1]. Failure atomicity, also called all-or-nothing property, is ensured by an atomic commitment protocol, launched at the end of the transaction: the atomic commitment protocol returns either commit or abort.

The cost of the atomic commitment protocol is an important factor in the performance of distributed transactions. Commit processing has involved strong attention because of its effect on the performance of the transaction processing. In [5], using simulation model, it has been shown that distributed commit processing can have more influences than distributed data processing on the throughput performance. The time to commit accounts for about one third of transaction’s lifetime [6]. It has been shown that the time to commit can be as high as 80 percent of the transaction time in the wide area network (Internet) environments.

For achieving atomic commit, several protocols have been devised. For example, Two phase commit protocol (2PC) [3] and Three phase commit protocol (3PC) [9] are the most common examples of atomic commitment protocol. Two phase commit protocol is the simplest one with blocking nature. On the other hand three phase commit protocol reduces the blocking characteristic with increased latency. Thus if two phase commit protocol can reduce the blocking phenomena it will simply the best. Another scope of work remains in decreasing the forced-writes of 2PC protocol as these forced-writes consume a significant amount of time.

In this paper, we propose a variant of 2PC protocol to upgrade the performance of 2PC by reducing the amount of forced-writes and blocking probability. The paper is organized as follows. In section 2, we explain the motivation of our protocol. In section 3, system model has been described. We briefly describe 2PC in section 4. In section 5, we propose proxy based two phase commit protocol with wait (P2PC-W). In section 6, we discuss the performance issues. Finally we have drawn conclusion in section 7.

2. Motivation
2PC is the simplest (it only requires three communication steps to commit) but blocking protocol. Although 3PC solves this problem, in the absence of failures it has a higher latency (i.e., it needs more time to commit) than 2PC: this is due to two additional communication steps between the coordinator and the participants in the 3PC protocol. In [5] and [10], it has been reported that as compared to 2PC protocol, the performance is further degraded with 3PC protocol due to an extra round of message transmission. Besides, with realistic failure
detectors (that can make mistakes), 3PC protocol still blocks. And bad case arises during network partitioning when the network splits the participating processes into two or more sets of operational processes. Above all, it can be proved that there are no non-blocking commit protocols for asynchronous networks. This is the motivation for working with 2PC instead of employing 3PC. And also 2PC protocol is widely applied protocol in commercial database systems. Even though 3PC protocol eliminates blocking, it has not entered into commercial database systems. In this situation, we have made an effort to resolve the blocking problem of 2PC protocol by employing proxies. In addition, the proposed protocol can easily be adapted with existing 2PC protocol implementations.

3. System Model

We consider a distributed transaction composed of a finite set of processes \( \pi = \{P_1, P_2, \ldots, P_n\} \) which resides on different sites and are completely connected through a set of channels. Communication is done by message passing, asynchronous and reliable. Asynchrony means that there is no bound on communication delays. A reliable channel ensures that a message sent by a process \( P_i \) to a process \( P_j \) is eventually received by \( P_j \), if \( P_i \) and \( P_j \) are correct (i.e. do not fail). Processes fail by crashing. We are assuming fail-stop assumption. The fail-stop model makes following assumptions about the failure behavior of processes: i) processes fail only by permanently crashing, ii) when a process crashes, surviving processes will eventually detect that failure and iii) stable storage contents are assumed not to be corrupted by system failure. We do not consider Byzantine failures. Transactions are represented by \( T_i, T_j, \ldots \) where \( i, j \) are integers. Each coordinator is augmented with a proxy. The originating site of \( T_i \) acts as a coordinator in \( T_i \)'s commit processing. The sites which are involved in the processing of \( T_i \) are called cohorts or participants of \( T_i \). The proxy site does not assume the role of coordinator when coordinator is alive. Both coordinator and proxy are repairable items.

4. Two Phase Commit Protocol

In 2PC there is a coordinator site and one or more participant sites or cohorts. A coordinator site, usually the site where the global transaction originates, is responsible for orchestrating the sub transactions. Its responsibility is to coordinate the atomic commitment, by receiving the votes of cohorts and forwarding the final decision after all votes are received. The responsibility of cohorts is to send their votes after they are ready to commit or abort. In the former case, the coordinator’s decision is required before the actual commit. Cohorts communicate with their coordinator by messages sent through the communication network.

