A Survey Report on One Phase, Two Phase and Three Phase Commit Protocols

Nitesh Kumar

School of Computer Engineering, KIIT University, Bhubaneswar, India

Email: niteshkumarkiit@gmail.com

***Abstract*— Transaction Management plays the key role in any type of database system. In order to make the Application Query and many protocols have been designed. We have developed very efficient an enhanced the Three Phase Commit Protocol (3PC), after studying the drawback of 1PC, and 2PC. In One Phase Commit Protocol, we observed that when a transaction runs across two sites one site may commit and another one may fail due to an inconsistent state of the transaction, and in 2PC, we also observed that if two side transactions sender side communicates to the receiver side in the form of performing initial values prepare for commit or abort message. We investigated that data is not sure from both side. If 3PC will use in such type of cases then it will avoid blocking problems of 1PC, and 2PC, because after abort/failed the data of both protocol. Data are blocked and it reduced the blocking problem solved through 3PC techniques. It has one active data for backup, if failed/abort the data to the both sides. It has stored and adds multiple sites for decision pays for committing not for abort. It is called pre-commit decision process and records of the data are stored in multiple sites (i.e., K sites).**

***Index Terms*—Commit Protocol, Transaction Management System, Distributed Database System, Remote Procedure Call Application, Directory Structure**

I. INTRODUCTION

A Transaction is a sequence of operations that takes the database from a consistent state to another consistent state. It represents a complete and correct computation. Two types of transactions are allowed in our environment: query transactions and update transactions. Query transactions consist only of read operations that access data objects and return their values to the user. Thus, query transactions do not modify the database state. Two transactions conflict if the read-set of one transaction intersects with the write-set of the other transaction. During the voting process, Update transactions consist of both read and write operations. Distributed database system is a technique that is used to solve a single problem in a heterogeneous computer network system. A major issue in building a distributed database system is the transactions atomicity. When a transaction runs across into two sites. It may happen that one site may commit and other one may fail due to an inconsistent state of transaction. Two-phase commit protocol is widely used to solve these problems. The choice of commit protocol is an important design decision for distributed database system. A Commit protocol in a distributed database transaction should uniformly commit to ensure that all the receiving sites agree to the final outcome and the result may be either a commit or an abort situation.

II. DISTRIBUTED DATABASE SYSTEMS (DDBS)

Distributed database is a system, which is used for the collection of multiple database are used for organizational and technological reasons. This distributed database system is interrelated to the communication network with replication, transparency, and fragmentation issues. Multiple database is communicating to each other with network, which is located in different locations. That's been local or global database (such as Oracle, mongoDB, Ingress, DB2, Informix etc.)[29]. This type of database is performed with organizational and technological reasons for homogeneous database system and heterogeneous database system and others . In homogeneous distributed database system are used for data collect, retrieve and storing purpose, and it performed in distributed database system. It will use the same type of database at a time, which is situated in multiple location for the purpose of multiple process as like storing, retrieving, saving, execution, and transformation [1]. But in a heterogeneous distributed database system is used for multiple database are used at a same time [30]. As an example, if A1 is a database, which is situated in New Delhi and its communicate to global database, which is situated in New York then communicate between among themselves multiple database (New Delhi are used DB2 and New York are used Oracle at a time) for distribution of the database for the purpose of transparency, replication, and fragmentation issued [2]. It is less expensive compared to a homogeneous distributed database but it has some disadvantages that's are it has only one copy of the backup, if it will crash/failed any issued then it will not recover. But in a homogeneous database system it has multiple copy are located in different sites , if one site failed then other sites are available for retrieval of the data are called reliability processing system [31]. But this system is so expensive compared to heterogeneous database systems and handles more complexity. This type of system is used in multiple applications such as ADhoc system, which is used for military system, and sensor network [1].

In distributed database system are based on Transaction Processing, which is online data are accessible, retrieved and stored in a database system through the Transaction Management system [24, 25, 26]. This type of system is handled deadlocking problem, concurrency control system, phase locking problems. The transaction management system is a system which is performed sequentially by sequence. In this system, there are two types of operation are performed (i) read, (ii) and write. Transaction management is based on ACID properties. Such as: (i) Atomicity (ii) consistency (iii) Isolation (iv) and durability. Data items that are performed in transaction Reads is called Read set and data items that are performed in transaction Writes is called Write set. If mutually exclusive union process between Read set and Write set called Base set, which is given below [1].

As example,

$$\text{Base Set} = \text{Read Set } \mathbf{U} \text{ Write Set}$$

Consider the Reservation Transaction.

RS [Reservation] = {Flight. STSOLD, Flight. cap}.

WS [Reservation] = {Flight. STSOLD, FC. FNO}.

BS [Reservation] = {Flight. cap, FC.FNO}.

The remainder of this paper is organized as the 2nd section presents distributed database system. 3rd section contains fundamentals of transaction management. 4th section contains homogeneous distributed database system. 5th section contains related works on 1PC, 2PC, and 3PC. 6th section contains literature overview of 1PC, 2PC, and 3PC. 7th section contains implementation of 1PC, 2PC, and 3PC. 8th section contains directory structure of 1PC, 2PC, and 3PC. 9th section contains an overview of RPC. 10th section contains test cases of 1PC, 2PC, and 3PC. 11th section contains compilation coding and execution of 1PC, 2PC, and 3PC. The 12th section concludes the paper.

A. Distributed Database Management System –Characteristics [33.]

- Replicated database
- Transparent database
- Fragmented database

B. Merits of Distributed Database System

- Reduced communication stack
- Economics reasons
- Incremental growth
- Improved Performance consideration
- Organizational and Technological purpose
- Improved Reliability and Availability

C. Demerits of Distributed Database System

- Complexity
- Lack of Experience
- Additional software is required

D. Types of Distributed Database Management System

1. Homogeneous Distributed Database Management System
   - All sites use same Database Management System product (e g., Oracle)
2. Heterogeneous Distributed Database Management System
   - All sites use different Database Management System product (e g., Oracle and DBII)

E. Issues in Distributed Database Design

Three key issues we have to consider:

- Data Allocation: where are data placed? Data should be stored at the site with "optimal" distribution.
- Fragmentation: relation may be divided into a number of sub-relations (called fragments), which are stored in different sites.
- Replication: copy of a fragment may be maintained at several sites.

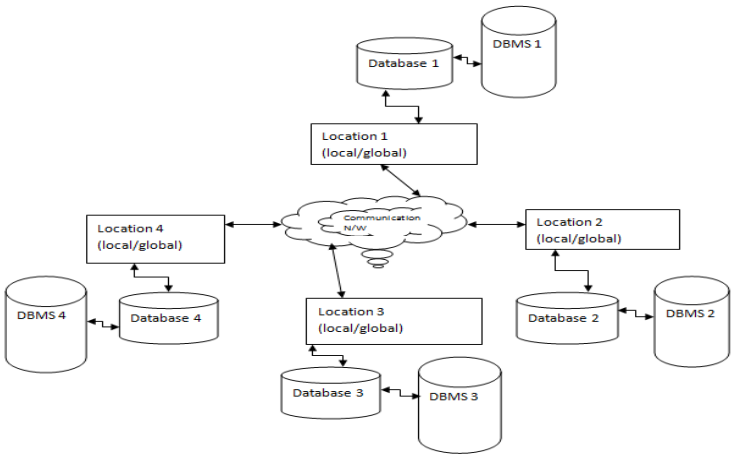

Figure 1: Distributed database system Architecture

III. FUNDAMENTALS OF TRANSACTION MANAGEMENT

Transaction management system, it deals with the problem of keeping the database in a consistent state even when concurrent accesses and failures occur [2].

A. What is a Transaction?

A Transaction, performed sequentially and it consist a series of operations for performance on a database system. The transaction management system in a consistent state prior to the initiation of a transaction to a consistent state after the transaction is completed to the database. In the transaction, when programmer is written a program in high-level for data manipulation language or programming language or query language, where it is delimited by statements or function calls of the begin and end form of the transaction [35]. The transaction, it consists of all operation such as begin transaction and end transaction is compiled and executed between such cases. A transaction is a set of/unit of program for compilation and execution that various type of data items are accessed and possibly updated. It should be done irrespective of the fact that transactions were successfully executed simultaneously or there were failures during the execution [3], [4]. The sequence of operations of a transaction it takes the database from a consistent state to another consistent state. It represents correct and complete computations. Two type of transactions are allowed in our environment: Query Transactions and Update Transactions. Query transactions, it consist only of read operation that access data objects and return their value to the user. Thus, query transactions do not modify the database state. It conflict two transactions read-set of one transaction intersects with the write-set of the other transaction. During the voting process, update transactions consists of both read and write operations. It is also a unit of consistency and reliability operations. When a transaction starts executing, it may terminate. It depends on the success or failure of the transaction [32]. It has two possibilities. (i) Abort_Transaction, for a failure of the transaction during execution. (ii) Commit_Transaction, for a completed/successfully of the transaction during execution. Figure 2, shows an example of a transaction that aborts during process 2 (A2). On the other

hand, Figure 3, shows an example of a transaction that commits, since all of its processes are successfully completed.

