International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 10, October- 2014

Atomic Commit in Distributed Database Systems: the Approaches of Blocking and Non-Blocking Protocols

[1]Olowookere Toluwase Ayobami, [2]Asagba Prince Oghenekaro and [3]Obasi Chinedu Kingsley
[1, 2, 3]Department of Computer Science,
University of Port Harcourt, Choba,
Port Harcourt, Nigeria.

***Abstract* - In distributed database systems, the primary need for commit protocols is to maintain the atomicity of distributed transactions. Atomic commitment issue is of prime importance in the distributed system and the issue becomes more necessary to deal with if some of the sites participating in the execution of the transaction commitment fail. Several atomic commit protocols have evolved to terminate distributed transactions. This paper presents an overview of a distributed transaction model, and a description of the two phase commit (2PC) protocol (which is blocking) and the one phase (1PC) commit protocols (which is non-blocking). This paper further examines the assumptions of these commit protocols in their bid to addressing the atomic commitment issue in distributed database systems. By restricting possible encountered failure to site failure, drawbacks in the assumptions of these atomic commit protocols were identified, which clearly show that the non-blocking protocol studied addresses the drawbacks of the widely used blocking protocol, 2PC, but in itself is no messiah (as it also constitutes drawbacks in practice). This work will spur other researchers to a more vigorous reconsideration of the 1PC non-blocking protocol.**

Keywords— Atomic commit protocols,Blocking,Distributed Database Systems, Stable database

I. INTRODUCTION

The execution of transactions in a distributed database system (DDBS) involves accessing data located at different sites. A *transaction* is a set of related operations that form a *logicalunit of work*. The main idea of a transaction is its *indivisibility*, i.e. either all the operations of the transaction are permanently performed or none of them is, and its *partial* results are not visible to other transactions. Transaction semantic is defined traditionally by the ACID properties: Atomicity, Consistency, Integrity, and Durability [9, 13]. The Atomicity property, 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 data (the transaction is said to be committed), or it fails and it has no effect at all (the transaction is aborted). Consistency means that the transaction does not violate the integrity constraints of accessed shared data, while Isolation means that the intermediate effects of a transaction are not visible to concurrent transactions (Isolation has been formalized through the theory of serializability). Durability means that the updates of a committed transaction are permanent, e.g., stored on a stable storage that sustains failures [7].

Distributed transaction processing systems are designed to facilitate transactions that span heterogeneous, transaction-aware resource managers in a distributed environment [1]. A distributed transaction will consist of a local transaction at each of the sites participating in the distributed transaction (such that if any local transaction aborts, the distributed transaction aborts and if all local transaction commits, the distributed transaction commits). The execution of a distributed transaction requires coordination between a global transaction management system and all the local resource managers of all the involved systems.

In order to ensure atomicity property of a distributed transaction, all local sites participating in the transaction must coordinate their actions so that they either unanimously abort or unanimously commit the transaction [7], and so the transaction's effects either persist at all sites involved in the transaction or are obliterated from them as if the transaction has never existed. This is achieved by employing an *atomic commit protocol* (ACP) that executes a commit or an abort operation across multiple sites as a single logical operation [13].

The Atomic commit protocols (ACP) terminate distributed transaction by addressing the distributed execution of the abort and commit commands. A transaction always terminates, even when there are failures. If the transaction can complete its tasks successfully, we say that thetransaction *commits*. If, on the other hand, a transaction stops without completing itstasks, we say that it *aborts*. Transactions may abort for a number of reasons; a transaction may abort itself because of a condition that would prevent it from completing its tasks successfully. Additionally, the database management system (DBMS) may abort a transaction due to, for example, deadlocks or other conditions. When a transaction is aborted, its execution is stopped and all of its already executed actions are undone by returning the database to the state before their execution, a situation known as rollback. The importance of commit is twofold. The commit command signals to the DBMSthat the effects of that transaction should now be reflected in the database, therebymaking it visible to other transactions that may access the same data items. Second,the point at which a transaction is committed is a "point of no return." The results ofthe committed transaction are now permanently stored in the database (or *stable database* as used herein) and cannot be undone [8].

