International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2019 IJSRCSEIT | Volume 5 | Issue 2 | ISSN : 2456-3307
DOI : https://doi.org/10.32628/CSEIT1952228

A Survey on Efficient Concurrency Control Algorithm in Distributed Database Systems

Rebecca Nyasuguta Arika, W. Cheruiyot
Jomo Kenyatta University of Agriculture and Technology, Nairobi, Kenya

ABSTRACT

Transaction commit protocols help in reaching an agreement among the participating nodes when a transaction has to be committed or aborted. To initiate an agreement each participating node is asked to vote its decision on the operations on its transactional fragment. The participating nodes can decide to either commit or abort an ongoing transaction. In case of a node failure, the active participants take essential steps such as running the termination protocol to preserve database correctness. This paper sought to investigate the current distributed databases commit protocols such as 2PC and 3PC in order to pin-point their shortcomings. For instance, 2PC suffers from blocking of participant site in case of coordinator failure and increased latency due to forced writes of logs. On its part, 3PC suffers more communication overhead due to extra pre-commit phase. Based on these setbacks, an efficient protocol is suggested towards the end of this paper that it believed to address some of the challenges such as blocking and extra message exchange between communicating nodes.

Keywords : Commit Protocols, 2PC, 3PC, Distributed Databases

I. INTRODUCTION

A distributed database is a single logical database that is spread physically across computers in multiple locations, and these computers are in turn connected by a data communication network [13]. On the other hand, distributed database systems (DDBS) are systems that have their data distributed and replicated over several locations [3]. The commit processing in a Distributed Real Time Database (DRTDBS) can significantly increase execution time of a transaction and as such, designing a good commit protocol is important in these databases.

Reference [17] shows that transactions that miss their deadlines before the completion of processing are aborted while all the successful transactions are committed. The performance of the commit protocol is usually measured in terms of number of transactions that complete before their deadlines. If the transactions run across different sites, it may commit at one site and may drop at another site, leading to an inconsistent transaction. Since transactions in a real time database system have deadlines to process the workloads, they need to process transactions before these deadlines expire. Therefore, according to [4], distributed database systems implement a transaction commit protocol to ensure transaction atomicity.

Reference [18] explained that in distributed databases, in situations where there are no failures, all transactions will complete successfully. However, if a transaction may not complete its execution successfully, such a transaction is said to have aborted. In ensuring the atomicity property, an aborted transaction does not have to effect on the state of the database. This is further supported by [18]. This means that any changes that the aborted transaction made to the database have to be undone. Since the changes caused by an aborted transaction need to be undone, this transaction is said to have rolled back. It is part of the responsibility of the recovery scheme to manage transaction aborts. On the other hand, if a transaction completes its execution successfully, it is said to have committed. A committed transaction is a transaction that has performed updates, transforms the database into a new consistent state, which must persist even if there is a system failure [16].

There are two types of commit protocols used for concurrency control, Two Phase commit protocol (2PC) the Three Phase commit protocol (3PC). In 2PC commit protocol, the sites having more queries become primary site and those which are having fewer queries become secondary sites. There are two phases in two phase commit protocol: the voting phase and the commit phase. During the voting phase, the primary site asks all secondary sites to vote either to commit or to abort after which the secondary sites cast their votes. On the commit phase, based on the votes cast by secondary sites, coordinator decides to commit if all secondary sites votes commit or abort if any of the secondary sites votes to abort and after making decision coordinator notifies the result to all the sites.

The 2PC protocol has a blocking problem in which either the coordinator or some participating site is blocked. The 3PC protocol was introduced as a remedy to this blocking challenge and can therefore be regarded as an extension of 2PC protocol. It introduces an extra phase which ensures the non blocking property of this protocol. The site on which transaction is generated becomes coordinator and other becomes cohorts [5].

II. RELATED WORK

In their paper, [11] proposed a new architecture for 2PC by employing a Backup coordinator, which reduces the transaction blocking considerably. However in worst case, the blocking can occur in the backup coordinator. If such a rare case occurs, the client has to wait until the recovery of either the coordinator or the backup coordinator. In this model, new component called connection manager, keeps on monitoring the coordinator and backup coordinator. Whenever the coordinator fails, the transactions are automatically transferred to the backup coordinator with the help of connection manager and vice versa [6]. In turn connection manager will have a common log file for both coordinator and backup coordinator. Synchronization between them will be achieved with the help of connection manager.