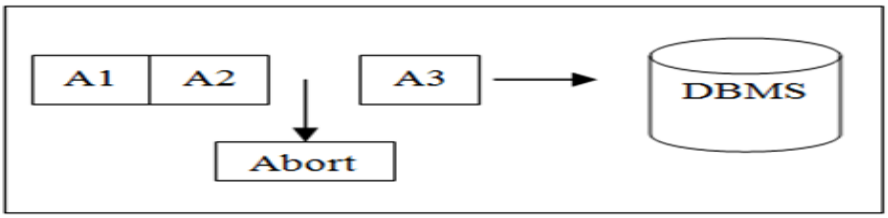

Figure 2: Abort Transaction

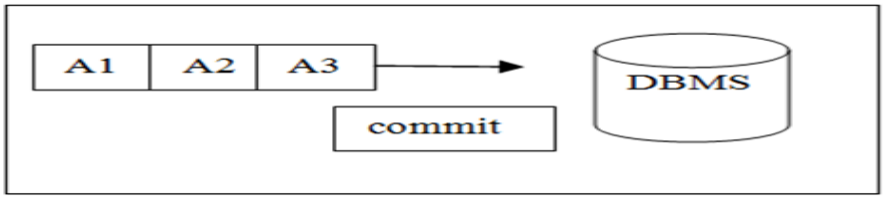

Figure 3: Commit Transaction

B. Types of Transaction

1. Flats Transaction: In flat transaction have a single start point (Begin-Transaction) and a Single termination point (End-Transaction).
2. Nested Transaction: An alternative transaction model is to permit a transaction to include other transaction with their own begin and commit points, such transaction are called nested transaction.

Example of Transaction,

Begin-Transaction Reservation
Begin
Begin-Transaction Bus

End. {Bus}.
Begin-Transaction Restaurant

End. {Restaurant}.
Begin-Transaction Car

End. {Car}.

End.

3. Workflows: A workflow is a collection of tasks organized to accomplish some business process.

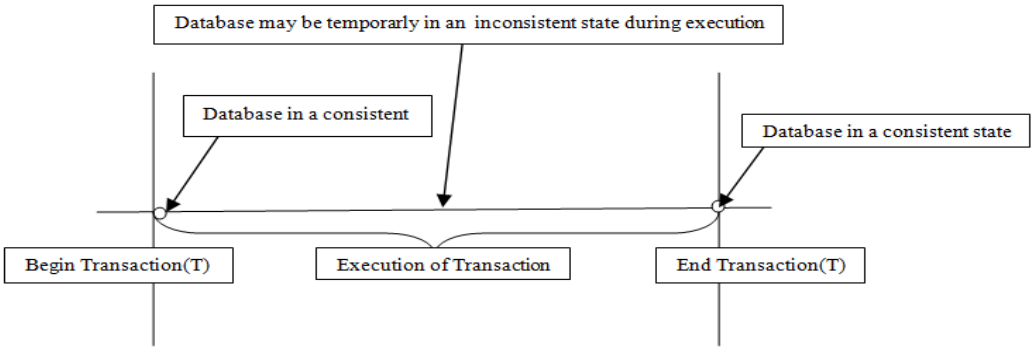

Figure 4: Transaction Model

C. Properties of Transactions

A Transaction has four properties that lead to the consistency and reliability of a distributed database. These are Atomicity, Consistency, Isolation, and Durability (ACID) [34].

1. Atomicity: This refers to the fact that a transaction is treated as a unit of operation. Consequently, it dictates that either all the actions related to a transaction are completed or none of them is carried out. For example, in the case of a crash, the system should complete the remainder of the transaction, or it will undo all the actions pertaining to this transaction. The recovery of the transaction is split into two types corresponding to the two types of failures: the transaction recovery, which is due to the system terminating one of the transactions because of deadlock handling; and the crash recovery, which is done after a system crash or a hardware failure.

2. Consistency: Referring to its correctness, this property deals with maintaining consistent data in a database system. Consistency falls under the subject of concurrency control. For example, ―dirty data‖ is data that has been modified by a transaction that has not yet committed. Thus, the job of concurrency control is to be able to disallow transactions from reading or updating "dirty data".

3. Isolation: Each transaction to see a consistent database at all times. An executing transaction cannot reveal its results to other concurrent transaction before its commitment.

Example,

T1: read (x)
x<---x+1
write (x)
commit
T2: read (y)
y<---y+1

write (y)

commit

It has two concurrent transactions (T1 & T2) both of which access data item x and y. One possible sequence of execution of the actions of these transactions, which is given below in Figure 5.

Transactions	Query Processing
T1 :	read (x)
T1 :	x<-- x+1
T1 :	write (x)
T1 :	commit
T2 :	read (y)
T2 :	y<--y+1
T2 :	write (y)
T2 :	commit

Figure 5: Isolation Processes

4. Durability: This property ensures that once a transaction commits, its results are permanent and cannot be erased from the database. This means that whatever happens after the COMMIT of a transaction, whether it is a system crash or aborts of other transactions, the results already committed are not modified or undone.

D. Type of Failure in Transaction Management

1. Transaction failures: When a transaction fails, it aborts. Thereby, the database must be restored to the state it was in before the transaction started. Transactions may fail for several reasons. Some failures may be due to deadlock situations or concurrency control algorithms.

2. System failures: Site failures are usually due to software or hardware failures. These failures result in the loss of the main memory contents.

3. Media failures: Such failures refer to the failure of secondary storage devices. The failure itself may be due to head crashes, or controller failure. In these cases, the media failures result in the inaccessibility of part or the entire database stored on such secondary storage.

4. Communication failures: Communication failures, as the name implies, are failures in the communication system between two or more sites.

IV. HOMOGENEOUS DISTRIBUTED DATABASE SYSTEM

In homogeneous distributed database system, the sites involved in distributed DBMS use the same DBMS software at every site but the sites in heterogeneous system can use different DBMS software at every site. While it might be easier to implement homogeneous systems, heterogeneous systems are preferable because organizations may have different DBMS installed at different sites and may want to access them transparently [5], [6]. Distributed DBMS can choose to have multiple copies of relations at different sites or choose to have only one copy of a relation [6]. The benefits of data replication are increased reliability – if one site fails, then other sites can perform queries for the relation. The performance will increase, as transaction can perform queries from a local site and not worry about network problems. The problem with

data replication is decreased performance when there are a large number of updates, as distributed DBMS have to ensure that each transaction is consistent with every replicated data [27]. This adds additional communication costs to ensure that all copies of the data are updated at the same time.

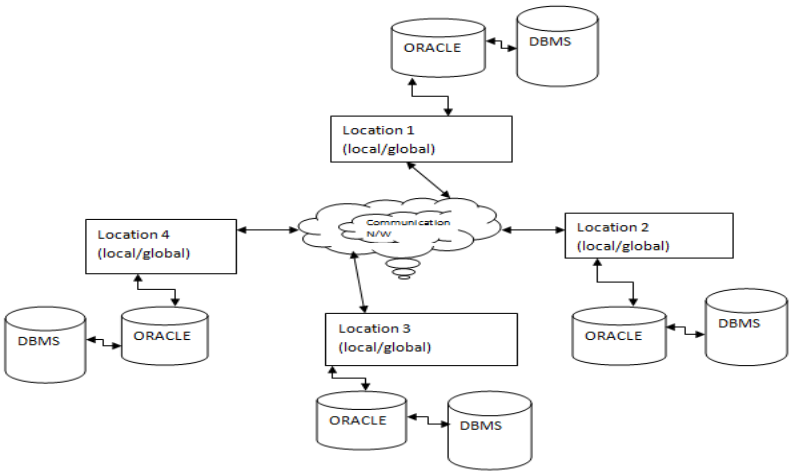

Figure 6: Homogeneous distributed database environment

A homogeneous distributed database environment is depicted in Figure 6. This environment is typically defined by the following characteristics (related to the non-autonomous category described previously):

- Data are distributed across all the nodes.
- The same DBMS is used at each location.
- All data are managed by the distributed DBMS (so there are no exclusively local data).

V. RELATED WORKS ON 1PC, 2PC, AND 3PC

When programmer/user developed the application, then it will used transparent mechanism for correctness of the distributed database systems and protocols such as (1PC, 2PC, and 3PC). Vote_Commit protocols are used for transaction message stored in the database systems with the help of computer/communication network. When require a transaction sender, it can be either a database such as Oracle, a transaction monitor such as Tuxedo or an application server such as an Oracle Web Logic Server [7], [8], [9].

Oracle-Controlled Distributed Transactions: The Oracle database can be used to send a distributed transaction across two or more Oracle databases receiver in a single transaction. For the purposes of this section only transactions that utilize a Database Link are discussed, examples of Oracle based products that use Database Links as part of a distributed transaction are Advanced Replication and Oracle AQ propagation. There are other distributed transaction mechanisms used within the Oracle database such as XA and those used by the procedural gateways, these differ from the standard distributed transaction processing described [10].

XA Based Distributed Transactions: The X/Open Distributed Transaction Processing Model, the XA standard is an X/Open specification for distributed transaction processing (DTP) across heterogeneous data sources (e.g. Oracle Database and DB2) that were published in 1991.It describes the interface between the transaction sender and the data sources that received in the distributed transaction. Within XA the transaction sender is termed the Transaction Manager and the receiver data sources are termed the Resource Managers. Transaction Managers and Resource managers that follow this specification are said to be XA compliant. Some products such as the Oracle database and Oracle Web Logic Server can act as either Transaction Managers or Resource Managers or both within the same XA transaction.