The two Phase Commit protocol, (2PC) is one of the atomic commit protocols used in the atomic commitment. Unfortunately this protocol is blocking in some failure scenarios, for example, when the initiating coordinating site fails, and at least, one participating site is waiting for the final decision. The one Phase Commit protocol (1PC) described here is non-blocking. A protocol is non-blocking if it permits a transaction to terminate at the operational sites without waiting for recovery of the failed site [8]. It is commitment protocol that ensures that at least some sites of a multi-site transaction do not block in spite of any single failure [3]. This would significantly improve the response-time performance of transactions.

The objective of this paper is the exposition of the paradigms employed by the blocking 2PC protocol and the non-blocking 1PC protocol in addressing the commitment problem.

The rest of this paper is structured as follows. In section II, we overview a model of a distributed transactional system. Section III describes the two-phase commit (2PC) protocol, while section IV discusses the assumptions of the one phase commit (1PC) protocol. In section V, the paper concludes.

II. A CONCEPTUAL MODEL OF TRANSACTIONAL SYSTEM

This paper considers a distributed database system composed of a finite set of sites completely connected through communication channels. Each site has its local resource manager and transaction manager. We adopt a common abstraction: we assume that at the originating site of a distributed transaction, there is a coordinator process and at each site where the transaction executes, there are participant processes. Thus, the ACP protocols are implemented between the coordinator and the participants.

In a distributed database system, data are typically stored in disjoint manners at different sites [13]. This distribution of data is transparent to a distributed transaction that accesses data by submitting database operations to its coordinator. When a coordinator receives an operation on a particular data item, it sends the operation to the appropriate site for execution. If the coordinator receives an abort request from the transaction, it sends an abort request to all the participants which it has earlier involved in the transaction, i.e., the sites participating in the execution of the transaction. On the other hand, when the coordinator receives a commit request from the transaction, it initiates an atomic commit protocol (ACP).

As stated earlier, a distributed transaction will consist of a local transaction at each of the sites participating in the distributed transaction. The coordinator reaches a global termination decision regarding a transaction according to two rules that govern its decision, which, together, are called the global commit rule:

1. If even one participant votes to abort the transaction, the coordinator has to reach a global abort decision (i.e., if any local transaction aborts, the distributed transaction aborts).
2. If all the participants vote to commit the transaction, the coordinator has to reach a global commit decision (if all local transactions commit, the distributed transaction commits).

Here are the elements involved in our system model;

Resource Manager: The resource manager (RM) is a database management system (DBMS), such as Oracle or SQL Server. Resource manager is Responsible for maintaining and recovering its own resource (the database).

Transaction Manager: The transaction manager (TM) is responsible for coordinating the operations of its local resource manager. (A transaction manager may also act as a transaction coordinator in a circumstance that it starts a transaction).

Transaction Coordinator: The transaction coordinator (CTM) is the transaction manager of the site where an application initiates the distributed transaction. The transaction coordinator orchestrates the distributed transaction by communicating with its local resource manager and with the TMs of the remote sites that are to participate in the transaction.

The diagram in Figure 1 shows our conceptual transactional system model.

III. DESCRIPTION OF THE BASIC TWO PHASE COMMIT PROTOCOL (2PC)

The two-phase commit (2PC) is a very simple and elegant protocol that ensures the atomic commitment of distributed transactions. It is a set of rules that extends the effects of localatomic commit actions to distributed transactions by insisting that all sites (participants) involved in the execution of a distributed transaction agree to commit the transaction before its effects are made permanent [8].

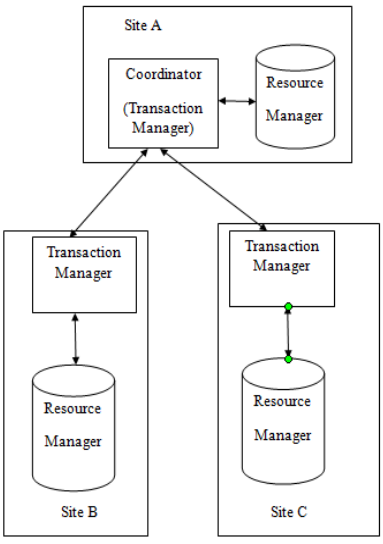

Figure 1: A High Level Conceptual Transactional System Model.

