Token Based Decentralized Non-Blocking Two Phase Commit Protocol

HARIT SHARMA
M.E. Student
Department of Computer Science & Engineering
Dr. B.R Ambedkar National Institute of Technology, Jalandhar (Punjab)

Abstract

Transactional standards in distributed transactional systems, allows heterogeneous resources to participate in an Atomic Commitment Protocol (ACP). An ACP ensures the atomicity of distributed transactions even in the presence of site failures. Atomic commitment is one of the key functionalities of modern distributed information systems. Two-Phase Commit Protocol (2PC) is a well-known and widely accepted standard algorithm which safeguards the ACID properties of a transaction in the distributed system. But, since this algorithm introduces a substantial time delay and is a blocking protocol, there has always been an open challenge or issue to solve while designing a reliable, highly available and efficient distributed system. Blocking in distributed transactions, occurs during two-phase commit processing, when the coordinator itself fails and at the same time one or more sites are ready to commit the transaction. Several variants of 2PC, optimized protocols and non-blocking protocols have been proposed in the literature, but none of them combines high efficiency during normal processing with fault-tolerance (i.e. non-blocking). Generally the proposed protocols either violate site autonomy or are inherently more costly in time and increase communication overhead. This paper proposes a Decentralized Non-Blocking Two Phase Commit Protocol that will drastically reduce the cost of distributed transaction commitment in terms of time delay to reach a common view, i.e. a consensus between the cooperating processes across various sites on the global state, and message complexity as well. Since this algorithm is decentralized in nature, it will overcome the disadvantage of centralized architecture which relies heavily on the central computing unit (master controller or co-ordinator). This paper describes a token-based non-blocking commitment protocol that also subsumes the functions of termination and recovery protocols. The protocol survives any single site crash or network partition provided that the failure is not falsely detected. The protocol is correct despite the occurrence of failures at multiplesites and whether or not failures are falsely detected. Performance measures show that the protocol is significantly faster and efficient than obsolete 2PC.The protocol is simple, and has several advantages, including prevention of deadlocks, avoids blocking and single point of failure kind of dependencies as well. The protocol is equipped with fast abort properties, i.e., aborting a transaction as soon as possible if some site wants to abort which is involved in distributed transaction, and the fast commit property, i.e., committing a transaction as soon as possible when all the sites are ready to commit, and no site crashes. This protocol allows operational sites to continue transaction processing even though site failures have occurred. So for an algorithm to be efficient, it is desirable to keep the number of message exchanges on a low level and keeping the number of phases as minimal as possible, which redirect us to the two-phase commit protocol. In order to achieve the non-blocking property there have been presented a variety of solutions throughout the years. To maintain a low level of message exchanges the protocol uses the concept of token and does not uses acknowledgements, as a substitute the participants use the Store and Forward manner and update their vote on the token. And since the architecture is decentralized, there's no need to elect any co-ordinator either. The vast majority of complex systems like conventional distributed databases, transaction processing monitor, or distributed object platforms implement atomic commitment using some variation of 2 Phase Commit, (although 2PC may block under certain conditions) and since this algorithm inherits few properties of 2PC this will make the protocol compatible with the existing DDBS's.

Keywords: Atomic Commitment Protocol (ACP), ACID properties , Two-Phase Commit Protocol (2PC).

I. INTRODUCTION

Recently, much emphasis has been laid on the development of distributed systems integrating several data sources. Some standards define transactional interfaces that allow distributed data sources to participate in a two-phase commit (2PC) protocol, generally coordinated by Transactional Processing Monitors.[1]The transaction concept has proven to be very important for the design and implementation of fault-tolerant distributed applications.[2]

Transaction is an atomic set of operations updating shared data objects. Transaction atomicity, also called all-or-nothing property, means that either the transaction successfully executes to completion and the effects of all of its operations are recorded in the accessed objects (the transaction is said to be committed), or it fails and it has no effect at all (the transaction is aborted). To ensure the atomicity of a transaction accessing distributed data objects, all participants in the transaction must coordinate their

actions so that they either unanimously commit or unanimously abort the transaction. This is achieved through an Atomic Commitment Protocol (ACP), launched at the end of the transaction.[3] Two Phase Commit (2PC) Protocol is the simplest, best known and most widely used ACP, but it has two major drawbacks. First, it is blocking: if the coordinator crashes between the vote phase and the decision phase, a transaction can hold system resources for an unbounded period. Second, it incurs three sequences of message rounds (request for vote, vote and decision) that introduce a substantial delay in the system even in the absence of failures. This makes 2PC not adequate to today's highly reliable distributed platforms. [4]

Several solutions have been proposed to eliminate the voting phase of the 2PC, and a number of non-blocking commit protocols have been proposed in the literature. The simplest is the three phase commit protocol (3PC). [5] 3PC is non-blocking at the expense of two extra message rounds needed to terminate a transaction even in the absence of failures. This high latency makes the 3PC not adapted to today's systems with long mean time between failures.