Examples of an XA Transaction Manager are: Tuxedo, Oracle Web Logic Server, the Oracle database and IBM Web Sphere Application Server [10]. Examples of an XA Resource Manager are: Oracle Database, IBM DB2, MS-SQL, IBM MQ-Series and JMS. A description of the legacy 2PC-based protocols using UML Sequence diagrams [11]. Descriptions of committing and abort protocols are supported and can be implemented as various configurations built with reusable components. This paper improves the adaptation policy to consider the commit rate variation depending on the number of transaction participants [10]. The support of committing protocols for transactions with predictable issues has been introduced. The completion time of transactions with predictable issues, such as transaction aborting unilaterally, is improved compared to traditional commit protocols.

Paper [12], [13] was mainly centered on the Simulation of the 2PC for ensuring atomicity in distributed transactions. RMI was used in our message-passing and communication model, instead of using Socket to handle communication. Some other considerations related to this protocol are also taken into account and improved upon in order to construct an optimized Simulation. In a generic system is simulated in a distributed environment to represent the real world scenario more widely. the fact that distributed transaction processing systems are widely used in many different organizations of varying size, as well as the nature of task distribution in a networking environment.

Transaction Management is an old concept in distributed database management systems (DDBMS) research. However, Oracle was the first commercial DBMS to implement a method of transaction management the two phase commit. Though it was very difficult to obtain information on Oracle's implementation of this method. Many organizations do not implement distributed database because of its complexity. However, with global organizations and multi-tier network architecture, distributed implementation becomes a necessity. Organization in the implementation of distributed databases when installing Oracle DBMS, or encourage organizations to migrate from centralized to distributed DBMS. Universities could also contribute to this process by having graduates with the knowledge of Oracle DBMS capabilities [14]. In terms of transactions management, the systems most related to Camelot are Argus and Quicksilver. Camelot has taken certain techniques, especially those related to communication support, from another IBM research system, R* [15], [16]. Camelot and Argus have nearly the same transaction model; these two projects represent the two implementations of Moss-Model nested transactions. the only major difference is that in Argus a transaction can make changes at only one site. Diffusion must be done within a

nested transaction. Argus has paid close attention to the performance of their implementation of two-phase commit [17].

Remote Method Invocation (RMI) is a mechanism that enables an object on one Java virtual machine to invoke methods on an object in another Java virtual machine (i.e., Method invocation in distributed environment). Any object that can be invoked in this way must implement the java.rmi.Remote interface and extends java.rmi.server.UnicastRemoteObject. Remote object can be bound or registered in RMI registry by using java.rmi.Naming.rebind (String name, Remote object) or java.rmi.Naming.bind (String name, Remote object), where the name is in URL form: "rmi://host/name" and object is the object to be bound. RMI registry must be available on the host. For example, (referring to the class diagrams):

Naming.rebind ("rmi://" + rmiHost + "/" + name + "/SenderLog", senderLog);

To obtain references for a remote object *java.rmi.Naming.lookup (String name)* is used.

For example, (referring to the class diagrams):

LogQueryListenerlogQueryListener= (LogQueryListener) Naming.Lookup ("rmi://" + rmiHost + "/" + senderAddress + "/SenderLog");

Below is a list of remote objects in the 2PC simulator:

a) SenderLog, ReceiverLog.

b) Transaction Manager, DataManagerImpl.

c) LockingManager, TimerImpl.SimulationEventHandler, NewTransactionHandler.

d) SimulatorServerImpl.

Simplicity and availability of RMI package in Java SDK 1.4 Standard Edition are the main drivers behind the use of RMI in the 2PC simulator.

Extensible Markup Language (XML) is a simple and flexible language to keep the data as a text string in a file. In our project, TransactionDataLog, SenderLog, ReceiverLog and DataManagerImpl store these data as XML documents. Classes for processing XML documents, which have available in Java SDK 1.4 Standard Edition provided us with an easy means to create and edit XML documents and to transform XML documents between file stream and the Document Object Model (DOM), which is a structural form of XML in system memory [4]. In addition, data in XML documents can be displayed in any XML browser by applying an Extensible Stylesheet Language - Transformations (XSLT) style sheet to the XML document. Packages used in this project for processing XML documents are javax.xml.parsers and javax.xml.transform.

VI. LITERATURE OVERVIEW OF 1PC, 2PC, AND 3PC

In Distributed database system, there are multiple types of phases are used in this, which is 1PC, 2PC, and 3PC.

A. One Phase Commit Protocol (1PC)

When one phase commits protocol is used in transaction management system for homogeneous distributed database system. When a transaction runs across two sides, first called sender side, which is also called

Transaction sender and 2nd are called receiver side, which is also called Transaction receiver. In this system, first time when used the transaction management system for performed the transaction read and write operations. We find that, it has not surety both sides because, when transaction will perform from sender side to prepare to commit of initial value may commit and other may fail/abort for an inconsistent state of the transaction. It means not surety from sender side and receiving side that are committed [18]. According to figure 7, sender side, send the commit data to the receiver side performed for execution of the transaction and get the response for committing acknowledgement, but this transaction after execution from receiving side [22]. It is lost in the middle way, when it will transform receiver to sender side. Then again it will send the commit side data. Sender side data are always committing, but receiver side data are not shared. It will commit or abort. That's the problem is avoiding through 2PC [23]. It will evaluate this type of problems, because it will not evaluate then one transaction will send multiple times, it will lose multiple time. Time will waste and memory is also lost . This process will run again and again [18]. This figure is given below:

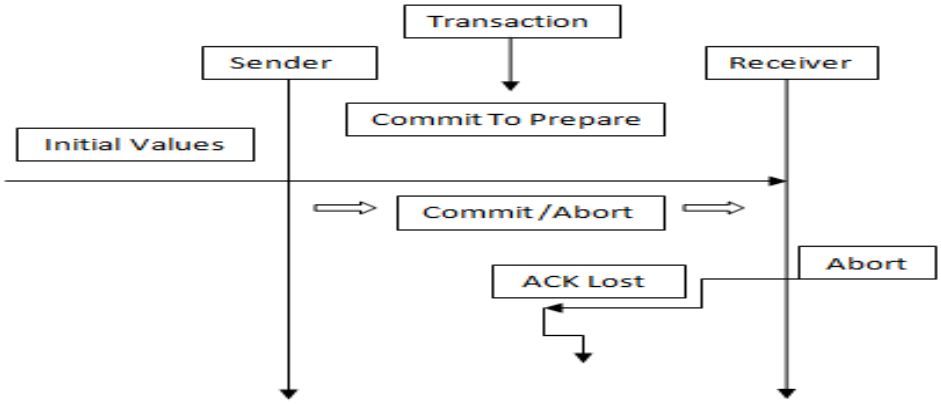

Figure 7: One Phase Commit Protocol Work Flow

Advantages of 1PC

- Simple protocol fewer overheads.
- Low latency due to fewer disk writes.
- Useful in case of low bandwidth networks as lesser messages are exchanged.
- In most cases all the updates will be logged before commit so durability is assured.

Disadvantages of 1PC

- It can only handle immediate consistency constraints as there is no voting phase.
- It cannot handle the deferred consistency constraints.

B. Two Phase Commit Protocol (2PC)

In 2PC process, when transaction will perform two sides, i.e., one side called sender side and 2nd side called receiver side. They are interconnected between network communications. When it will perform the transaction between them that time, they have four types of possibility [28]. They are:

(i) 1st case sends the data from the sender side to the receiver side for execution of the transaction, but they have not surety that is data has committed from sender side. In this case sender side data are not commit it will abort; if it will abort that means both sides are failing/abort the data.

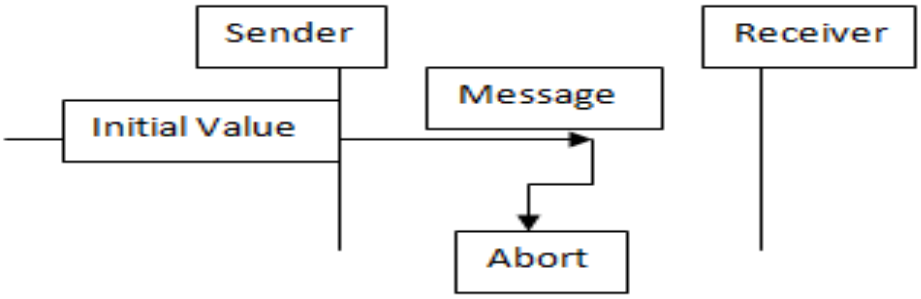

Figure 8: Abort Message from sender side

ii) In the 2nd case, send the data form sender side it will commit but receiver side it will not surely that send the acknowledgement from receiving side that is committed, the data are lost in the middle. It means the data are aborted.