In their paper, [7] proposed a Backup Commit (BC) protocol by including backup phase to 2PC protocol. In this, one backup site is attached to each coordinator site. After receiving responses from all participants in the first phase, the coordinator communicates its decision only to its backup site in the backup phase. Afterwards, it sends final decision to participants. When blocking occurs due to the failure of the coordinator site, the participant sites consult coordinator's backup site and follow termination protocols. In this way, BC protocol achieves non-blocking property in most of the coordinator site failures. However, in the worst case, the blocking can occur in BC protocol when both the coordinator and its backup site fail simultaneously. If such a rare case occurs, the participants wait until the recovery of either the coordinator site or the backup site. BC protocol suits best for DDBS environments in which sites fail frequently and messages take longer delivery time. Through simulation experiments it has been shown that BC protocol exhibits superior throughput and response time performance over 3PC protocol and performs closely with 2PC protocol [14].

III. METHODOLOGY

In this paper, a number of research papers in the field of commit protocols including 1PC, 2PC and 3PC were examined. This facilitated the comparisons of these three most common protocols together with the other protocols that have been proposed over the years by researchers in this field.

A. One phase commit protocol

Reference [2] explains that in one phase commit protocol (IPC), the cost of atomic commit is reduced by eliminating the voting phase of 2PC: two communication steps, together with their associated forced log writes, are thus eliminated. Despite its efficiency, 1PC has been largely ignored in the implementation of distributed transactional systems, mainly due to the strong assumptions made by 1PC protocol designers about transactional processing. For instance, both Coordinator Log (CL) and Implicit Yes-Vote (IYV) protocols assume that participants in a transaction externalize their log records and follow a strict two-phase locking concurrency control. According to [5], these assumptions are unrealistic in most existing transactional systems.

The ideal scenario where 1PC can be utilized instead of 2PC is when there is only a single participant in which short-lived transactions involving only one participant can commit without requiring initial prepare phase. As such, there is no overhead to check whether the participant is prepared to either commit or rollback. As [9] points out that several variations of 1PC protocol have been proposed. The Early Prepare (EP) protocol forces each cohort to enter a prepare state after the execution of each operation. It makes cohort's vote implicitly YES and this protocol exploits the Presumed Commit (PC) as well. Nevertheless, a coordinator may have to force multiple membership records, because the transaction membership may grow as transaction execution progresses. Above all, the main drawback comes from the fact that the log of each operation has to be written in the cohort's log disk per operation, and this leads to a serious disk blocking time. Only if every server has a stable storage so that log forces are free, EP can be considered to be used.

B. Two Phase Commit Protocol

The 2PC is a distributed algorithm used in computer networks and distributed database systems, particularly when simultaneous data updates are to be applied within a distributed database. In this protocol, one node acts as the coordinator (master) and all the other nodes in the network are called participants (slaves). In its first phase, all these participants agree or disagree with the coordinator to commit (vote yes or no) while in the second phase they complete the transaction simultaneously by getting the commit or the abort signal from the coordinator.

Reference [5] explains that available protocols for handling distributed namespace operations such as the two phase commitment protocol are expensive since they require the exchange of a large number of messages between metadata servers as well as synchronous writes to stable storage to log vital information. Moreover, such protocols adopt locking schemes to protect the resource during the operation, which force multiple operations on the same directory to be serialized. This severely impacts the performance of high performance computing applications in typical scenarios such as high rate of file create operations. This is further confirmed by [1] who noted the increased latency due to forced writes of logs.