Two phase commit protocol ensures that all participants commit if and only if all can commit successfully and simultaneously. The two phases are the voting phase and the decision phase. On sensing a need (either by receiving a message from some other site or due to local triggering) to decide whether to commit or abort a particular transaction \( T \), the coordinator initiates the protocol starting voting phase. When the coordinator determines that the local decision is to commit, it records the transaction id, and cohorts’ list for the transaction (in the log). After this information is successfully recorded, coordinator asks all the cohorts to prepare for commit sending \textit{PREPARE} messages. Each cohort that agrees to commit \( T \) writes a \textit{Committed} record in the log and (then) sends a \textit{Commit} vote to the coordinator. On the other hand, if a participant is unable to prepare to commit for any reason, it votes \textit{Abort}. The decision phase starts when the coordinator either receives Commit votes from all cohorts or an Abort vote from any cohort. In the former case, it writes a Committed record in the log and broadcast a COMMIT message to all cohorts. In the latter case coordinator broadcast an ABORT message to all cohorts, undone \( T \) and writes an Abort record in log. When \( T \) is committed, all items updated by \( T \) (if any) are copied from the log to the database and an End record is written for \( T \) in the log. Each cohort in the transaction commits or aborts the effects of the transaction based on the coordinator’s outcome. It can then release locks on local resources, such as databases or files, making them available to other transactions.

Fig. 1 shows the message flows and forced log writes involved in the classic two-phase commit protocol. The first two flows comprise the voting phase, while the next two flows comprise the decision phase.
substantial delay, which affects response time. The protocol also requires that information about transactions be stably recorded to ensure that the system can continue to guarantee transaction atomicity even if one or more elements of the system should fail during the commit protocol itself. This is usually accomplished by writing information to a log. When information should be stably recorded at some point in the protocol, the log must be forced i.e., the write must be completed before proceeding to the next step. Forced writes are more costly than simple writes because they translate into actual I/O whether a block of the log is filled or not.

To commit a transaction, 2PC coordinator does two log writes - the commit record (forced) and the transaction end record (non forced). In addition, it sends two messages to each of its cohorts - PREPARE and COMMIT. In response, each cohort performs two log writes - a prepare record and a commit record (both forced) and sends two messages - a COMMIT vote and a final ACK.

5. Proxy Based Two Phase Commit Protocol with Wait

A proxy will be associated with each coordinator. The proxy is a process may be located on the same site of its coordinator or on a remote site. It is up to the availability of the site of the coordinator process where the proxy will be located. If the availability is high, proxy can be on the same site. Otherwise proxy will be located on a remote site. The proxy remains passive while the coordinator is active. Proxy acts as coordinator whenever coordinator crashes. Both coordinator and proxy maintain timer of their own, \( T_{coord} \) and \( T_{proxy} \) respectively, to maintain timeout periods. Timer for proxy, \( T_{proxy} \) starts when proxy gets \( START \ COMMIT \) message from its coordinator. We discuss about how to synchronize \( T_{proxy} \) with \( T_{coord} \) in the next section. We employ timeout mechanism because i) it may be that some of the participating processes are holding resources, which are needed for other transactions ii) holding these resources may reduce throughput of transaction processing, which, of course, is a bad thing and iii) timeout mechanism may help to find out that something is wrong.

5.1 Protocol

P2PC-W protocol works in two phases like 2PC protocol as shown in Fig. 2.

First Phase

1. If the coordinator likes to commit the transaction it records a \( START \ COMMIT \) in its log, sends a \( START \ COMMIT \) message to its proxy, then sends \( PREPARE \) messages to all cohorts and set timer \( T_{coord} \).

The coordinator enters a wait state while the \( PREPARE \) messages are delivered. Coordinator will wait until it gets Commit messages from all the cohorts or an \( Abort \) message from any cohort or the timer \( T_{coord} \) expires.

2. Upon receiving the \( START \ COMMIT \) message from the coordinator, the proxy writes \( Start Commit \) in its log and sets its timer \( T_{proxy} \) to \( timeout\_period\_1 \).

3. When a cohort receives a \( PREPARE \) message, it will send one of the following messages:

   - Commit message – if the cohort is prepared to commit the transaction. Before sending a Commit message to coordinator, the cohort will write Prepared to commit in its log (forced write).
   - Abort message – if the cohort has failed to perform the operations requested by the coordinator. After sending an Abort message to coordinator, the cohort suspends and immediately forgets about the transaction.
   - Wait message – if the cohort is successfully performing its job but not yet finished its part. Whenever the cohort finishes its job, it will write Prepared to Commit in its log (forced write) and sends a Commit message to the coordinator.