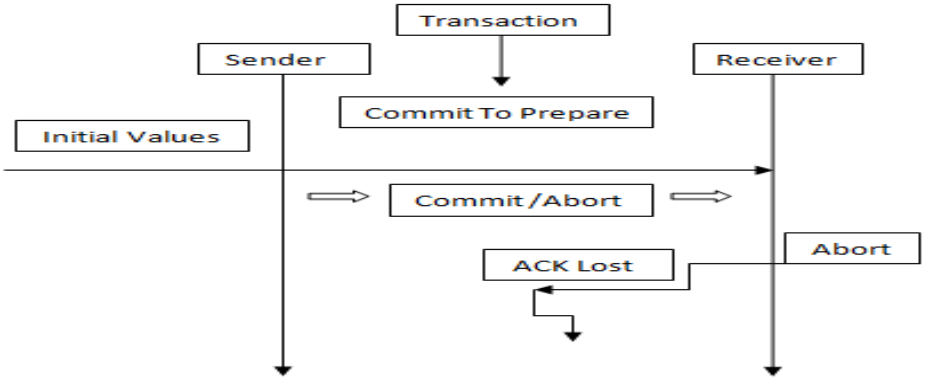

Figure 9: Abort Message from receiver side

(iii) In the 3rd case, data are committing both side sender and receiver side, but not always,

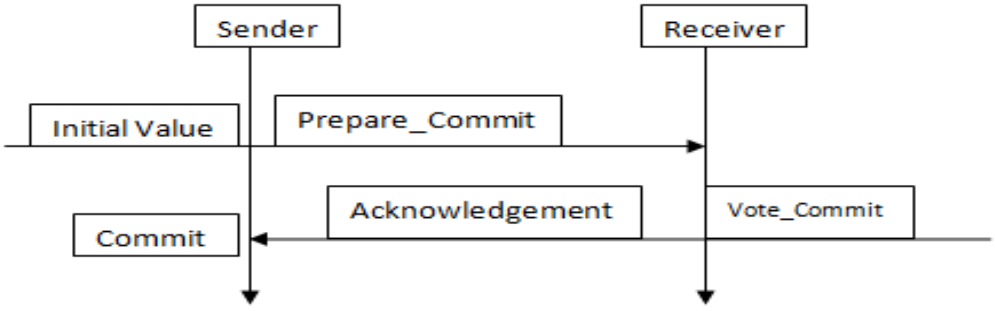

Figure 10: Commit Message from both sides

(iv) in 4th case, data are committing always from sender side but receiver side data are not sure that is committed or abort at a time. it means commit to commit and commit to abort.

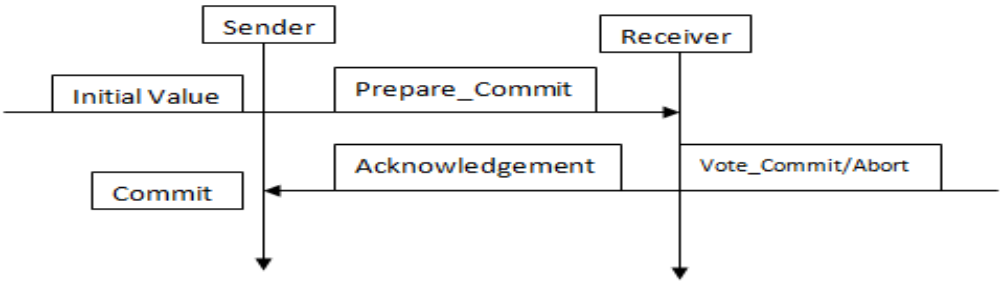

Figure 11: Vote_Commit/Abort Message from receiver sides

According to the figure 12, its complete two phases commit protocol work flow, this is given below:

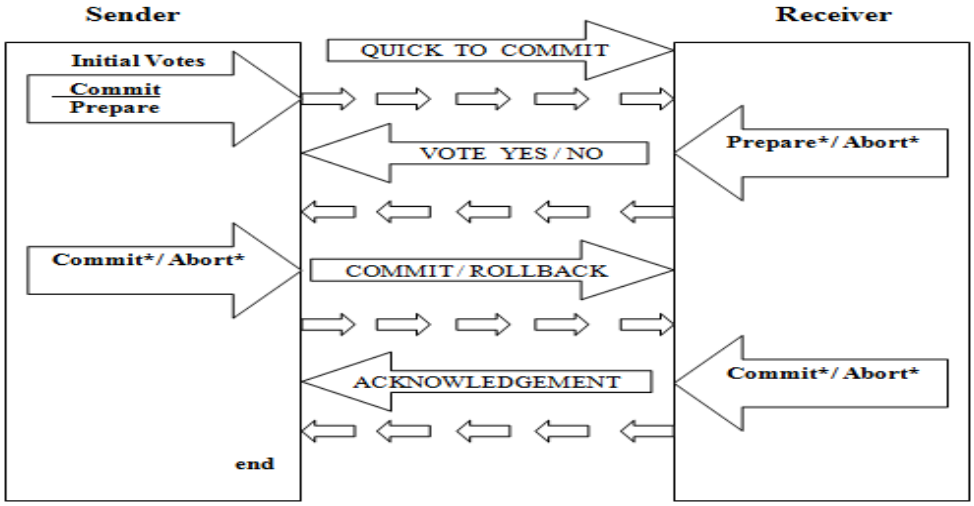

Figure 12: Complete Two Phase Commit Protocol Work Flow

Advantages of 2PC

a. It ensures atomicity even in the presence of deferred constraints.
b. It ensures independent recovery of all sites.
c. Since it takes place in two phases, it can handle network failures, disconnections and in their presence assure atomicity.

Disadvantages of 2PC

a. Involves a great deal of message complexity.
b. Greater communication overheads as compared to simple optimistic protocols.
c. Blocking of site nodes in case of failure of the sender.
d. Multiple forced writes of logs, which increase latency.

e. Its performance is again a trade-off, especially for short lived transactions, like Internet applications.

Types of 2PC

1. The Centralized Two-Phase Commit Protocol

In the Centralized 2PC shown in Figure 13 and 14 Communication is done through the sender's process only, and thus no communication between subordinates is allowed. The sender responsible for transmitting the PREPARE message to the subordinates, and, when the votes of all the subordinates are received and evaluated, the sender decides on the course of action: either abort or COMMIT. This method has two phases:

First Phase: In this phase, when a user wants to COMMIT a transaction, the sender issues a PREPARE message to all the subordinates, (Mohan et al., 1986). When a subordinate receives the PREPARE message, it writes a PREPARE log and, if that subordinate is willing to COMMIT, sends a YES VOTE, and enters the PREPARED state; or, it writes an abort record and, if that subordinate is not willing to COMMIT, sends a NO VOTE. A subordinate sending a NO VOTE doesn‘t need to enter a PREPARED state since it knows that the sender will issue an abort. In this case, the NO VOTE acts like a veto in the sense that only one NO VOTE is needed to abort the transaction. The following two rules apply to the sender’s decision, (Ozsu et al., 1991): If even one receiver votes to abort the transaction, the sender has to reach a global abort decision. If all the receivers vote to COMMIT, the sender has to reach a global COMMIT decision.

Second Phase: After the sender reaches a vote, it has to relay that vote to the subordinates. If the decision is COMMIT, then the sender moves into the committing state and sends a COMMIT message to all the subordinates informing them of the COMMIT. When the subordinates receive the COMMIT message they in turn, move to the committing state and send an acknowledgement (ACK) message to the sender. When the sender receives the ACK messages, it ends the transaction. If, on the other hand, the sender reaches an ABORT decision, it sends an ABORT message to all the subordinates. Here, the sender doesn‘t need to send an ABORT message to the subordinate(s) that gave a NO VOTE.

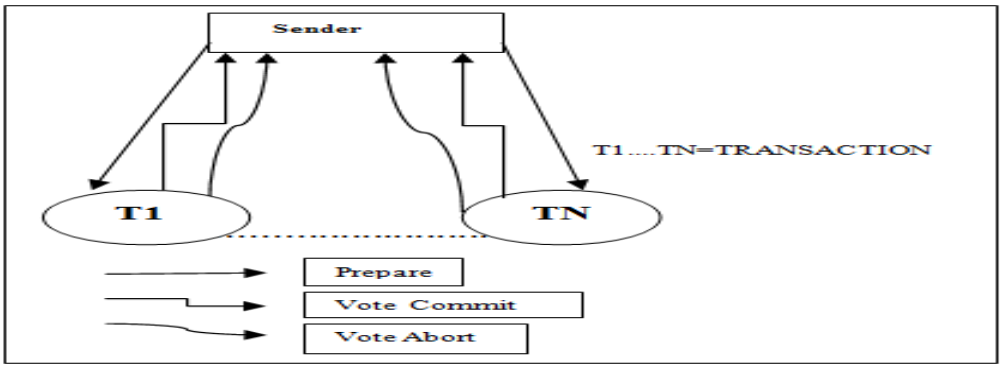

Figure 13: The Centralized Two-Phase Commit Protocol

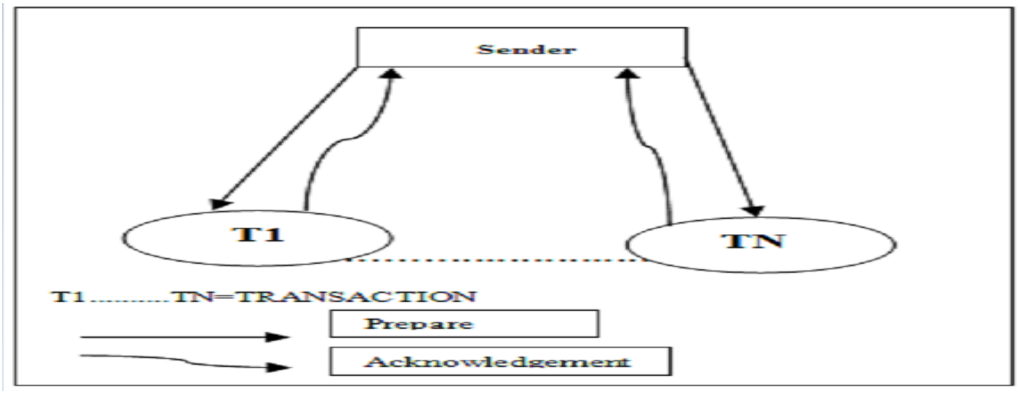

Figure 14: The Centralized Two-Phase Commit Protocol

2. The Distributed Two-Phase Commit Protocol

In the distributed 2PC, all the nodes communicate with each other. According to this protocol, as Figure 15 shows, the second phase is not needed as in other 2PC methods. Moreover, each node must have a list of all the receiver nodes in order to know that each node has sent in its vote. The distributed 2PC starts when the sender sends a PREPARE message to all the receiver nodes. When each participant gets the PREPARE message, it sends its vote to all the other receivers. As such, each node maintains a complete list of the receivers in every transaction. Each receiver has to wait and receive the vote from all other receivers. When a node receives all the votes from all the receivers, it can decide directly on COMMIT or abort. There is no need to start the second phase, since the sender does not have to consolidate all the votes in order to arrive at the final decision.

