

Project Report

One and Two Phase Commit Protocols

Anshu Veda(04329022)
KReSIT, IIT Bombay

Kaushal Mittal(04329024)
KReSIT IIT Bombay

20/10/2004

Contents

1	Introduction	2
1.1	Purpose	2
1.2	Objective	2
1.3	Definitions, Acronyms, and Abbreviations. Abbreviations:	2
1.3.1	Acronyms	2
1.3.2	Definitions	2
2	Literature Overview	3
2.1	Two Phase Commit	3
2.2	One Phase Commit	3
2.3	Presumed Abort Two Phase Commit	4
2.4	Presumed Commit Protocol	4
2.5	1-2 PC Protocol	4
2.6	Three Phase Commit	5
2.7	Nonblocking Two phase commit	5
2.8	Utility in case of handheld	5
3	Implementation Details	5
3.1	Data Structures	5
3.2	Algorithm -Two Phase Commit	7
3.2.1	Coordinating Site	7
3.2.2	Participating Site	8
3.3	Algorithm - One Phase Commit	8
3.3.1	Coordinating Site	8
3.3.2	Participating Site	9
3.4	Directories and Files	9
3.4.1	Source Code Organisation	9
3.4.2	Source Files	9
3.4.3	Functions	10
3.5	Interfaces	11
3.5.1	Transaction Manager	11
3.5.2	LogManager	12
3.6	Languages Tools and Libraries	12
3.6.1	Programming Language	12
3.6.2	RPC Paradigm	12
3.6.3	Multi-Threading	12
3.6.4	Mutexes and Locks	13
4	Testing and Build	13
4.1	Testcases	13
4.1.1	Two Phase Commit Protocol - Testcases	14
4.1.2	One Phase Commit Protocol - Testcases	14
4.2	Compilation	15
5	Conclusion and Future Work	15

1 Introduction

1.1 Purpose

To design and implement an optimized commit protocol (1 PC and 2 PC) for transactions in distributed Database Management Systems which ensure atomicity and durability, in particular for handheld devices.

1.2 Objective

1. The protocol development involves algorithms for the coordinator and participating processes within Transaction Manager to provide global atomicity.
2. It guarantees uniform execution of transaction ensuring that all participating sites commit or all abort even when subject to network failures, coordinating and participating site failures.
3. The distributed computer system ensures reliability by providing redundant resources on different nodes. It leads to problems like lack of global state information, the possibility of partial failure and performing parallel operations. Hence, to maintain consistency an atomic transaction model is required.
4. In D-DBMS, database may be replicated or fragmented on multiple sites. A single transaction may involve the modifications in the database at multiple sites. For maintaining the consistency, more specifically, guarantying the ACID properties it is necessary for the transaction manager to ensure that the transaction gets successfully completed at all the sites participating in the transaction or on none of them. The Commit Protocol is to be designed to handle all these issues.
5. This protocol is intended to be applicable in DBMS for handheld devices, e.g. Simputer. It has to handle the issues common to the mobile environment like frequent disconnections, high communication prices etc.
6. Both the protocol 2PC and 1PC, have been designed and implemented

1.3 Definitions, Acronyms, and Abbreviations. Abbreviations:

1.3.1 Acronyms

D-DBMS Distributed Database management system

1-PC One Phase Commit.

2-PC Two phase commit.

1.3.2 Definitions

Coordinating site: The site that initiates the transaction.

Participating sites: The sites at which the transaction gets executed.

Deferred constraints: The constraints checks that are validated only at the time of the commit.

Voting phase: The phase in which the coordinating site communicate with

2 Literature Overview

The correctness of a distributed protocol lies in its Atomicity. The main issue to be considered is that, a, distributed transaction is a set of independent transactions executing at different sites (being processed by their respective transaction managers), but the result of transaction should be a consensus among all. To explain it further, when a distributed transaction finishes execution, in addition the local consistency that a usual transaction manager checks, it has to make sure that either all the sites that executed the same transaction commit or all abort. Thus the transaction is initiated by a coordinator, which ensures the above property. The Participating sites execute the transaction initiated by the coordinator independently on their local copies of Database. and commit/abort as per the final verdict of the coordinator.

In addition to maintain the communication between the two, proper Logs have to be generated which are to be later used by the recovery manager, in case a site failure occurs. Several designs have been suggested for the problem. The most relevant of these with their tradeoffs and advantages have been summarized below:

2.1 Two Phase Commit

This is the most widely used protocol which commences in two phases

Voting , to ensure that all sites are ready to commit

Decision , to ensure uniformity at abort or commit at all sites.