This paper proposes a new atomic commitment protocol, a Token Based Decentralized Non-Blocking Two Phase Commit Protocol. It has a low latency (either to commit or abort), is non-blocking and preserves the site autonomy. During normal execution as well as in case of one or more site failures, a consensus is reached for transaction to be either committed or abort. Performance shows that this protocol is more efficient than existing non-blocking protocols, and can be applied to a wide variety of commercial transactional systems, since it has an inherent feature of classical 2PC.

A. The Non-Blocking Atomic Commitment Problem

The aim of an atomic commitment protocol is to bring all the participants in a transaction to agree on an outcome of the transaction (either commit or abort). The outcome of the transaction depends on the votes (yes or no) provided by the participants involved in that transaction. The outcome can be <COMMIT>only if the votes of all participants are yes, otherwise <ABORT>. [6]

Blocking occurs when the coordinator (generally in centralized architectures) of the ACP fails, while the participants are uncertain about the commit/abort outcome of the transaction, then participants must wait for the recovery of the coordinator. This paper describes a non-blocking commitment protocol which is decentralized in its architecture and permits at least some functioning sites to terminate in spite of any failure or site crash or a network partition.

II. SYSTEM MODEL

Let there be a token packet containing a token queue S [1, n], maintaining the states of all the sites, where 'n' represents the number of sites. A site can be in any of the following four states:

- Site has received the token & wants to abort.
- R- Site has received the token; wants to commit & has agreed.
- Site wants to commit & has initiated a token for the same.
- N- None of the above or Site hasn't received any token so far.

Any site can initiate or receive the token and can only update its state, and forward it to any other site.
No site other than the initiator can destroy the token in the network until it is about to issue a <COMMIT> broadcast.
One site can initiate only one token, until it gets a broadcast saying either to commit or abort.
Only the sites involved in the transaction can initiate a token for gathering votes of all other participating sites.
This protocol assumes that every participating site knows the number of sites participating in the corresponding transaction for which the token has been raised.

III. PROTOCOL DESCRIPT

This protocol is advancement into the existing basic two phase commit (2PC) protocol. But it requires very less number of message exchanges and is non-blocking too. The best feature of this algorithm is its architecture, which is decentralized, unlike other existing traditional solutions. Decentralized architecture is an add-on to any distributed system. The fundamental difference between traditional, centralized systems and distributed systems is that distributed systems have the partial failure property, i.e. a component may fail, but the rest of the system continues to work [7]. This paper also describes the failure handling strategies, proving that the algorithm remains un-affected in case of site failures and preserves site autonomy. Performance and comparison with other commitment protocols is also mentioned to support the fact that the proposed solution capitalizes on previously proposed protocols while alleviating their drawbacks.

A. PHASE – 1

1) *Let site 'i' wants to commit, then it will issue a token packet, updating its state as*

- S[i] = I (initially for all i, S[i] = N) & will send the token to any site 'j' where S[j] = N
- {either randomly or by using some metric}.
- Site 'j' on receiving the token packet may reply in COMMIT or ABORT.

- If it wants to abort, then update S[j] = A, & send the token back to site 'k';
- where S[k] = I.
- If it wants to commit, then it will update its state S[j] = R, & forward the packet to any site 'k'; where S[k] = N.

2) *If any site 'I' (whose state is N, means it hasn't received any token so far) want to abort, then it can broadcast a message < i, ABORT > at any point of time, & any site 'j' receiving this message will abort.*

- In case site 'j' holds the token, it will send the token back to site 'k'; such that S[k] = I.
- In case site 'j' is the initiator, it will wait for the token, destroy it upon receiving & abort it-self.

B. PHASE – 2

1) *At any site j, if for all i-k; S[i] = R & S[k] = I, the site j will broadcast a message commit,*

- <i, COMMIT >; where S[i] = I, and commit itself.
- On receiving COMMIT message every site will commit.