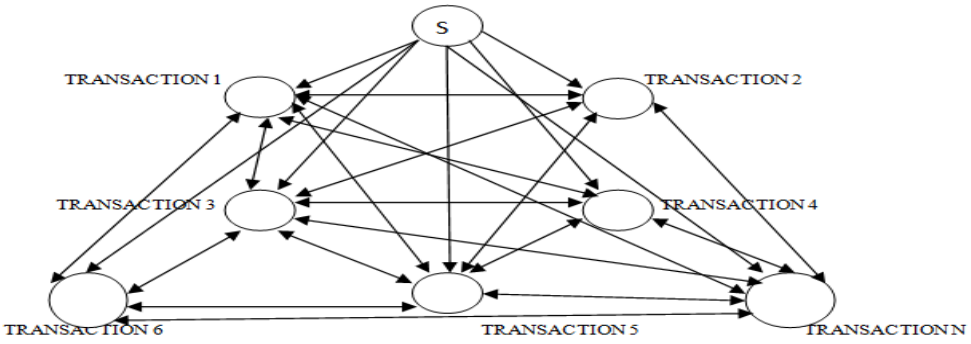

Figure 15: The Distributed Two-Phase Commit Protocol

3. The Linear Two-Phase Commit Protocol

In the linear 2PC, as depicted in Figure 16 and 17, subordinates can communicate with each other. The sites are labeled 1 to N, where the sender is numbered as site 1. Accordingly, the propagation of the PREPARE message is done serially. As such, the time required to complete the transaction is longer than centralized or distributed methods. Finally, node N is the one that issues the Global COMMIT. The two phases are discussed below:

First Phase: The sender sends a PREPARE message to receiver 2. If receiver 2 is not willing to COMMIT, then it sends a VOTE ABORT (VA) to receiver 3 and the transaction is aborted at this point. If receiver 2, on the other hand, is willing to commit, it sends a VOTE COMMIT (VC) to receiver 3 and enters a READY state. In turn, receiver 3 sends its vote till node N is reached and issues its vote.

Second Phase: Node N issues either a GLOBAL ABORT (GA) or a GLOBAL COMMIT (GC) and sends it to node N-1. Subsequently, node N-1 will enter an ABORT or COMMIT state. In turn, node N-1 will send the GA or GC to node N-2, until the final vote to commit or abort reaches the sender node.

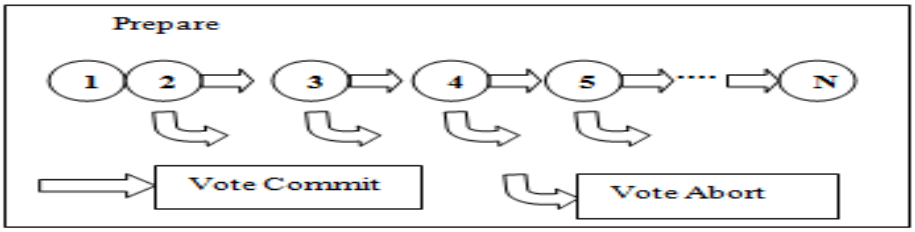

Figure 16: The Linear Two-Phase Commit Protocol

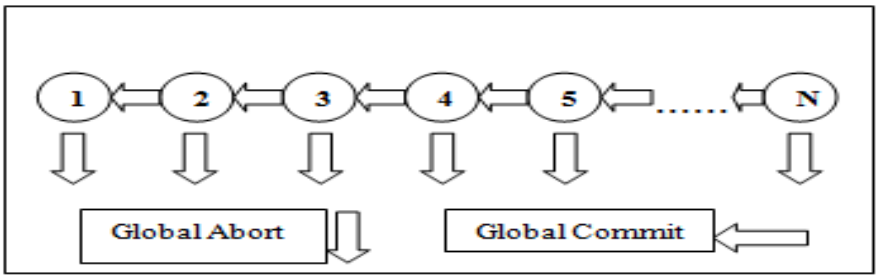

Figure 17: The Linear Two-Phase Commit Protocol

C. Three Phase Commit Protocol (3PC)

In this 3PC system, when two sides performed the transaction for execution of the acknowledgement. It creates a three phase, phase one called prepare, 2nd called pre-commit, and 3rd called acknowledgement commit/abort. This system is used for avoiding the blocking problem, which is created in 2PC at the time of execution of the transaction [19], [20]. It decides the pre-commit despite failure of the acknowledgement. It is also called non-blocking protocol, which is used to eliminate/reduce the blocking problem [20]. From this analysis, we investigate Three Phase Commit Protocol (3PC) to avoid such states and it is resilient to such failure. An extra round of message transmission further it reduces the system's performance fast as compared to 2PC protocol. This system work as like 2PC, but in this system one extra phase has been created for pre-commit decision of k-sites for the failure time.

(i) In preparing phase/initializes phase: in this sender side case, prepare phase initialized the sender side for the Begin_Commit request message to all receivers and it will wait for the state, when receiver side,

receive the request message. It is totally depends on the receiver side. If they want to commit the message means Vote_Commit, otherwise Vote_Abort to respond the sender side.

(ii) In pre-commit phase: On the sender side, within time if the sender receives the message in form of Vote-Commit. They broadcast to the all receivers in pre-commit. Given time if a receiver side, the receiver accepts the message from the sender side in the form of pre-commit data. It will send acknowledgement message to the sender side.

(iii) In commit/abort phase: In this phase sender side, the sender receives the message successfully acknowledgement state and it will inform to the receiver side that is committing the transaction message or abort the transaction message and it will inform to the receiver side for the next state transaction to prepare state. All operations are performed according to this figure 18 and figure 19 (Table 1) summarizes the message of each protocol as shown given below.

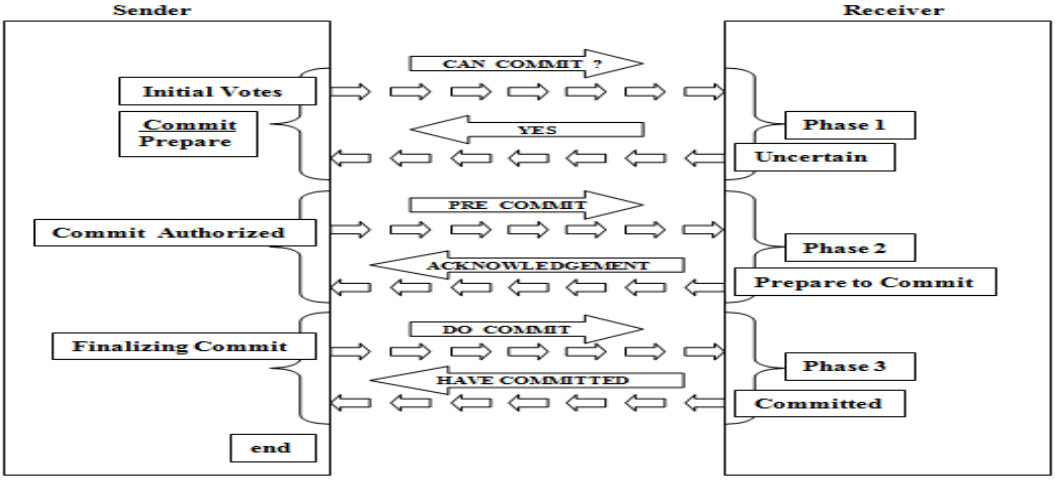

Figure 18: Three Phase Commit Protocol Work Flow

Message Functions			
Protocol	Phase	Sender	Receiver
1PC	2 Phases	Prepare_Commit	Commit
	end	Prepare_Commit	Abort
2PC	2 Phases	Prepare_Commit	Commit
		Prepare_Abort	Abort
		Prepare_Commit	Abort
	end	Prepare_Commit	Commit/Abort
3PC	3 Phases	Prepare_Commit	Vote_Commit/Vote_Abort
		Vote_Commit/Pre_Commit	Pre_Commit
	end	Acknowledgement State	Commit_Transaction/Abort_Transaction

Figure 19: Table 1. The Message Distributed Commit Protocols

Advantages of 3PC

a) Avoids blocking under some situations.
b) Multiple sites in decision phase to commit.

Disadvantages of 3PC

a) Extra overheads (driving mechanism).
b) Increased complexity and costs.
c) Complicated to implement.
d) Having more communication overhead maintains inconsistency towards network partitioning.