2PC is largely used in all distributed systems.

1. Advantages

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

2. Disadvantages

- Involves a great deal of message complexity.
- Greater communication overheads as compared to simple optimistic protocols.
- Blocking of site nodes in case of failure of coordinator.
- Multiple forced writes of log, which increase latency.
- Its performance is again a trade off, especially for short lived transactions, like Internet applications.

2.2 One Phase Commit

This protocol overlaps the voting phase with the execution of transaction and just has a decision phase. There are two implementation the Implicit Yes Voting and the Coordinate Log. These protocols are similar in all respects except the way they recover and assumptions they make about Locking Protocols and recovery semantics.

1. Advantages

- 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.

2. Disadvantages

- The greatest disadvantage is that it can only handle immediate consistency constraints as there is no voting phase. It can not handle the deferred consistency constraints.

In addition to these, there are certain variations of the above mentioned protocol.

2.3 Presumed Abort Two Phase Commit

This protocol is a variant of 2PC in which on Abortion the coordinator does not force write abort decision, just sends abort messages and the participants too do not write logs in case of abort.

1. Advantages

- Reduced abortion cost as compared to 2PC, both at coordinator and participant end.

2.4 Presumed Commit Protocol

This protocol is a variant of 2PC in which missing information about a transaction is presumed to be committed. However the coordinator force writes an initiation Record before sending prepare to ensure that the missing information of the Transaction is not misinterpreted as a commit after the coordinator's failure.

1. Advantages

- Reduced cost as compared to 2PC, both at coordinator and participant end.

2.5 1-2 PC Protocol

This protocol, is suitable for internet transactions. It uses both 1 and 2 PC protocols. Initially the nodes start with 1PC and if an undeferred consistency constraint is encountered they inform the coordinating site and the site responds with the voting phase thereby switching the protocol to 2PC.

1. Advantages

- Combines advantages of both major commit protocols to optimize performance.
- Short-lived transactions with just immediate consistency constraints are efficiently executed, without overheads
- Transactions with deferred consistency constraints can be handled.

2. Disadvantages

- Switching may be an Overhead at times.

2.6 Three Phase Commit

This is an extension of two phase commit to avoid blocking problem. An extra phase is added here where multiple sites participate in decision phase to commit.

1. Advantages

- Avoids Blocking under some situations.

2. Disadvantages

- Extra overheads.
- Increased complexity and costs.

2.7 Nonblocking Two phase commit

It is a variant of the 2PC protocol optimised for handling the problem of blocking. It handles three kinds of overheads namely, Forced writes, Message overhead, Convoy effect (transaction waiting for other transactions to commit). May involve overheads sometimes due to switching. It proposes the use of uniform multicast for the handling the message overhead, use of stable storage media as buffer for log messages instead of direct write to the physical stable storage media. In case of simputer we can use smart cards as a buffer for storing the logs and later on flushing it to the flash memory. It proposes to release the locks optimistically for handling the convoy effect. By optimistic commit it means that the other transaction may continue, in spite of the possibility of the abort of the transaction optimistically committed. It ensures that abort is restricted to one level.

2.8 Utility in case of handheld

The 2PC protocol requires network connectivity throughout its execution which may be an issue in case of handhelds. This may lead to frequent abortions and increased latency. 1PC alone is not so useful, so we suggest a combination of 1-2PC Protocol for handhelds with some optimizations. We have designed and implemented both 1PC and 2PC as a part of the project.

3 Implementation Details

The commit protocol begins as soon as the all the transactions from the participating site send their acknowledgements. Due to inavailability of interface from the Transaction Manager in current simpDB, we have simulated the transaction manager to start and execute the transaction and as soon as the transaction gets completed and attempts to execute the commit statement, our protocol begins.

3.1 Data Structures

We are listing here the important data structures for the Two phase commit protocol . The similar data structure have been used for the One phase commit protocol also. For details please refer to the documentation of the code.

1. Participant - It is a C structure consisting of the string hostname, port, username, password of the databse at the participating site.

2. GlobalTransactionStateData - It is a C structure maintaining the list of the participating site, transactionId, number of participating sites, and the state of the transaction.
3. List - It is a generic structure written to maintain the link list of any kind of structures or integers.
4. TransactionStateData - Maintained at the participating sites. It is a structure maintaining the transactionId assigned by the local transaction manager, transactionId given by the coordinator, Coordinator's Ip address, starttime of the transaction and the state of the transaction.
5. LogRecord -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 Coordinator site/Participant Site and the transaction Id.
6. 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 participating site as a response of the PREPARE-T message from the coordinator site. The PREPARE-T message is sent implicitly through Remote Method Invocation by invoking the PremmoitTransaction function of the participating site.
7. TransactionStates - We have defined constants for the different states of the transaction - START, PRECOMMIT, COMMIT_READY, COMMIT, ABORT, END.