Simply, the 2PC guarantees that every single transaction in a distributed system is executed to its completion or none of its operations is performed, i.e., all-or-nothing.

Ensuring atomicity in a distributed system requires a *transactioncoordinator*, which is responsible for the following;

i. Starting the execution of the transaction.
ii. Breaking the transaction into a number of sub-transactions, and distributing these sub-transactions to the appropriate sites for execution.
iii. Coordinating the termination of the transaction, which may result in the transaction being committed at all sites or aborted at all sites [12].

Just as its name indicates, 2PC [6] is formed of two phases, namely a *Voting phase* (phase 1) and a *Decision phase* (phase2) as shown in Figure 2, the basic two phase commit protocol. During the voting phase, the coordinator of a distributed transaction requests all the sites participating in the transaction's execution to prepare to commit by sending a "Prepare" message, whereas, during the decision phase, the coordinator either decides to commit the transaction if all theparticipants are prepared to commit ("vote-commit"), or to abort if any participant has decided to abort ("vote-abort"). If a participant has decided a "vote-commit", it can neither commitnor abort the transaction at this stage until it receives the final decision from the coordinator [13].

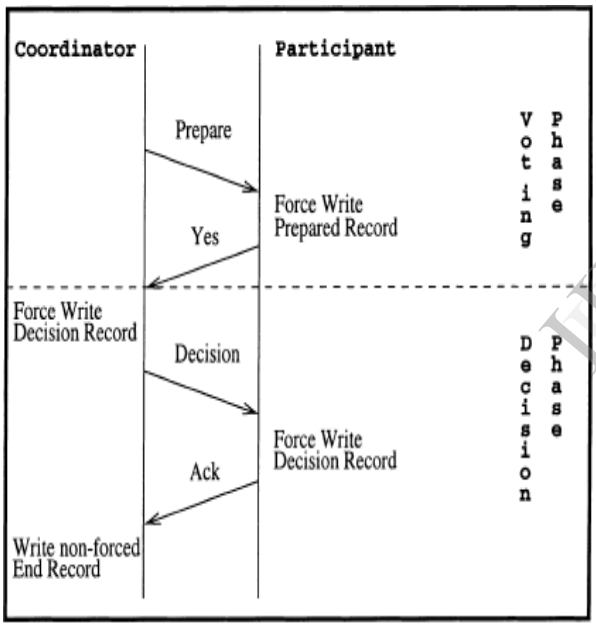

Figure 2: The basic two phase commit protocol. (Source: Ozsu, et al., 2011)

In details, as shown in Figure 3, the 2PC protocol actions [8], the coordinator writes a *begin_commit* record in its log (it must be a forced write, because the coordinator must have a record of the transaction in its log prior to any of the participants), then sends a "prepare" message to all participant sites, and enters the WAIT state. When a participant receives a "prepare" message, it checks if it could commit the transaction. If so, the participant writes a *ready* record in its log, sends a *"vote-commit"* message to the coordinator, and enters READY state; otherwise, the participant writes an *abort* record and sends a *"vote-abort"* message to the coordinator. If the decision of the site is to abort, it can forget about that transaction, since an abort decision serves as a veto (i.e., unilateral abort). After the coordinator has received a reply from every participant, it decides whether to commit or to abort the transaction. If even one participant has registered a negative vote, the coordinator has to abort the transaction globally. So it writes an abort record, sends a "global-abort" message to all participant sites, and enters the ABORT state; otherwise, it writes a commit record, sends a "global-commit" message to all participants, and enters the COMMIT state. The participants either commit or abort the transaction according to the coordinator's instructions and send back an acknowledgment and releases all the resources held by the transaction (i.e., releases the locks held by the transaction, removes the transaction's control block from its table, etc.). At the reception of the acknowledgement, the coordinator terminates the transaction by writing an *end_of_transaction* record in the log [8, 13].