2) *At site 'i', if token packet is received back, such that site 'i' initiated the token, S[i] = I, then:*

- It will broadcast a message < i, ABORT >, if there is at least one 'A' in the states of sites in the token received.
- It will broadcast a message < i, COMMIT >, if states of all other sites in the token is 'R'.
- Else it will wait for the token initiated by some other site, or for a broadcast saying either commit or abort.
- And finally destroy the token.

NOTE-1: Now, since this approach is sequential to reduce the number of message exchange, it might be slow. So, to improve its speed we can have multiple tokens initiated by multiple sites in the network. This approach will enhance the speed of producing result or reaching the consensus faster.

3) *Any site receiving the token will use Store & Forward manner. It means, it will copy the token and then forward it. Now if the site receives another token initiated by any other site, then, it will compare the number of 'N' in both the tokens (provided, it already has the image of previous token):*

- If number of 'N' is less in the new token, it will update its state and forward the token as per rules, and copy the image of new token replacing the old one.
- If number of 'N' is more in the new token or equal to the last token, then it will send the token back to its initiator, to a site 'k'; where S[k] = I.

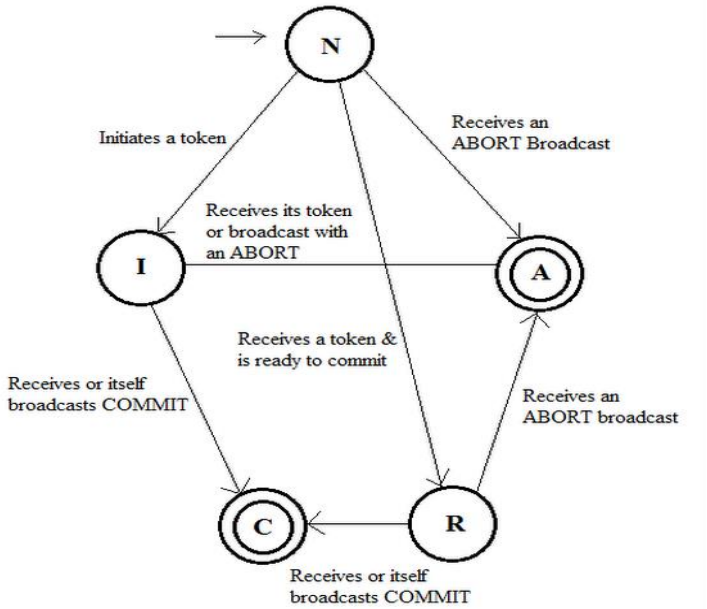
NOTE-2: Any site 'i' which initiated a token will wait for a broadcast or for its token until a certain TIMEOUT is expired.

- If it receives nothing within the pre-decided TIMEOUT period then it will broadcast a message < i, ABORT > and then wait for its token. Now, if it receives the token back after the broadcast, it will destroy it and abort it-self. This case considers that due to site failures, it is not possible to reach a consensus. And if it doesn't receives it token back within (2/n)*TIMEOUT period (where 'n' is the number of sites), then it will abort itself considering that no other site is alive & it is impossible to reach any consensus.
- If the site receives the token back before the TIMEOUT, it will destroy the token and wait for the broadcast, and after if it receives a broadcast, it will process accordingly. But, if it receives a broadcast before its token then:
- If broadcast says < j, COMMIT >, then it will commit immediately.
- But if broadcast says < j, ABORT >, then it will wait for its token to come back.

Waiting for the token by initiator might delay the action performed by the initiator itself. But, this has been adopted to preserve the existence of a token in the network. This ensures that token can only be destroyed by the initiator of that token, no other site can forcefully destroy the token, initiated by any other site.

NOTE-3: If any site 'i' which initiated a token receives a token initiated by any other site and there are exactly (n-1) 'N' states in the token, then it will send the token back to its initiator, after updating its state. This will somehow reduce the redundancy of tokens in the network, which has been initiated by various sites with a common concern of gathering votes for committing the transaction.

IV. STATE DIAGRAM



N– Initial state; when site has neither initialized any token nor received.
A/C – Final state, site will either commit or abort.

An extra state like waiting or prepare can be added before commit, to remove commit from the concurrency set of the states; I or R. But failing in these states may not really affect the consensus, provided the sites don't miss any broadcast. Here, the protocol assumes that every broadcast is always successfully sent and received, as network is reliable. But adding an extra state will add an extra round of message exchange and also affect the latency.

V. PERFORMANCE

Atomic commitment protocols have a great impact on the performance of any distributed database system. Much of the literature focuses on improving performance in failure cases by providing a non-blocking 2PC that streamlines recovery processing at the expense of extra processing in the normal case. This paper focuses on improving performance in the normal case without adding any extra overhead. The performance and applicability of the protocol makes it especially important in the context of environments that are characterized by high volume of short transactions.