Reference [3] shows that the 2CP has two types of nodes to complete its processes: the coordinator and the subordinate. The coordinator's process is attached to the user application, and communication links are established between the subordinates and the coordinator. The 2PC goes through two phases with the first phase being the prepare phase, in which the coordinator of the transaction sends a *prepare* message. Here, the coordinator sends a Prepare message along with the transaction to all participants and asks each one of them to cast their vote for commit or abort. If participant can commit the transaction, *Vote-commit* is sent to the coordinator and if participants cannot commit, *Vote- abort* is sent to the coordinator. The second phase is decision-making phase in which the coordinator issues a *commit* message, if all the nodes can execute the transaction, or an *abort* message, if at least one subordinate node cannot execute the required transaction. Decision for commit or abort is taken by the coordinator in this phase. If *Vote-commit* is received from all the participants, then *Global-commit* is send to all the participants and if at least one *Vote-abort* is received, then coordinator send *Global abort* to all those voted for commit. In addition, the coordinator asks for acknowledgement (*Ack*) from participants. If a participant receives *Global-commit*, it commits the transaction and *Ack* is sent to the coordinator. In case participant receives *Global-abort,* it aborts the transaction. Figure 1 shows two-phase commit protocol.

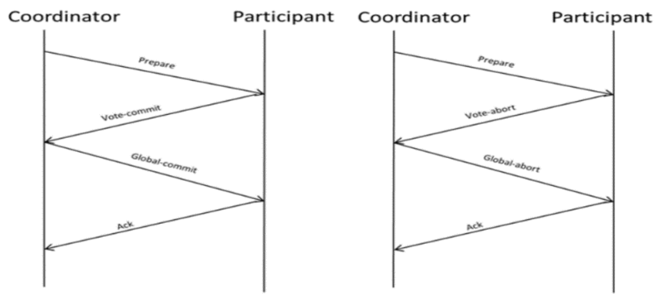

Figure 1 : Two- phase commit process

Two–phase commit protocol ensures atomicity and can handle network failures. But it suffers from blocking of participant site in case of coordinator failure, increased latency due to forced writes of logs and more communication overhead as compared to simple optimistic protocol. Figure 2 presents the state diagram representation of the 2PC protocol. It also shows the set of possible states (and transitions) that a coordinating node and the participating nodes follow, in response to a transaction commit request.