A. *Implementing the basic 2pc in the conceptual transactional system model*

When an application starts a distributed transaction, the TM on the same node becomes the CTM. Following are the steps that are involved in consummating the distributed transaction.

a. The CTM first checks that the TM software is running on all the nodes participating in the transaction. If the TM software is not running, the CTM returns an error and does not start the distributed transaction.

b. If all the TM's are available, the CTM generates a distributed transaction identifier and associates the identifier with all the participants in that particular transaction. When the application is ready to commit all the changes to the RMs involved in the distributed transaction, all the sites in the transaction must execute both phases of the two-phase commit protocol, the *voting* phase and the *decision* phase.

i. During the voting phase, the CTM asks each RM participating in the transaction whether or not it is prepared to commit the transaction. If the TC receives a "vote-commit" response from *all* the RMs, the CTM instructs the participants in the transaction to enter its decision phase.

ii. During the commit phase, the CTM instructs the RM to make permanent changes to its data, i.e. to commit the changes. The RM then commits the changes and the transaction is completed.

In failure scenarios, more precisely; when the coordinator fails, and at least, one participant keeps waiting for the final decision of the coordinator, such scenario depicts the 2PC as blocking, this means that participants cannot terminate the transaction (neither commit the transaction, nor abort it) pending the recovery of the coordinator.

Although 2PC is the most widely used A*tomic Commit Protocol* (ACP), it has two major drawbacks. First, it is

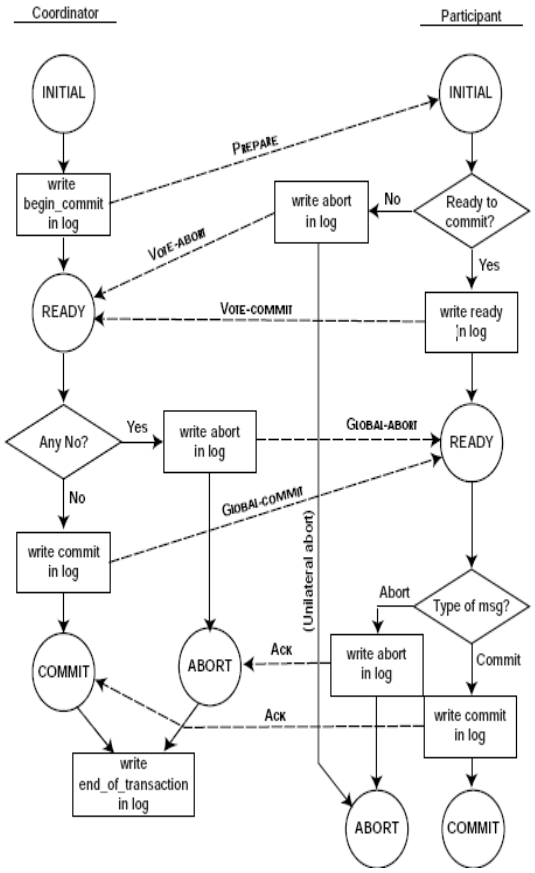

Figure 3: The 2PC protocol actions. (Source: Ozsu, et al., 2011)

blocking: as earlier stated in this article, if the coordinator crashes between the *voting* 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 time delay (two phases to commit) in the system even in the absence of failures. This makes 2PC not adequate to today's highly reliable distributed platforms [12].

The next section discusses one phase commit protocol (1PC) as it addresses some major drawbacks of the 2PC.

IV. DESCRIPTION OF THE BASIC ONE PHASE COMMIT PROTOCOL (1PC)

In section 3 we have discussed the ideas of the 2PC protocol and it was emphasized that the 2PC protocol can guarantee transaction atomicity by walking through two phases (*voting* phase and *decision* phase) and by logging state information as well as data item updates to stable storage, it is however plagued with its blocking activity.Several optimized protocols and non-blocking protocols have been proposed. Optimized protocols generally violate site autonomy while non-blocking protocols are inherently more costly in time and increase communication overhead. Here we discuss the one phase commit (1PC) protocol which reduces the number of these phases from two to one, thus minimizing the communication overhead introduced by the 2PC protocol and the number of forced log writes. This is achieved by cutting off the voting phase and by piggybacking the transaction execution onto the decision/commit phase. Indeed, the aim of the voting phase is to collect the PREPARED messages by the *coordinator* from all the involved *participants* and to allow the *coordinator* to make a decision about what becomes of the distributed transaction.