VII. IMPLEMENTATION OF 1PC, 2PC, AND 3PC

Initialized and begins the commit protocols and send the transaction to the all receiver sites send the transaction and their acknowledgement [1]. The time of unavailability of interfaces from the transaction manager in currently simputerDB, the simulation of the transaction manager, it starts then after that it will execute the transaction and got complete transactions then attempts for the commit protocol's execution then after that it begins the protocols and it will use multiple techniques such as: (i) data structure of 1pc, 2pc, and 3pc (ii) algorithm of 1pc, 2pc, and 3pc (iii) directory structures of 1pc, 2pc, and 3pc.

A. Algorithm of 1PC

Step I: **Voting Site/Prepare Site/Initial Site/sender site**

Arguments

- Set of Sites
- Message_id
- Initial_Votes
- Status of Message (Prepare_Commit)

Begin

if each message status= **COMMIT/ SUCCESS**

Force-Write Sending **VOTE_COMMIT**

send **VOTE_COMMIT** message to all.

else if any message =**VOTE_ABORT**

Force-Write Sending **VOTE_ABORT**

Send **VOTE_ABORT** message to all.

End.

Step II: **Receiver Site**

Arguments

- Message_id

- Status of local/global message (Abort/Commit)

Begin

Wait for **VOTE_COMMIT** message from sender site

receive **VOTE_COMMIT** message from sender site

if each message=**VOTE_COMMIT**

Force-Write **VOTE_COMMIT**

Commit local/global message

else if message=**VOTE_ABORT**

Force-Write **VOTE_ABORT**

Abort local/global message

End.

B. Algorithm of 2PC

Step I: **Voting Site/Prepare Site/Initial Site/sender site**

Arguments

- Set of Sites
- Message_id
- Initial_Votes
- Status of Message (Prepare_Commit)

Begin

Write Initial_Votes Sending **PREPARE_COMMIT**

for each site in set

Send **PREPARE_COMMIT** messages

Phase 1: Receive message from all sites

if each message = **READY_COMMIT**

Write **VOTES_YES** Sending **VOTE_COMMIT**

Send **VOTE_COMMIT** message to all

else if any message = **READY_ABORT**

Write **VOTES_NO** Sending **VOTE_ABORT**

Send **VOTE_ABORT** message to all

End.

Step II: **Receiver Site**

Arguments

- Message_id

- Status of local/global message (Abort/Commit)

Begin

Wait for **PREPARE_COMMIT** message from Sender Sites

if each message = **MESSAGE_PRE-COMMIT**

Send **READY_COMMIT** message

Change the state of the message to **MESSAGE_COMMIT_READY**

Phase 2: Receive message from all Sender sites

if each message = **MESSAGE_COMMIT_READY**

Write VOTES_YES **VOTE_COMMIT**

Send **ACKNOWLEDGEMENT (ACK)**

VOTE_COMMIT message to all Sites

Commit local/global message

else if any message = **MESSAGE _ABORT_READY**

Write VOTES_NO **VOTE_ABORT**

Send **ACKNOWLEDGEMENT (ACK)**

VOTE_ABORT message to all Sites

Abort local/global message

End.

C. Algorithm Implementation of 3PC

Step I: **Voting Site/Prepare Site/Initial Site**

Arguments

- Set of Sites
- Message_id
- Initial_Votes
- Status of Message (Prepare_Commit)

Begin

Write Initial_Votes Sending **PREPARE_COMMIT**

for each site in set

Send **PREPARE_COMMIT** messages

Phase 1: Receive message from all sites

if each message = **READY_COMMIT**

Write **VOTES_YES** Sending **VOTE_COMMIT**

Send **VOTE_COMMIT** message to all

else if any message = **READY_ABORT**

Write **VOTES_NO** Sending **VOTE_ABORT**

Send **VOTE_ABORT** message to all

End.

Step II: **Pre-Commit Site**

Arguments

- Message_id
- Status of local/global message (Abort/Commit)

Begin

Write Sending **PRE-COMMIT** message

for each site in set

Send **PRE-COMMIT** messages to all

Receiver sites and stored the **PRE_COMMIT**

message in K-Sites for the time of any

failure message

Phase 2: Receive message from all Sender sites within time

if each message = **READY_PRE-COMMIT**

Write **VOTES_YES** Sending **VOTE_COMMIT**

Send **VOTE_COMMIT** message to all

else if any message = **READY_ABORT**

Write **VOTES_NO** Abort/Failure_Message

then retrieve the message from other sites,

Which is stored in **K-Sites**

End.

Step III: **Commit Site**

Arguments

- Message_id
- Status of local/global message (Abort/Commit)

Begin

Wait for **PREPARE_COMMIT** message from Sender

Sites

if each message = **MESSAGE_PRE-COMMIT**

Send **READY_COMMIT** message

Change the state of the message to

MESSAGE_COMMIT_READY

Phase 3: Receive message from all Sender sites

if each message = **MESSAGE_COMMIT_READY**

Write VOTES_YES **VOTE_COMMIT**

Send **ACKNOWLEDGEMENT (ACK)**

VOTE_COMMIT message to all Sites

Commit local/global message

else if any message = **MESSAGE _ABORT_READY**

Write VOTES_NO **VOTE_ABORT**

Send **ACKNOWLEDGEMENT (ACK)**

VOTE_ABORT message to all Sites

Abort local/global message

End.

D. Data Structure of 1PC, and 2PC

We are the important of data structures for the Two phase commit protocol. The similar data structure have been used for the One phase commit protocol also. For details to the documentation of the code is given below step by step [19].

- **Receiver -** It is a C structure consisting of the string hostname, port, username, password of the database at the receiver site.
- **Global Transaction State Data -** It is a C structure maintaining the list of the receiver site, transaction id, number of receiver sites, and the state of the global transaction.
- **List -** It is a generic structure written to maintain the link list of any kind of structures or integers.
- **Transaction State Data -** Maintained at the receiver sites. It is a structure maintaining The transaction id assigned by the local transaction manager, transaction id given by the sender, sender's IP address, start time of the transaction and the state of the global transaction.
- **Log Record -**It is a C structure to maintain the information to be written into the logs. It consist of the timestamp, the type of the log message , the log message , the IP address of the sender site/receiver site and the transaction id.

- **Messages -** In a Distributed DBMS the highest cost being the network cost, we have tried to keep it as low as possible. The Messages passed are some constant integers for representing the Messages like READY, ACK, ABORT , FAIL. The READY message is sent by the receiver site as a response of the PREPARE-COMMIT message from the sender site. The PREPARE-COMMIT message is sent implicitly through Remote Method Invocation by invoking the Pre-commit Transaction function of the receiver site.
- **Transaction States -** We have defined constants for the different states of the transaction -START, PRE-COMMIT, COMMIT- READY, COMMIT, ABORT, END.

E. Data Structure of 3PC

We are using data structures of three phase commit protocol and also similar data structure have been used in two phase commit protocol, and one phase commit protocol. Through data structure techniques, we are used documentation of the coding step by step [19].

- **Receiver:** It has used C and construct c structure for consist to the string hostname, port number, username, and password of the distributed database at the receiver (server) sites.
- **Global Transaction Data:** It has used C and construct c structure for maintenance of list of receiver site transactions, number of receiving sites, and state of the global transactions.
- **List:** It has used C structure for maintaining the linked list and integers for any types of structures.
- **Transaction Data**: It maintained the transaction Identification and assign to the global transaction manager, transaction identification given to the sender (client) sites, client IP address, and start time of the state of the transaction.
- **Message**: Message is passed in distributed database system such as: READY, ACK, FAIL, VOTE_COMMIT, VOTE_ABORT, PREPARE-T.
- **Log Record**: It is used to maintain the C structure for information and it will write into the logs consists of the Timestamp.
- **Transaction State**: It has multi types of transaction states used in this such as: START, PRE-COMMIT, COMMIT-READY, COMMIT, ABORT, END.