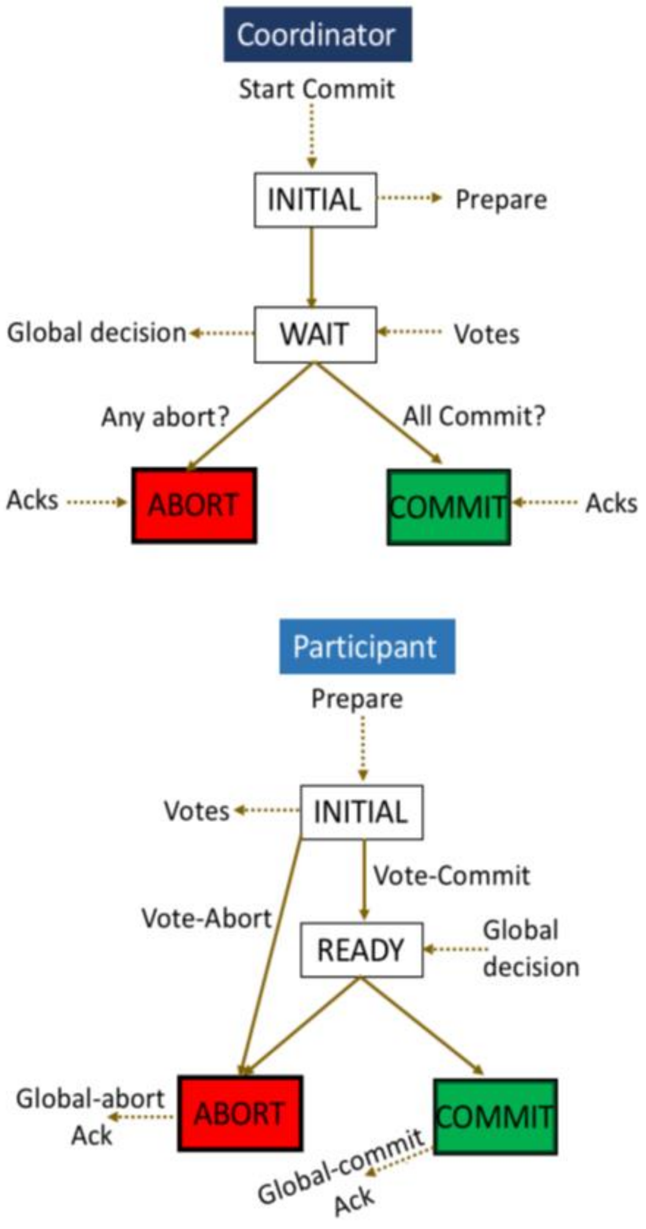

Figure 2 : 2PC Protocol State Diagrams

While the solid lines are employed to represent the state transitions, the dotted lines are utilized to represent the inputs or outputs to the system. The coordinator starts the commit protocol on transaction completion, and requests all the participants to commence the same by transmitting Prepare messages. In case of multiple failures, the 2PC protocol has been proved to be blocking. For instance, if the coordinator and a participant fail, and if the remaining participants are in the READY state, then they cannot make progress (blocked), as they are unaware about the state of the failed participant.

The 2PC protocol has been proved to be blocking under multiple node failures. To illustrate this behavior, a consideration is made of a simple distributed database system with a coordinator C and three participants X, Y and Z. An assumption of a snapshot of the system when C receives Vote-commit from all the participants is made, and hence it decides to send *Global-Commit* message to all the participants. Unfortunately, C fails after transmitting *Global-Commit* message to X, but before sending messages to Y and Z. The participant X on receiving the *Global-Commit* message commits the transaction. Now, suppose X fails after committing the transaction while nodes Y and Z timeout due to no response from the coordinator? These two would be blocked indefinitely, as they require node X to reach an agreement. They cannot make progress, as they neither have knowledge of the global decision nor the state of node X before failure. This situation can be prevented with the help of the three-phase commit protocol [10].

A. Three Phase Commit Protocol

This blocking characteristic of the 2PC protocol endangers database availability, and makes it unsuitable for use with the partitioned databases. The inherent shortcomings of the 2PC protocol led towards the design of resilient 3PC protocol which introduces an additional PRE-COMMIT state between the READY and COMMIT states. This ensures that there is no direct transition between the non-committable and committable states. This simple modification makes the 3PC protocol non-blocking under node failures. Moreover, [12] explained the three phases of 3PC protocol: the voting phase, prepare to commit phase and the decision phase. During the voting phase, the site at which transaction originates becomes coordinator and then it asks the other cohorts to vote to either commit or to abort. The cohorts cast their votes to coordinator and based on the voting done by cohorts, coordinator decides to commit the transactions if all cohorts are in favour of commit. Otherwise it decides to abort even if any of the cohorts is against of committing the transaction. During the p*repare to commit phase,* the coordinator notifies its decision to all cohorts .If the decision is to committing the transaction, then a message “enter into ready to commit stage” is sent to all cohorts. The final phase is the decision phase in which if the decision made by coordinator is to commit the transaction, then it will send *global-commit* to all cohorts and wait for receiving their acknowledgement. After receiving their acknowledgement, it decides to commit the transaction. If the decision made by coordinator is to abort the transaction, then it will send *global-abort* to all sites and abort the transaction. In this protocol the final decision is made after receiving the acknowledgements [15]

Reference [9] shows that 3PC is a non-blocking protocol, in which a new state called *pre-commit* is introduced for the coordinator in. The coordinator gets to this *pre-commit* state only if all other participants have voted to commit (yes vote). In case this state is not reached, the participants abort and release the blocked resources after a specific time. When the coordinator gets to the *pre-commit* state, there is only one option to abort the transaction and that is a timeout, which corresponds to a failure of a participant, otherwise the transaction gets completed with an acknowledgement from the participants. It is also possible that the coordinator fails at this state; even then it will proceed for global commit [8].

The 3PC avoid blocking by introducing additional round of message exchange and delaying the prepared state until processes receive pre-commit message. Unlike 2PC, 3PC does not immediately commit if all participants send vote-commit. Instead, the coordinator sends out prepare-to-commit message, on receiving this message participants enter into Pre-commit state and send an acknowledgement. After receiving acknowledgement from all participants, coordinator sends commit and participants commit the transaction as shown in Figure 3.

However, the 3PC protocol acts as the major performance suppressant in the design of efficient distributed databases. It can be easily observed that the addition of the PRE-COMMIT state leads to an extra phase of communication among the nodes. This violates the need of an efficient commit protocol for geo-scale systems.