3.2 Algorithm -Two Phase Commit

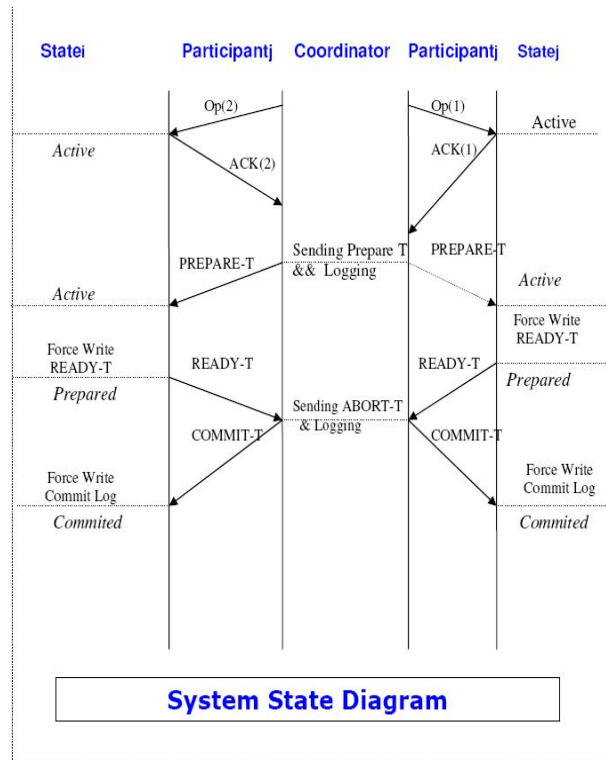


Figure 1: Two Phase Commit Protocol.

3.2.1 Coordinating Site

Arguments

- List of sites
- Transaction id
- Status of transaction (PRECOMMIT)

Begin

Force-Write Sending **PREPARE-T**

for each site in list

 Send **PREPARE-T** messages

 Recieve messages from all sites.

if each message=**READY-T**

 Force-Write Sending **COMMIT-T**

 send **COMMIT-T** message to all.

else if any message=**ABORT-T**

 Force-Write Sending **ABORT-T**

Send **ABORT-T** message to all.

End

3.2.2 Participating Site

Arguments

- Transaction id
- Status of local transaction (ABORT/Fail)

Begin

Wait for PREPARE-T message from coordinator

if transaction_state= **TRANS_PRECOMMIT**

Send **READY-T** messages

change the state of the transaction to **TRANS_COMMIT_READY**

else if transaction_state=**ABORT**

Send **ABORT-T** message

Receive message from Coordinator

if message=**COMMIT-T**

Force-Write **COMMIT**

Commit local Transaction

else if message=**ABORT-T**

Force-Write **ABORT-T**

Abort local Transaction

End

3.3 Algorithm - One Phase Commit

3.3.1 Coordinating Site

Arguments

- List of sites
- Transaction id
- Status of transaction (SUCCESS/FAILURE)

Begin

if transaction_status=**SUCCESS**

Force-Write Sending **COMMIT-T**

send **COMMIT-T** message to all.

else if transaction_status=**ABORT-T**

Force-Write Sending **ABORT-T**

Send **ABORT-T** message to all.

End

3.3.2 Participating Site

Arguments

- Transaction id
- Status of local transaction (ABORT/Fail)

Begin

Wait for message from coordinator
Recieve message from Coordinator
if each message=**COMMIT-T**
 Force-Write **COMMIT**
 Commit local Transaction
else if message=**ABORT-T**
 Force-Write **ABORT-T**
 Abort local Transaction

End

3.4 Directories and Files

3.4.1 Source Code Organisation

We have organised the code into the following directory structure. We are giving the directory structure of the project for Two phase commit protocol implementation. The similar directory structure follows for the 1PC commit also.