VIII. DIRECTORY STRUCTURE OF 1PC, 2PC, AND 3PC

A. Directory Structure of 1PC, and 2PC

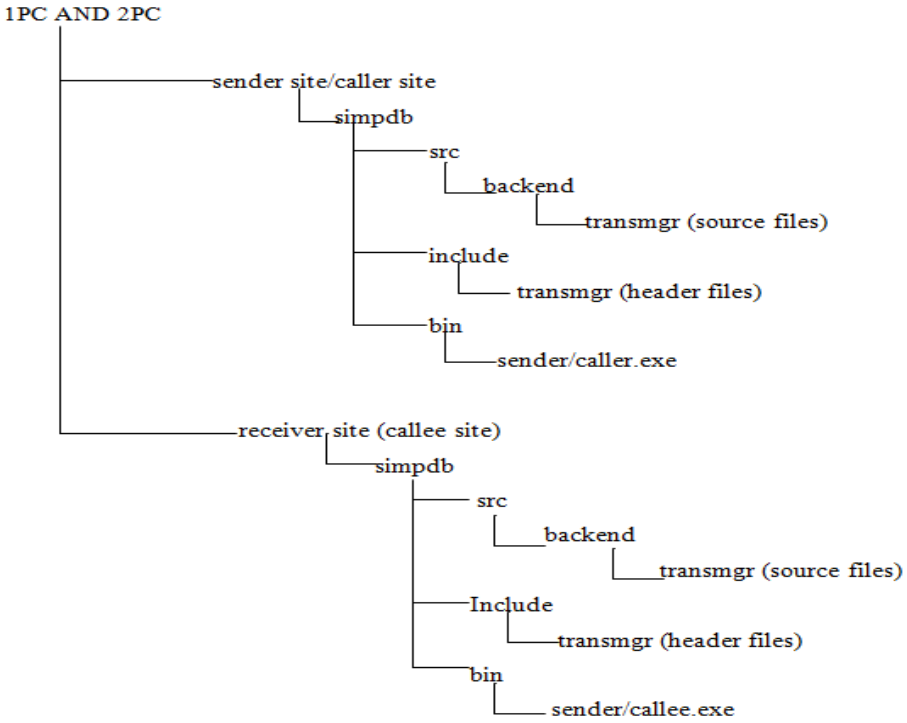

Figure 20: Directory Structure for 1PC and 2PC

B. Directory Structure of 3PC

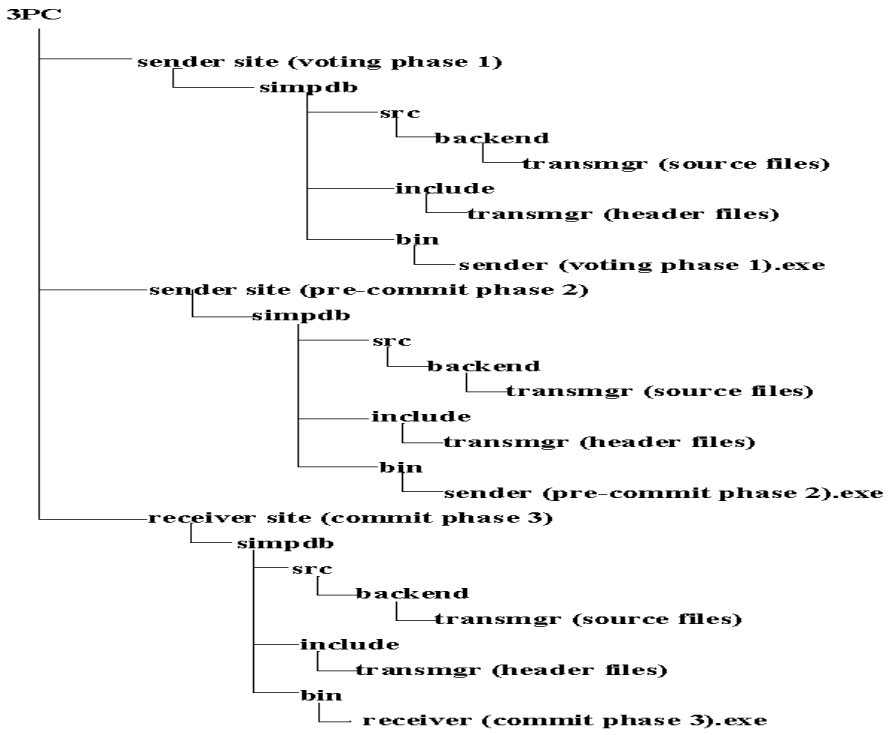

Figure 21: Directory Structure for 3PC

IX. An Overview of RPC

Remote Procedure Call (RPC) is a most powerful techniques, which is used for constructing distributed applications. The receiver sites have RPC daemons/runtime, its running on the port. The sender sites, transaction manager invoke the required methods on each of the sites in the list of receiving sites for a particular transaction. RPC compiler is used UNIX-C RPC library for compiling stubs for communication [21]. RPC is a valuable communication mechanism and it is suitable for constructing building a fairly large number of distributed application's features such as: its communicate between processes on the same machine, efficiency, generality, well-defined interfaces, local procedure call similarity, syntax call etc. It used some models and also used transparency such as: semantic, and syntactic. RPC is totally based on/concept of stubs, which is used for local procedure call abstract by cancelling from programs the communication interface to the RPC system underlying. The processes of RPC, it involves a client (sender site) global transaction manager to RPC stub and communicate to RPC daemon/runtime then receiver site (global transaction manager). It gives the response to sender site through RPC daemon/runtime and RPC stubs. When implement the RPC mechanism, it has five elements of the program: (i) Client (ii) Client stub (iii) RPC Runtime/Daemon (iv) Server (v) Server stub, which is based on given Figure 22 below [21].

- **Client:** The process of client is initiating an RPC through users and it invokes a corresponding procedure in the client stub
- **Client stub:** It follow type of receipt (a) call request from the client site, and (b) result of procedure execution.
- **RPC Runtime/daemon:** This system is used for transmission of message across the communication network between client and server machines.
- **Server:** It is used for receiving all requests from the sender sites and it executes the procedure and returns the result of procedure execution of the server sites.
- **Server stub:** It is used for, when executes the procedure and returns the result of procedure execution of the server sites. It receives and communicate to the RPC stub such as: (a) call request message from the local/global RPC Runtime/daemon, and (b) result of procedure of execution from the server sites.

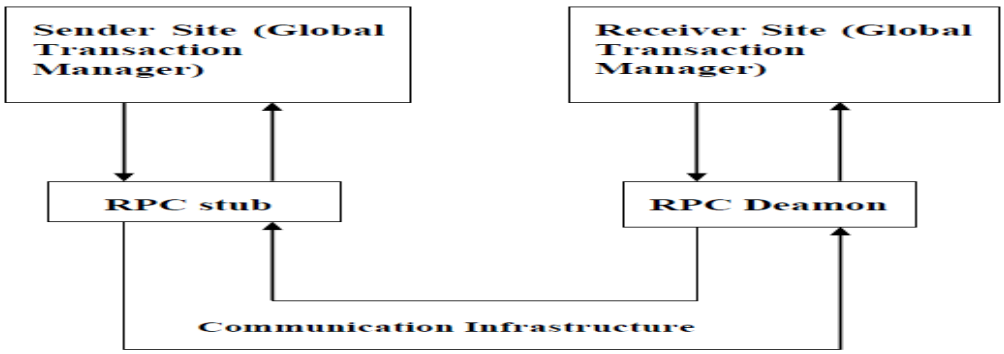

Figure 22: Communication Interface

X. TEST CASES OF 1PC, 2PC, AND 3PC

We are listing down all the testcases for the Two phase commit protocol and the one phase commit protocols. For executing and testing the following testcases you only need to run the receiver site for the receives once, and then you need to run the sender site 9 times for each of the testcases. We have written a simulation function in the receiver site for demonstration that takes care of different testcases automatically. In that function we change the state of the transaction to Pre-commit or Abort, and set the flag for deferred constraints checks to be pass or fail. Based on the requirements of the different testcases, different values are set for these variables and the protocol then shows the corresponding behaviors. The correctness can be ensured from the logs and the debug messages left for demonstration [19].

A. Test Cases of 1PC

- Transaction is successfully executed and reaches the commit state. no delays takes place so no site fails. The expected output is the COMMIT log on the sender site and the receiver site.
- Transaction is successfully executed and reaches the commit state. During commit delay takes place at receiver sites so receiving site fails. The expected output is the COMMIT log on the sender site.
- Transaction fails and reaches the abort state. no delays takes place so no site fails. The expected output is the ABORT log on the receiver site.
- Transaction fails , and reaches the abort state . During abort delays takes place at receiver sites so receiving site fails. The expected output is the ABORT log on the receiver sites.

B. Test Cases of 2PC

- Transaction is successfully executed. It reaches the pre-commit state and votes yes. No delays takes place so no site fails. The output is PREPARE and COMMIT log in the sender sites log file and the READY and COMMIT message in the logs of the receiver sites.
- Transaction is successfully executed. It reaches the pre-commit state and votes yes. During Pre-commit delay takes place at receiver sites so receiving site fails. The output is PREPARE and ABORT log in the sender sites log file and the READY and ABORT message in the logs of the receiver sites.
- Transaction is successfully executed. It reaches the pre-commit state and votes yes. During commit delays takes place so receiver site fails. The output is PREPARE and COMMIT log in the sender sites log file and the READY and COMMIT message in the logs of the receiving sites except for the sites where commit delayed.
- Transaction is successfully executed. It reaches the pre-commit state, deferred constraints failed and it votes NO. No delays takes place so no site fails. The output is PREPARE log in the sender sites log file and the READY and ABORT message in the logs of the receiver sites.
- Transaction is successfully executed and reaches the pre-commit state. Deferred constraints failed and site votes NO. During pre-commit delay takes place at receiver sites so receiver site fails. The

output is PREPARE log in the sender sites log file and the READY and ABORT message in the logs of the receiving sites.

- Transaction is successfully executed and reaches the pre-commit state. Deferred constraints failed and site votes NO. During abort delays takes place so receiving site fails. The output is PREPARE log in the sender sites log file and the READY and ABORT message in the logs of the receiving sites except for the sites where abort delayed.
- Transaction fails and goes to abort state sites votes NO. No delays takes place so no site fails. The output is PREPARE log in the sender sites log file and the READY and ABORT message in the logs of the receiving sites.
- Transaction fails and goes to abort state sites votes NO. During pre-commit delay takes place at receiver sites so receiving site fails. The output is PREPARE log in the sender sites log file and the READY and ABORT message in the logs of the receiver sites.
- Transaction fails and goes to abort state, votes NO. During abort delays takes place so receiver site fails. The output is PREPARE log in the sender sites log file and the READY and ABORT message in the logs of the receiving sites.