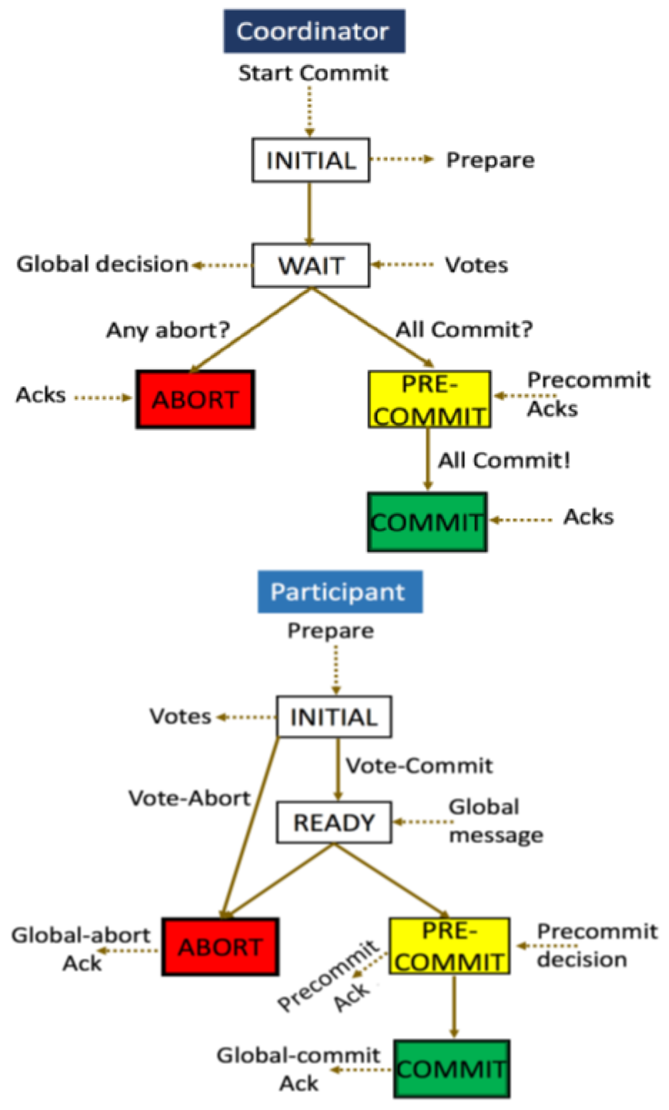

Figure 3 : Three-Phase Commit Protocol

In the first phase, the coordinator and the cohorts, perform the same set of actions as in the 2PC protocol. Once the coordinator checks all the votes, it decides whether to abort or commit the transaction. If the decision is to abort, the remaining set of actions performed by the coordinator (and the cohorts) are similar to the 2PC protocol. However, if the coordinator decides to commit the transaction, then it first transmits a Prepare-to-Commit message, and adds a pre-commit entry to the log. The cohorts on receiving the Prepare-to-Commit message, move to the PRE-COMMIT state, add a corresponding pre-commit entry to the log, and acknowledge the message reception to the coordinator. The coordinator then sends a Global-Commit message to all the cohorts and the remaining set of actions are similar to the 2PC protocol.

IV. PROTOCOLS COMPARISONS

Table 1: Comparison Table for the Various Commit Protocols

Parameters	2PC	3PC
Atomicity	Violates the atomicity at the time of multiple site failures	Violates the atomicity at the time of multiple site failures
Message exchange	Has $4(n-1)$ messages exchange	Has $5(n-1)$ messages due to extra phase PRE-COMMIT
Latency	Medium latency	High latency
Blocking	High blocking under multiple node failures	Non- blocking under node failures by adding extra (PRE-COMMIT) phase
Communication overhead	Medium	High
Log writes	$2n$ log writes	$2n$ log writes
Complexity	Less complex and less costly to implement	More complex and costly to implement
Performance	More performance compared to 3PC	Less performance compared to 2PC

As compared to 2PC protocol, BC protocol requires extra messages and time duration, essentially to communicate with the backup site. However, as compared to 3PC protocol, independent of number of participants, BC protocol requires only two messages and fixed time duration during the second phase. In BC protocol, the latency during the second phase is considerably reduced as compared to 3PC protocol. In addition, by making the nearby site to the coordinator as the backup site, the latency can be minimized. This brings the performance of BC protocol close to 2PC protocol by achieving non-blocking property in most of the coordinator failures. The 2PC is blocking because a transaction's progress is curtailed due to the coordinator's failure when the participant is in the ready-to-commit state. On the other hand, 3PC is said to be non-blocking but both of these protocols violate the important property of atomicity at the time of multiple site failures for 3PC and single site failure for 2PC.