By eliminating the need for votes, 1PC protocol indeed achieve better performances than 2PC protocols. As have been observed, the basic assumption underlying 1PC is that a participant does not need to vote. This does not however mean that the transaction's outcome is known in advance. *Commit* is decided if all operations have been acknowledged and no failure occurs, and *Abort* might be decided otherwise. However, in most cases (i.e., commit cases) the coordinator just has to send a single message to the participants asking them to commit.

Unlike 2PC, participants here do not vote. In other words, 1PC does not take care of also ensuring the ACID properties of the corresponding local branches of the transaction. This means that before triggering the commit protocol, the coordinator makes sure that these properties are locally ensured at all participants. This means that the coordinator acts as a liberal dictator and makes sure that no participant can have any tenable reason to "vote-abort".

This observation leads to better understanding of the assumptions usually made (explicitly or implicitly) by 1PC protocols. More precisely:

1. 1PC protocol [1, 14] assumes that all the transaction operations have been acknowledged (i.e., all operations have been successfully executed till completion) before the protocol is launched. This means that the Atomicity of all the local transaction branches (i.e., local Atomicity) is already ensured at commit time.

2. 1PC protocol assumes that integrity constraints are checked after each update operation and before acknowledging the operation. Thus, Consistency is ensured for all the local transaction branches at commit time (e.g., the possibility of discovering, at commit time, that there is not enough money for a bank account withdrawal is excluded).

3. 1PC protocols assume that all participants serialize their transactions using a pessimistic concurrency control protocol that avoids cascading aborts (i.e., strict two phase locking [3]). In this context, a transaction that executes successfully all its operations can no more be aborted due to a serialization problem. This actually means that serializability (Isolation) of all the local transaction branches is already ensured at commit time (e.g., optimistic concurrency control protocols that check serializability at commit time are excluded).

4. 1PC protocols assume that, at commit time, the effects of all the local transaction branches are logged on stable storage, and hence the Durability property is ensured [15]. In the Coordinator Log protocol, participants do not maintain their updates in a local stable log. Instead, they send back, within the acknowledgment message of every update operation, all the log records (undo and redo log records) generated during the execution of the operation. The coordinator is thus in charge of logging the transaction updates before performing the commit protocol (local log externalization). To recover from a crash, a participant asks the coordinator for the log records it needs to reestablish a consistent state of its database.

A. How come the Non-Blocking 1PC?

Several solutions have been proposed to eliminate the voting phase of the 2PC. The *early prepare* protocol [10] forces each participant to enter in a *prepare* state after the execution of each operation. A participant is thus ready to commit a transaction at any time, thereby making its vote implicit. The main drawback comes from the fact that each operation has to be registered in the participant's log on disk, thus introducing a blocking I/O. The *coordinator log* protocol [10] exploits the *early prepare* idea and avoids the blocking I/O on the participants by centralizing the participant's log on the coordinator. However, this violates site autonomy by forcing participants to externalize their log records. More recently, the IYV (*implicit yes-vote*) protocol [13] has been proposed to exploit the performance and reliability properties of future gigabit-networked database systems. IYV capitalizes on the *early prepare* and *coordinator log* protocols. Participants in a transaction pass in the acknowledgment messages their redo log files and read locks to the coordinator. Thus the coordinator can forward recover a transaction on failed participants. Although well adapted to gigabit-networked DBMSs, this protocol (i) does not preserve site autonomy by forcing participants to externalize logging and locking information, (ii) puts strong hypothesis on the network bandwidth and (iii) increases the probability of blocking since a coordinator crash that occurs at any time during the transaction processing will block the participants until the coordinator's recovery.