Number of Messages (M), Number of Sites (n):

1) *When site 'i' aborts in Phase-1.*

If no token has been initiated so far by any site and any site 'i' wants to abort, then it will broadcast a message <i, ABORT>. And total number of messages will be equal to:

$$M = n - 1 \text{ (one broadcast) } \{\text{Best case}\}.$$

If all the sites want to commit and have initiated the token other than only one site which wants to abort, then it will broadcast a message < i, ABORT >. Then as a consequence the entire tokens in the network will be sent back to their respective initiators and then destroyed. This will somehow increase the number of message exchange. But this will be highly rare, when the site initiating the broadcast will realise that it wants to abort, after (n-1) tokens have been initiated. This can be the worst case for this algorithm, here total number of messages will depend on the number of token initiated and traversed successfully in the network. So, total number of messages will be equal to:

$$M = 3(n - 1) \text{ } \{\text{Worst case}\}.$$

NOTE: To control the number of tokens in the network, any site can initiate its token, only if it hasn't initiated a token already, and it hasn't received any token initiated by some other site. If it has, then it will wait for the broadcast, or it's token to come back.

2) *When site 'i' aborts in Phase-2.*

If site 'i' has initiated the token, and the site receiving the token wants to abort, then it will send the token back to its initiator after updating its state, then initiating site will destroy the token and release a broadcast < 'i', ABORT >. Then total number of messages will be equal to:

$$M = n + 1 \text{ } \{\text{Best Case}\}$$

If site 'i' has initiated the token, and n-th site wants to abort and sends the token back to its initiator after updating its state, then initiating site will destroy the token and release a broadcast < 'i', ABORT >. Then total number of messages will be equal to:
M = 2n - 1 {Worst Case}

3) *When site 'i' commits.*

If site 'i' initiates a token, and all the sites are ready to commit the transaction, and any site 'j' finds the condition to be true:

for all i-k; S[i] = R & S[k] = I;

the site j will broadcast a message < i, COMMIT >; where S[i] = I, and commit itself. Then total number of messages will be equal to:

$$M = 2 (n - 1)$$

A. Comparison with Commit Protocols

Protocol	Message exchange	Degree of blocking
2PC	4(n-1)	High
3PC	5(n-1)	Low
This Protocol	3(n-1) {worst} n+1 {best-abort} 2(n-1) {best-commit}	Very Low

Table. 1: Comparison of Commit Protocols

- In case (n-1) sites want to abort, then in case of classic two phase commit protocol, it will tale about 4(n-1) messages, but in case of the proposed protocol, it will take only n+1 messages.
- In case all sites want to commit, then in case of classic two phase commit protocol, it will tale about 4(n-1) messages, but in case of the proposed protocol, it will take only 2(n-1) messages.

VI. HANDLING FAILURES

Failure handling in distributed computing is a wide area with a significant body of literature that is vastly diverse in methodology and terminology. Failures in distributed systems are partial – some components may fail while others continue to function. Hence failure handling can be difficult. There can be many points where any site may crash or fail, while executing this protocol. Failure handling in this protocol doesnot add any extra overhead, so these strategies will provide ability to the participating sites in the distributed transaction enabling them to roll back a transaction that was not completed.

A site "i" can fail after initializing the token. If any site receives a token, it updates its state from N to either R or A. But in case site receives the token initiated by it-self, the state in the token will be I instead of N. Now, when the site is recovered after the failure it will destroy the token and broadcast <i, ABORT>.

A site "i" can fail holding the token. In that case initiator will broadcast <i, ABORT> after a certain TIMEOUT, assuming that some site has failed and there is a delay in the process of reaching token to all the sites.

Any site "i" which fails either before receiving the token or after forwarding the token, will not affect the consensus. As in the former case, site will receive the token later and in the latter case, site will receive a message saying either to commit or abort. NOTE: This algorithm is un-affected by multiple site failures, if all the failing sites have either not received or forwarded the token (if all failing sites was either in N or R state). However, if the site holding the token fails then, indirectly it becomes the reason to abort. This can be handled by providing a stable storage where sites can keep the image of token received. So that even when they fail, upon recovery they can get the status of token and if the site was last to update the token it will forward the token and continue the process again, until a consensus is reached.