The key difference between the 2PC and 3PC protocol is the PRE-COMMIT state, which makes the latter non-blocking. The design of 3PC protocol is based on the Skeen's design of a non-blocking commit which dictate that: no state should be adjacent to both the ABORT and COMMIT states, and that no non-committable state should be adjacent to the COMMIT state. These requirements motivated Skeen to introduce the notion of a new committable state (PRE-COMMIT) to the 2PC state transition diagram. The existence of PRE-COMMIT state makes the 3PC protocol non-blocking.

The above mentioned multi-node failure case does not indefinitely block the nodes Y and Z which are waiting in the READY state. The nodes Y and Z can make safe progress by aborting the transaction as they are assured that the node X could not have committed the transaction. Such a behavior is implied by the principle that no two nodes could be more than one state transition apart. The node X is guaranteed to be in one of the following states: INITAL, READY, PRE-COMMIT and ABORT, at the time of failure. This indicates that node X could not have committed the transaction, as nodes Y and Z are still in the READY state. In 3PC, the coordinator sends the Global-Commit message after it transmits Prepare to- commit message to all the nodes. Interestingly, if either of nodes Y or Z is in the PRE-COMMIT state, then they can actually commit the transaction. However, it can be easily observed that the non-blocking characteristic of the 3PC protocol comes at an additional cost, an extra round of handshaking.

The 2PC and 3PC protocols can also be compared using six parameters: blocking, message exchanges, communication overhead, log writes, complexity and performance. In terms of blocking, 2PC protocol causes blocking of participants site when coordinator site fails while 3PC avoid blocking by adding an extra phase called Pre-commit. Concerning message exchanges, taking n to be the number of participants, 2PC has $4(n-1)$ messages exchange comprising of $n-1$ messages in Vote-request, $n-1$ in local decision for commit or abort, $n-1$ in Global-commit/Global-abort and $n-1$ messages exchange for ack. 3PC commit protocol causes $5(n-1)$ messages to exchange as compared to $4(n-1)$ in 2PC. The extra $n-1$ message exchanges in 3PC are due to extra phase in 3PC.

In terms of communication overhead, 3PC has more communication overhead due to an extra phase as compared to 2PC protocol while for the case of log writes; both 2PC and 3PC have $2n$ log writes. Considering complexity, 3PC protocol is more complex and costly to implement compared to 2PC protocol while in terms of performance, 3PC protocol has more message exchanges which result in less performance as compared to 2PC. However, 3PC's performance is superior to that of 2PC in case of the failure of coordinator site.

V. CHALLENGES OF THE CURRENT COMMIT PROTOCOLS

The literature reviewed has pointed out that there are two problems with the Two-phase Commit Protocol, which include blocking and state inconsistency. The 2PC protocol goes to a blocking state by the failure of the coordinator when the participants are in an uncertain state. The participants keep locks on some resources until they receive the next message from the coordinator after its recovery. In most cases, this happens due to coordinator's failure when the participant is in the ready-to-commit state. On the other hand, state inconsistency crops in when its global state vector contains both the commit and abort states. This inconsistency can be observed using a state vector, particularly when the participant is at its pre-commit state and fails. The coordinator shows the committed state after sending *commit* message but for the failed participant the protocol is declared non-resilient for assigning new state.

The 2PC protocol is blocking under multiple failures and although 3PC addresses this problem, the database community is still reluctant to use the 3PC protocol, as it acts as a scalability bottleneck in the design of efficient transaction processing systems.