A number of non-blocking commit protocols have been proposed in the literature. The simplest is the *three phase commit protocol* (3PC) [8]. 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.

In [10], Pucheral et al. proposed an atomic commitment protocol, noted NB-SPAC (Non-Blocking Single-Phase Atomic Commit) protocol. NB-SPAC has a *low latency* (one phase to commit), is *non-blocking* and preserves *site autonomy*. During normal execution as well as in case of one or more site failures, a transaction is committed in a single phase under the assumption that participating DBMSs are ruled by a rigorous concurrency control protocol. This assumption is exploited to eliminate the voting phase of the 2PC. Non-blocking is achieved by using a reliable broadcast primitive to deliver the decision messages. Finally, NB-SPAC preserves site autonomy by exploiting techniques introduced in multidatabase systems to recover from failures.

B. Implementing the 1PC (NB-SPAC) in the Conceptual System model

TM directly receives the decision of the CTM for a transaction after the transaction's last operation invocation. The CTM informs the participants in a transaction (TMs) of its decision through a simple broadcast primitive. Participant (i.e., TM) *decides* on a transaction when it delivers the decision message to the local DBMS, whereas a local DBMS decides on a transaction when it executes the decision received from its TM. Local DBMSs eventually conform to the decision of their corresponding TM even in the case of failures. A key step in the NB-SPAC is the dissemination of the decision message to all the participants by the CTM. In order to achieve non-blocking, the NB-SPAC assumes a reliable (broadcast) communication between the sites. Reliable broadcast guarantees the following properties:

Uniform Agreement: if any Participating TM, correct or not, delivers a message m, then all correct Participating TMs will eventually deliver m.

Timeliness: There exists a known constant Δ such that if the broadcast of a message *m* is initiated at time t, then no participant receives *m* after time t+Δ.

The reliable broadcast, noted R_broadcast can be implemented as follows. Every participant (TMs) relays the message it receives for the first time to all the others before delivering it to the local DBMS. It is obvious that this implementation satisfies the Uniform Agreement property.

When a TM detects a CTM crash, a time-out is set. The value of this time-out is the constant delay Δ of the reliable broadcast by which the delivery of a decision message must occur. If the delivery of a commit decision does not occur by the specified time, a participant can safely deduce that no other participant, correct or not, will deliver a commit decision (i.e., it can safely decide abort). This is due to the Δ-timeliness property of the reliable broadcast.

An atomic commit protocol is said to be non-blocking if it satisfies the following property that:

Every correct participant involved in the atomic commit protocol eventually decides.

V. CONCLUSION

In this paper, we discussed some basic atomic commit protocols involved in ensuring atomic commitment in distributed transaction management system. Having explored the studied protocols (2PC and 1PC), we conclude that both guarantee the atomicity and durability of distributed transactions even when failures occur. Blocking ACPs penalize system resources, which usually becomes heightened in a distributed database system. The one-phase protocols (1PC) can be made non-blocking, which would permit each site to continue its operation without waiting for recovery of the failed site. However, the performance of the NB-SPAC protocol depends on the performance of the reliable broadcast primitive; and concerning the I/O cost, the NB-SPAC protocol generates a single blocking I/O at the coordinator, to log the coordinator's decision along with the operations executed by the transaction that is being committed. Thus, there is no extra I/O compared with 2PC. But the size of this I/O increases with the transaction's update activity.

REFERENCES

[1] A. Wilson, "Distributed Transaction and Two-phase Commit," SAP white paper, USA, 2003.

[2] B. Lampson, "Atomic Transaction Distributed Systems: Architecture and Implementation- An Advanced Course, LNCS vol. 105, pp. 246-265, 1981.

[3] D. Duchamp, "A Non-blocking Commitment protocol" Lecture Notes in Computer Science, New York, 1989.

[4] G. Congiu, M. Grawinkel, S. Narasimhamurthy, A. Brinkmann, "One Phase Commit: A Low Overhead Atomic Commitment Protocol for Scalable Metadata Service," in: Proceedings of IEEE International Conference on Cluster Computing Workshops, 2012, pp. 16-24.