**Fig. 2: Proxy Based Two Phase Protocol with Wait**

Second Phase

4. If the coordinator receives a \( Wait \) message from any cohort, it resets the timeout period for \( T_{coord} \) and sends a \( RESET \) message to its proxy. The proxy will reset its timer \( T_{proxy} \) on receiving the \( RESET \) message. If the coordinator receives Commit messages from all cohorts, it writes Commit Transaction in the log, sends a \( COMMIT \_TRANSACTION \) message to its proxy and
sends COMMIT messages to all cohorts and wait for Ack messages. On the other hand, if the coordinator gets Abort message even from one cohort, the coordinator it sends ABORT messages to all cohorts after writing Abort Transaction to the log and sending an ABORT TRANSACTION message to the proxy. If the timer of the coordinator, $T_{coord}$ times out, the coordinator follows the same process as receiving an ABORT message as above.

5. Upon receiving either a COMMIT or an ABORT message from the coordinator within $timeout_{period_1}$, the proxy resets the timer $T_{proxy}$ to $timeout_{period_2}$.

6. If cohorts get COMMIT messages from the coordinator, they write Commit Transaction to their corresponding logs (forced-write) and send Ack messages to the coordinator after committing the transaction. Cohorts can now release locks on local resources, such as databases or files, making them available to other transactions.

7. The coordinator writes End Commit to its log and sends an END_COMMIT message to proxy.

8. The proxy stops its timer and writes End Commit to its log after receiving the END_COMMIT message from the coordinator.

5.2 Proxy’s action on coordinator crash

In normal operation, a proxy just receives messages from its coordinator and does not take part in the decision making for the outcome of the transaction. If the coordinator crashes, cohorts can communicate with proxy for the unanimous decision of the transaction. Also the coordinator can communicate with its proxy for the transaction it was executing and proxy informs its coordinator about the transaction status. Fig 4 depicts the communications among coordinator, proxy and cohorts during normal execution and failure.

![Fig. 3 Communications among Coordinator, proxy and cohort](image)

The proxy is designed to take part in sending messages to cohorts and coordinator after the coordinator process crashes. After proxy recognizes that the coordinator is crashed, all the writings in the log of proxy would be forced-write.

If the proxy does not receive either a COMMIT or an ABORT message from its coordinator within $timeout_{period_1}$, the proxy assumes the coordinator is crashed, the proxy takes over the role of the coordinator. It sends REPEAT messages to all the cohorts for votes and makes decision about the outcome of transaction. The cohorts reply to the proxy with the previous messages, either Commit or Abort, they sent to the coordinator. The proxy will decide whether to globally commit or abort the transaction according to the replies of the cohorts. If all cohorts reply with Commit messages, the proxy will write Commit Transaction in the log and send COMMIT messages to all the cohorts. If one of the cohorts sends an abort message to the proxy, it will write Abort Transaction in the log and send ABORT messages to all the cohorts.

If the proxy cannot receive an END COMMIT message by the time $timeout_{period_2}$ expires, the proxy obeys its coordinator’s decision and let the decision reach to all the cohorts. That is, the proxy will send either COMMIT or ABORT messages to all the cohorts. The cohorts will act according to the proxy’s message.

We assume that the communication between coordinator and proxy is reliable.

Figure 4 describes the communication among the proxy, the coordinator and its cohorts for coordinator crash at different stages in P2PC-W.

5.3 Recovery protocol

As a coordinator does not keep its log in stable storage, when the coordinator recovers from failure, it will not find any log. But the same log can be found in its proxy’s side. The proxy maintains its log in stable storage after the coordinator crashes. When the coordinator recovers from failure, it will contact to its proxy for the status of the coordinator at the time it fails. The coordinator may be in one of the following states – 1) if the coordinator finds Start_commit record but no other records in the log, it will abort the transaction. 2) If the coordinator finds Committed record in the log, it will commit the transaction.

When a cohort recovers from failure, it will check the log records of its own. If the cohort finds Prepared record in the log, it will send a Commit message to the coordinator and will wait for the response of the coordinator. If the coordinator finds the transaction has already aborted it will send an ABORT message to the cohort. Otherwise the coordinator sends a COMMIT message to the cohort. If the cohort does not find any Prepare message in the log, it will simply abort the transaction. If the cohort finds
Committed record in the log, it sends an Ack message to the coordinator and ends the transaction.

6. Performance Discussion

P2PC-W protocol gives better performance in three ways which are briefly described in the following paragraphs—

1. It reduces the required time for commit operation by reducing one forced-write in coordinator side.
2. It reduces the chance of cohorts being blocked due to the coordinator failure.
3. The protocol sometimes saves transaction from aborting and then restarting again. Thus helps transactions to finish execution earlier.