The 3PC protocol is problematic only when there are multiple sites failures. As an illustration, suppose the coordinator is in pre-commit state and fails just after sending a *commit* message and the slave also fails just before or after receiving this message. By its failure, the slave moves to the aborted state but according to the protocol specifications, the coordinator goes to the committed state, either it fails or receives acknowledgement. Hence, the coordinator moves to the committed state without receiving acknowledgement and the failed slave moves to the aborted state without sending the acknowledgement. In this way, coordinator and participant show different final states due to their failures.

VI. PROPOSED PROTOCOL

Based on the shortcomings noted in the current commit protocols, this paper proposes a commit protocol with a number of salient features to address these setbacks. To start with, a participant node cannot make a direct transition from the INITIAL state to the ABORT state. Secondly, the cohorts, irrespective of the global decision, always forward it to every participant. Thirdly, if the cohorts receive global decision from other participants, they need not wait for message from the coordinator. In addition, there will be some hidden states, Transmit-A and Transmit- C, only after which a node aborts or commits the transaction respectively.

The suggested protocol will be able to first count the number of abort votes given from participants and if this number is less than a given threshold, then for each one of these participants, the transaction should be executed for second time by sending a *prepare message*. In cases where majority of the participants will have voted commit, this message will not be sent to all the participants at a later time to execute this transaction.

Another feature of the proposed protocol is the Transaction Information Table (TIT) that will have three fields: Transaction Number, Value and site ID. Each transaction will have a unique number to identify it while the value field for each site will be either *complete* (if the transaction commit at that site) or *incomplete* (if the transaction abort at that site). On the other hand, each site will have a unique number for identification purpose.

The message passing process will begin by having the coordinator send message (including ID and transaction number) to *i*nconsistent site asking the site to complete the transaction. The *inconsistent* site will then forward this message (including transaction number) to its nearest site for updating. Finally, the inconsistent site will send this message (together with transaction number) to coordinator after completing the transaction.

The 3PC protocol acts as the major performance suppressant in the design of efficient distributed databases. However, it was observed that the addition of the *pre-commit* state leads to an extra phase of communication among the nodes. This violates the need of an efficient commit protocol for geo-scale systems. The proposed protocol will leverage 2PC and 3PC to achieve non-blocking characteristics. Two design issues will be important in this respect: to delay the commitment of updates to the database until the transmission of global decision to all the participating nodes, and the secondly to induce message redundancy in the network. Message redundancy will be introduced by ensuring that each participating node forwards the global decision to all the other participants (including the coordinator) before they commit.

The proposed protocol will be initiated by the coordinator node by sending the prepare message to each of the cohorts and shifting to the ready state. When a cohort receives the prepare message, it will send its decision to the coordinator, and moves to the ready state. On receiving the responses from each of the cohorts, the coordinator will first transmit the global decision to all the participants, and then commits (or aborts) the transaction. Each of the cohorts on receiving a response from the coordinator, first forward the global decision to all the participants (and the coordinator), and then commit (or abort) the transaction locally.

To facilitate recovery during node failures, multiple log entries will be introduced. Since this protocol permits the coordinator to commit as soon as it has communicated the global decision to all the other nodes, the coordinator does not need to wait for the acknowledgments. In case a node timeouts while waiting for a message, it executes the termination protocol.

VII. CONCLUSION

This paper sought to investigate the challenges of the current commit protocols in a distributed database environment. It was noted that the 2PC protocol suffers from blocking of participant site in case of coordinator failure, increased latency due to forced writes of logs and more communication overhead as compared to simple optimistic protocol. On its part, the 3PC protocol's addition of the PRE-COMMIT state leads to an extra phase of communication among the nodes. This violates the need of an efficient commit protocol for geo-scale systems. As such, an efficient protocol that is both non-blocking and utilizing less message exchanges is suggested. Future work in this area will involve the implementation and evaluation of the proposed protocol in real distributed database environment.

VIII. REFERENCES

[1]. Ashay. M. Yogesh, R. "Concurrency Control and Security Issues in Distributed Database Systems," International Journal of Engineering Development and Research, Vol.4, no. 2, pp. 40-44, 2016.

[2]. Chirag, N. "Enabling One-Phase Commit (1PC) Protocol for Web Service Atomic Transaction (WS-AT),". Master's Thesis, Pp. 1-75, 2014.