VII. CONCLUDING REMARKS

Considering the increasing needs of today's advanced distributed transaction systems, in terms of efficiency and fault tolerance, this paper proposes an atomic commitment protocol that can drastically reduce the cost of transaction commitment while being non-blocking and decentralized as well.

This paper discusses an atomic commitment protocol which provides thenon-blocking property while requiring the same numberof steps to commit as the 2PC protocol. This protocol provides significant better performances than the classicalnon-blocking 3PC protocol, [5] and using a broadcastnetwork, is even more efficient than the 2PC protocol. [8] Failure scenarios are handled using a uniform consensusprotocol as a termination protocol [9].
Furthermore, this protocol may appear to be thesole protocol that can cope with existing transactional systems, without violating their autonomy and supporting a decentralized architecture. Thiscontribution is further strengthened by the fact thattoday's modern distributed transactional systems are inherentlyheterogeneous, and none of the existing protocols is ableto cope with these systems through their standardinterfaces and most of the existing proposed solutions are all centralized in their architecture. This communication protocol can turn out to be an adequate base for running consensus protocols on a higher level.

VIII. FUTURE SCOPE

Reaching consensus among cooperating processes across different sites on a global state and guaranteeingcomputational progress is a major problem distributed systems supporting cooperative applications. Theproblem of reaching consensus can be identified on different levels within the system where each levelmay provide the adequate protocols. [10]
In the future, a paper about comparing different methods of non-blocking techniques regarding 2PC protocols would be of great benefit for those who will use an atomic protocol for a distributed system. Especially in the performance field, now it is difficult to compare different techniques as existing proposed solutions use different methods to evaluate their work and results.[11]
Most of the proposed solutions in the literature are based on centralized systems in which typically, only the coordinator node has all the information necessary to determine whether a transaction should commit or rollback. Therefore, if the coordinator node fails during a distributed transaction, all the participants in the transaction must wait for the coordinator to recover before completing the transaction. Thus, significant delays may be caused when the coordinator fails. [12]
The massive development of applications on top of distributed database systems makes the definition of efficient atomic commitment protocols (ACPs) a very challenging area. The efficiency of ACP is even more crucial when high transaction throughput has to be supported. Blocking ACPs penalize system resources, which is exacerbated in a multi-database system. Non-blocking ACPs generally incur a higher latency and a substantial communication overhead even during normal execution. [4] This makes them not adapted to today's systems with long mean time between failures. And also their centralized architecture is another issue to be resolved.
Hence, to avoid blocking in basic ACP's some sort of non-blocking technique is necessary, which must support the decentralized architecture to overcome the drawbacks of centralized systems, with a lower latency and small communication overhead. A little enhancement in this protocol can make it useful in parallel systems as well, which can be a giant step in itself.Moreover, further improvements in terms of the numberof distributed operations per second may be achieved byaggregating multiple operations together.

REFERENCES

[1] J. Gray and A. Reuter,Transaction processing: Concepts and Techniques. Morgan Kaufmann, 1993.
[2] P. A. Bernstein, V. Hadzilacos, and N. Goodman Concurrency Control and Recovery in Database Systems, Addison-Wesley, Reading, Massachusetts, 1987.
[3] M. Abdallah and P. Pucheral, A Low-Cost Non-Blocking Atomic Commitment Protocol for Asynchronous Systems, 1999.
[4] M. Abdallah and P. Pucheral, A Non-Blocking Single-Phase Commit Protocol for Rigorous Participants, 1998.
[5] D. Skeen, Non-Blocking Commit Protocols, In Proceedings of ACM-SIGMOD 1982 International Conference on Management of Data, 1982.
[6] R. Guerraoui and A. Schiper, The Decentralized Non-Blocking Atomic Commitment Protocol, 1983.
[7] S. Mullender : Distributed Systems, Addison Wesley, acm Press, 2nd edition, 1993
[8] J. Gray, Notes on Database Operating Systems. In Operating Systems: An Advanced Course, Springer Verlag. 1978.
[9] T. Chandra and S. Toueg, Unreliable failure detectors for reliable distributed systems, In the Proceedings of the 10th ACM Symposium on Principles of Distributed Computing,1991.
[10] Edgar Nett, Reaching Consensus - A Basic Problem in Cooperative Applications, 2007.
[11] P. Lannhult and B. Lindblom, Review of Non-Blocking Two-Phase Commit Protocols, 1998.
[12] V. Manikandan, R. Ravichandran, R. Suresh, F. Sagayaraj Francis, An Efficient Non-Blocking Two Phase Commit Protocol for Distributed Transactions, International Journal of Modern Engineering Research, 2012.