[5] H. Dubey, A. Srivastava, R. Misra, "Enhancer- A Time Commit Protocol," International Journal of Advanced Research in Computer Engineering & Technology, 1 (10), 2012.

[6] J. Gray, Notes on data base operating systems, in: R. Bayer, R.M. Graham, G. Seegmuller (Eds.), "Operating Systems: An Advanced Course," Lecture Notes in Computer Science, vol. 60, Springer, Berlin, 1978, pp. 393-481.

[7] M. Abdallah, R. Guerraoui, P. Pucheral, "One-Phase Commit: Does It Make Sense?," in: Proceedings of International Conference on Parallel and Distributed Systems, 1998.

[8] M. Ozsu, P. Valduriez, "Principles of Distributed Database Systems," (3ed), Springer, New York, 2011, pp. 427-455.

[9] N. Nouali, H. Drias, A. Doucet, "A Mobility-Aware Two-Phase Commit Protocol," The International Arab Journal of Information Technology, 3(1), 2006, pp.1-8.

[10] P. Pucheral, M. Abdallah, "A Non-Blocking Single-Phase Commit Protocol for Rigorous Participants."

[11] T. Lemlouma, N. Badache, "Non Atomic Commitment Problem: A comparative study between the 2PC and a new protocol based on the consensus paradigm,"

[12] T. Taibi, A. Abid, W. Jiann, Y. Chiam, C. Ng, "Design and Implementation of a Two -Phase Commit Protocol Simulator," The International Arab Journal of Information Technology, 3 (1), 2006, pp.20-27.

[13] Y.J. Al-Houmaily, P.K. Chrysanthis, "An atomic commit protocol for gigabit-networked distributed databases systems," Journal of systems architecture, vol. 46, 2000, pp. 809-833.

[14] Y.J. Al-Houmaily, P.K. Chrysanthis, "1-2PC: The One-Two Phase Atomic Commit Protocol," in: Proceedings of the ACM Annual Symposium on Applied Computing, 2004

[15] Y.J. Al-Houmaily, R. Conticello, J. Pike, P.K. Chrysanthis, "Performance of Five Atomic Protocols in Gigabit-Networked Database Systems," Journal of System Architecture, 48, 2002.

Authors' Brief

OLOWOOKERE, Toluwase Ayobami received a B.Tech.(Hons.) degree in Computer Engineering from Ladoke Akintola University of Technology, Ogbomoso, Nigeria, in 2010. He is currently concluding his M.Sc. degree in Computer Science at University of Port Harcourt, Nigeria. His research interest lies within the areas ofComputer Modeling and Simulation, Text and Data Mining, Virtualization, Distributed Computing, and Cloud Computing. He is a member of Institute of Electrical and Electronics Engineers, IEEE-Computer Society and a graduate member of Nigeria Society of Engineers. He can be reached via

ASAGBA,Prince Oghenekaro had his B.Sc. degree in Computer Science at the University of Nigeria, Nsukka, in 1991, M.Sc. degree in Computer Science at the University of Benin in April, 1998, and a Ph.D degree in Computer Science at the University of Port Harcourt in March, 2009. He is a Senior Lecturer and a visiting lecturer to several Universities in Nigeria since 2010. His research interest includes: Network Security, Information Security, Network Analysis, Modeling, Database Management Systems, Object-oriented Design, and Programming. He has published several articles in Learned Journals both in Nigeria and Internationally. He has authored and coauthored several books in Computer Science. He is a member of Nigeria Computer Society (NCS) and Computer Professional Registration Council of Nigeria (CPN).

OBASI, Chinedu Kingsleyholds a B.Sc (Hons) in Computer Science from Nnamdi Azikiwe University, Awka, Anambra State, Nigeria, in 2008. He is currently concluding his M.Sc Degree programme in Computer Science at University of Port Harcourt, Nigeria. His research interest area includes Machine Learning, Distributed systems, and Cloud Computing. He is a certified professional in international certifications like CCNA, MCTS and STS. He is a member of IEEE and IEEE-Computer Society. He can be reached via .