[3]. Christos, P. "An Algorithm for the Distributed Two-Phase Commit Protocol," International Journal of Computer Science and Information Technology Research. Vol. 4, Issue 3, pp. 233-244, 2016.

[4]. Fadia, A., Ahmed. K, Khalid. K, "A survey of Commit Protocols in Distributed Real Time Database Systems". International Journal of Computer Trends and Technology. Vol. 31, no. 2, pp. 61-66, 2016.

[5]. Giuseppe, C., Matthias. G., Sai. N, Andr´e. B., "One Phase Commit: A Low Overhead Atomic Commitment Protocol for Scalable Metadata Services. IEEE International Conference on Cluster Computing Workshops. Pp. 16-24, 2012.

[6]. Gupta, K, Sheetlani, J, Gupta, D. Shukla,B., "Concurrency Control and Security Issues of Distributed Databases Transaction". Research Journal of Engineering Sciences. Vol 1, pp. 70-73. Aug. 2012.

[7]. Krishna, P., Masaru, K., "Reducing the Blocking in Two-phase Commit Protocol Employing Backup Sites," Institute of Industrial Science, The University of Tokyo. pp. 1-10, 2016.

[8]. Mandeeep, K., Harpreet, K. "Concurrency Control in Distributed Systems".International Journal of Advanced Research in Computer Science and Software Engineering. Vol.3, Issue 7, pp1443-1446. July.2013.

[9]. Manoj, K. "Commit Protocols in Distributed Database System," A Comparison. International Journal for Innovative Research in Science & Technology. Vol. 2, Issue 12, pp. 277-281. 2016.

[10]. Monaj,K., Vinah,K., Tiwari, A. "Security and Concurrency Control in Distributed Database Systems," International Journal of Scientific Research and Management. Vol2, Issue12, pp 1439-1844. Dec.2014

[11]. Manikandan, V., Ravichandran, R., Suresh, R., & Sagayaraj F. "An Efficient Non Blocking Two Phase Commit Protocol for Distributed Transactions. International Journal of Modern Engineering Research. Vol.2, Issue.3, pp.788-791. 2015.

[12]. Mohit, K., Anjali S., Arjun S., Sachin ,S, " An Extension of Modified Three Phase Commit Protocol for Concurrency Control in Distributed Systems," International Journal of Research Studies in Computer Science and Engineering. Volume.1, Issue 4, pp. 38-45. Aug. 2014.

[13]. Mohit, R., Manish, L, "A survey on Distributed Operating Systems," International Journal of Innovative Research in Technology. Vol. 1, Issue 5, pp128-131. 2014

[14]. Paul, K., Charity, W., Newton, K., "Concurrency Control in Distributed Systems" Conference Paper. Pp 1-3. 2014.

[15]. Rakesh, K., Ramnyadevi, R., Vijaya, C., "Resolving Atomic Transaction Issues in Web Services- Business Activities," International Journal of Informative Research, Engineering and Technology. Vol. 3, Issue 3, pp 1680-1683. Mar. 2014.

[16]. Sonali, B., Paswan,R., "A survey on Recommender System Using Distributed Framework," International Journal of Science and Research. Vol. 5, Issue 1, pp1967-1970. 2013.

[17]. Song, Jiajia, "Computer Network Performance Optimization Approaches Based on Distributed Systems with Cloud Computing Environment," International Journal of Science and Research. Vol. 5, Issue 2, pp 733-735. Feb. 2016.

[18]. Suyash, G., Mohammad S., "EasyCommit: A Non-blocking Two-phase Commit Protocol," Proceedings of the 21st International Conference on Extending Database Technology. Pp. 157-168. 2018.

[19]. Tablez, Q, "An efficient Approach for Concurrency Control in Distributed Systems," Indian Streams Research Journal. Vol.3, Issue 9, pp5-8. Oct. 2013.

Cite this article as : Rebecca Nyasuguta Arika, W. Cheruiyot, "A Survey on Efficient Concurrency Control Algorithm in Distributed Database Systems", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 5 Issue 2, pp. 1176-1185, March-April 2019. Available at doi : https://doi.org/10.32628/CSEIT1952228
Journal URL : http://ijsrcseit.com/CSEIT1952228