C. Test Cases of 3PC

Three phase commit protocol test cases following are: executing and testing of these test cases only need to run the receiver site for the participants once, and sender site needs to run 6 times for each of test cases. Because 3PC is split into three phases (phase 1 called (voting phase), phase 2 called (pre-commit phase), and phase 3 called (commit protocol)). When we have written a function in the receiver site for demonstration. In this function, we change the state of transaction to pre-commit, commit or abort and set of flags of check constraints to be failed or pass [19]. Which is follow given below cases?

- The transaction is successfully executed, for phase 1 (voting phase): if initial vote yes to prepare the commit protocol and send to the message sender site to receiver site. If the site didn't fail, the output is PREPARE-T, COMMIT, and READY then commit message sent from the receiver site YES.
- The transaction is successfully executed, for phase 1 (voting phase): if the receiver site fails, the output PREPARE-T, ABORT, and READY then abort message send from receiver site NO.
- The transaction is successfully executed, for phase 2 (pre-commit phase): if receiver site not failed, then commit protocol is authorized and it will commit data convert into pre-commit data protocols and its stored in multiple sites, i.e. **K-sites**. Because if one site failed then it will retrieve the data from other sites.
- The transaction is successfully executed, for phase 2 (pre-commit phase): receiver site gets the sender site information and it will execute the transaction message for PREPARE-T, READY, and ACK send to ACK sender sites.

- The transaction is successfully executed, for phase 3 (commit phase): sender site receives the ACK from the receiver site and send the DO-COMMIT transaction message to the receiver sites for HAVE COMMITED YES.
- The transaction is successfully executed, for phase 3 (commit phase): receiver site sends the HAVE COMMIT transaction message and get confirm COMMIT message to the sender sites, and END the transaction processing.

XI. COMPILATION CODING AND EXECUTION OF 1PC, 2PC, AND 3PC

To compile the code then create a file. After creating a file then it will execute are as follows [1]:

- Untar a tar file. Then it will create project name folder.
- Go inside of the project name folder. The three directories corresponding to 3PC, 2PC, and 1PC it will be found.
- To enter the 3PC directory. type and create the command prompt. This will compile and execute the code and generates the .exe files.
- To enter the 2PC directory. type and create the command prompt. This will compile and execute the code and generates the .exe files.
- To enter the 1PC directory. type and create the command prompt. This will compile and execute the code and generates the .exe files.
- To run the sender of 3PC and go to the project location/3PC/sender/simpdb/bin. Updated the receiverlist.txt and the IP address of the machines for the receiver sites.
- Create and type ./sender.exe receivinglists.txt.
- To run the receiver sites, enter the project directory/3PC/receiver/simpdb/bin. type ./receiver.exe < testcases No.>(optional).
- Similar types of 2PC and,
- Similar types of 1PC also.

XII. CONCLUSIONS

The conclusions of this study are: the detail about Three Phase Commit Protocol (3PC) for distributed database management system. It implements and avoids the blocking problem of the 2PC, and 1PC protocols its variants- Vote_commit, Vote_abort. The communication between sender sites (client sites) and receiver sites (server sites) is multi-threading of an optimal performance for committing protocols. When implement to the transaction manager while easily source code of plugging in simputer DBMS. In this paper, we have identified and investigated that an optimization of Two Phase Commit Protocol (2PC) eliminating/reducing non-blocking commit, logging activity, multi-threading design, improved throughput, reduce multi-casting, reduce the variance of distributed commit protocol based on Three Phase Commit Protocol (3PC) in a communication network infrastructure environment. Performances of the transaction mechanism such as

Remote Procedure Call (RPC) Applications depend upon Client sites and Server sites. The Future scope consisting to develop the transaction manager and it's incorporated in the combination of 3PC, 2PC, and 1PC i.e. 2-3PC protocols, which can implement simply and reusing the code for 1PC, 2PC, and 3PC during the implementation of a transaction manager through RPC communication networks.

REFERENCES

[1] Ozsu, Tamer M., and Valduriez, Patrick [1991], Principles of Distributed Database Systems, Prentice H.

[2] S. Ceri, M.A.W. Houtsma, A.M. Keller, P. Samarati: A Classification of Update Methods for Replicated Databases, via Internet, May 5, 1994.

[3] Mohan, C.; Lindsay, B.; and Obermarck, R. [1986], "Transaction Management in the R* Distributed Database Management System." ACM Transaction on Database Systems, Vol. 11, No. 4, December 1986, 379-395.

[4] Coulouris, J. Dollimore, T. Kindberg: Distributed Systems, Concepts and Design, Addison-Wesley, 1994.

[5] W. Cellary, E. Gelenbe, and T. Morzy. Concurrency Control in Distributed Database Systems. North-Holland, 1988.

[6] R. Ramakrishnan. Database Management Systems. McGraw-Hill Book Company, 1998.

[7] J.Gray, “Notes on database operating systems,” in Advanced Course: Operating Systems, ser. Lecture Notes in Computer Science, M. J. Flynn, J. Gray, A. K. Jones, K. Lagally, H. Opderbeck, G. J. Popek, B. Randell, J. H. Saltzer, and H.-R. Wiehle, Eds., Springer, vol. 60, 1978, pp. 393–481.

[8] P. A.Bernstein, V. Hadzilacos, and N. Goodman, Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.

[9] Weikum and G. Vossen, Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[10] Tony Flint, XA and Oracle controlled Distributed Transactions, June 2010.

[11] Serrano-Alvarado, P., Rouvoy, R., Merle, P.: Self-Adaptive Component-Based Transaction Commit Management. In: 4th Middleware Workshop on Adaptive and Reflective Middleware (ARM), Grenoble, France, ACM Press, AICPS of vol. 116, 2005, pp. 1–6.

[12] Oberg R., Mastering RMI: Developing Enterprise Applications in JAVA and EJB, John Wiley & Sons, 2001.

[13] Pitt E. and MCNiff K., java.rmi: The Remote Method Invocation Guide, Addison-Wesley, 2001.

[14] Oracle8 Server Distributed Database Systems, Oracle, 3-1-3-35.

[15] Lindsay et. al., Computation and Communication in R*: A Distributed Database Manager . ACM Trans. on Computer Systems, 2(1): 24-38, February 1984.

[16] R. Obermarck C. Mohan and B. Lindsay. Transaction Management in the R* Distributed Database Management System. ACM Trans. on Database Systems,11(4): 378-396, December 1986.

[17] H. Liskov B. M. Oki and R. W. Scheifler. Reliable Object Storage to Atomic Actions. In Proc. Tenth Symp. on Operating System Principles, page 147-159, ACM, December 1985.

[18] Nitesh Kumar, Ashish Kumar, Md. Imran Alam : Enhanced " One Phase Commit Protocol " In Transaction Management, International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-3, Issue-4, September 2013.

[19] Anshu Veda, Kaushal Mittal, Project Report One and Two Phase Commit Protocols, KReSIT, IIT Bombay, 20/10/2004.

[20] Peter Lannhult, Bjom Lindblom, Review of Non-Blocking Two-Phase Commit Protocols, Master Program in Computer System Engineering, Emausgatan 29C/ Generatorgatan 1.

[21] Pradeep K. Sinha, Distributed Operating Systems Concepts and Design, Prentice Hall of India Pvt Ltd, 2005.

[22] Jayant Harista, Krithi Ramamritham: Revisiting Commit Processing in Distributed Database Systems.

[23] Yousef J. Al-Houmaily, Panos K. Chrysanthis: 1-2 PC: The one-two phase atomic commit protocol, 2004.

[24] Silberchatz, Korth, Sudarshan: Database System Concepts- Fundamentals of Database, Tata Mc Graw Hills.

[25] Elmsari, Navathe, Fundamentals of Database Systems.

[26] Ashwini G. Rao, Memory Constrained DBMSs with updates, 2003.

[27] Nitesh Kumar, An Overview of Transparency in Homogeneous Distributed Database System, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), Volume-2, Issue-10, October 2013, ISSN: 2278-1323.

[28] Nitesh Kumar, Enhanced Two Phase Commit Protocol: In Transaction Management, in the proceeding of 7th International Conference on Advanced Computing and Communication Technologies (ICACCT-2013).

[29] O'Brien, J. & Marakas, G.M.(2008) Management Information Systems (pp. 185-189). New York, NY: McGraw-Hill Irwin.

[30] Connolly, Thomas; Begg, Carolyn; and Strachan, Anne, Database Systems, A Practical Approach to Design, Implementation and Management, Addison-Wesley, 1997. Database Management System." ACM Transactions on Database Systems, vol. 11 no. 4, December 1986, pp. 379-395.

[31] A b Skeen, Dale, "A Formal Model of Crash Recovery in a Distributed System". IEEE Transactions on SoftwareEngineering9 (3): doi: 10.11.09/TSE.1983.236608, pp. 219-228, May 1983.

[32] Keidar, Idit; Danny Dolev, "Increasing the Resilience of Distributed and Replicated Database Systems". Journal

[33] of Computer and System Sciences (JCSS) 57(3):http://webee.technion.ac.il/~idish/Abstracts/jcss.htm,pp. 309–324, Dec. 1998.

[34] This article incorporates public domain material from the General Services Administration document "Federal Standard 1037C".

[35] R. Abbott and H. Garcia-Molin a, "Scheduling 'Real-Time Transactions: a Performance Evahration", F&c. of 14th VLDB Conj., August 1988.