6.1 Reducing time to commit

In 2PC protocol three forced-writes are necessary— one in coordinator side and two in cohort side. During the forced-write the write must be completed before proceeding to the next step. Forced writes cost more than simple writes because they require actual I/O, whether a block of the log is filled or not [13]. The forced-writes are performed because 2PC requires that information about transactions be recorded stably to ensure that transactions remain atomic even if there is a failure during the commit protocol. As a proxy has been introduced with the coordinator, the availability of coordinator side is increased and there is no need for a forced-write in coordinator side. Thus reduce the response time of atomic commitment protocol.

6.2 Reducing blocking probability

P2PC-W protocol reduces the blocking probability of cohorts in 2PC protocol by extending the services of a coordinator by the proxy when the coordinator is down. The protocol will be blocked only when both the coordinator and proxy crash at the same time—which is a very uncommon situation. Thus, with the introduction of the proxy, blocking probability is considerably reduced as compared to 2PC protocol. Further, it can be observed that the purpose of the proxy is to complete the blocked transactions at the participant sites when the corresponding coordinator is down. After the completion of the blocked transactions, even though the proxy fails, it does not affect the consistency of the database.

6.3 Saving transaction from unnecessary restarts

Suppose a transaction spans \( n \) sites. Whenever the coordinator sends \( PREPARE \) messages to all its cohorts, according to 2PC protocol, a cohort is bound to send either Commit or Abort message to the coordinator. In a general case, where not all the cohorts are equally speedy, this may happen that \((n-1)\) cohorts have completed their tasks and send Commit messages to the coordinator while one cohort is about to finish its job but not yet completed. In this case this cohort will send an Abort message to the coordinator. As the coordinator gets one Abort message, it will send \( ABORT \) messages to all the cohorts and the transaction will be aborted. If the transaction is restarted later the same incident may happen again. In P2PC-W protocol there remains a provision for a cohort to send a Wait message to the coordinator if the cohort is successfully doing its tasks while it gets the \( PREPARE \) message from the coordinator. After sending a Wait message, whenever the cohort ends successfully, it will send a Commit message to the coordinator. Otherwise it will send an Abort message. If the cohort does not send any Wait message and keeps on doing its tasks, the coordinator may timeout and assumes that the cohort was...
crashed. In this case the coordinator will send \textit{ABORT} messages to all cohorts and the transaction will be aborted. That’s why an explicit \textit{Wait} message is necessary.

6.4 Message overhead, latency and failures

As compared to 2PC protocol, to commit a transaction, P2PC-W protocol requires extra messages to communicate with the proxy. Nevertheless, communication between a proxy and its coordinator and between cohorts and their coordinator can take place in parallel. On the other hand as there is no need for forced-write in the coordinator side, time to commit will not increase for P2PC-W. However if the coordinator and the proxy reside in same site required time for P2PC-W will be reduced than that of 2PC protocol.

Also, in P2PC-W protocol, the overhead during the recovery is considerably reduced: after recovery, the coordinator terminates the transaction consistently by only contacting the proxy. However, in a rare case, if it is unsuccessful in contacting the proxy, it has to demand the state information from all the participants.

As an option, to minimize the latency we suggest to choose the nearest site to the coordinator for its corresponding proxy. For example, both the coordinator site and its proxy site are connected to the same local area network (Ethernet). When partition failure occurs in wide area network both the coordinator and corresponding proxy fall in the same partitioning group. As a result, the participants of other groups wait until the connection is repaired. In this way, P2PC-W protocol ensures consistent termination of transactions in case of partitioning failures.

7. Conclusion

In this paper we extend 2PC protocol to an atomic commitment protocol that reduces blocking probability and also helps transactions to avoid unnecessary aborts. A coordinator is augmented with a proxy to reduce the blocking probability. Whenever the coordinator crashes, the proxy can act as the coordinator. Thus cohorts do not need to wait for the coordinator’s response and can release the resources after committing or aborting the transaction. Moreover, as the probability of simultaneous crash of both the proxy and coordinator is significantly low, the need for log records being forced-write in the coordinator side is no more essential. Thus P2PC-W reduces the time of atomic commitment. Beside this, P2PC-W protocol allows cohorts to make request to their coordinator to wait for cohorts to finish their tasks. This provision readily helps transactions to avoid unnecessary aborts.

References