3.4.2 Source Files

The source code consists of the following files:

1. **transactionManager.c** - This file contains the main source code for the commit protocol along with the implementation of the assumed interface for the transaction manager. This file is different for participant and the coordinating sites. The file at the coordinating site, simulates the global transaction manager and contains the code to communicate with the participant sites. Participant sites have the local transactionmanager.
2. **logmgr.c** - This contains the code for the logmanager with functions to insert, flush and delete the log records.
3. **participantmethods.c** - This contains the interface for the methods call in the transaction manager corresponding to the start, precommit, commit and abort of the transaction.
4. **remote_participants_svc.c** - This is autogenerated stub using the rpcgen for the participating sites.
5. **remote_participants_clnt.c** - This is the autogenerated stub for the coordinator sites.
6. **remote_participants_xdr.c** - XDR routine to handle the input and output parameters of the RPC.
7. **transactionmanager.h** - Header file for the data structures of the transaction manager.

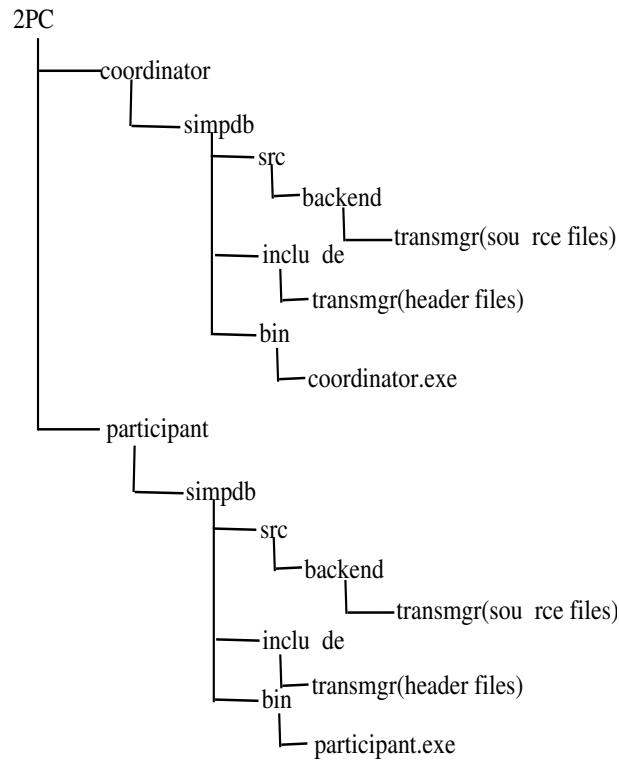


Figure 2: Directory Structure for 2PC

8. **logmgr.h** - The header file for the logmanager.
9. **remote_participants.h** - Autogenerated header file for RPC communication.

3.4.3 Functions

1. ***precommitGlobalTransaction*** - This create the threads for all the participant sites for invoking the *precommitTransaction* function on the participating site.
2. ***commitGlobalTransaction*** - This create the threads for all the participant sites for invoking the *commitTransaction* function on the participating site.
3. ***abortGlobalTransaction*** - This create the threads for all the participant sites for invoking the *abortTransaction* function on the participating site.
4. ***precommitParticipantTransaction*** - This calls the remote function *precommitTransaction* on the participant sites.
5. ***commitParticipantTransaction*** - This calls the remote function *commitTransaction* on the participant sites.
6. ***abortParticipantTransaction*** - This calls the remote function *abortTransaction* on the participant sites.

7. ***precommitTransaction*** - This function is in the transaction manager on the participant site to precommit the transaction, with the given transaction Id. It returns READY or ABORT message.
8. ***commitTransaction*** - This function is in the transaction manager on the participant site to commit the transaction with the given transaction Id. It returns ACK.
9. ***abortTransaction*** - This function is in the transaction manager on the participant site to abort the transaction with the given transaction Id. It returns ACK.

3.5 Interfaces

We have provided Interfaces for Transaction Manager and the Log Manager.

3.5.1 Transaction Manager

The transaction manager should implement this interface so that the code for the commit protocol can be plugged in the Simputer DB without any modification. The transaction manager that we have assumed is capable of Handling multiple transactions. We maintain a Link List of Transaction states modified atomically. This is ensured using semaphores. For each transaction we assign a transaction Id.

The functions that the interface contains are:

1. ***void StartTransaction (TransactionId)***
This method is used to initialise the transaction . While implementing the actual transaction manager you may need to change the prototype to initialise the transaction.
2. ***void AcquireResources (TransactionId)***
We have provided this dummy function so that the functions to be called for initialising the transaction , to acquire locks on the resources etc can be called within this function. And this function can be called from the StartTransaction Method as we currently do.
3. ***void ReleaseResources (TransactionId)***
We have provided this function as an interface to all the steps required to be executed while ending the transaction, like releasing the locks, freeing the memory, deleting or force writing the remaining logs etc. Currently we delete the remaining logs.
4. ***int getTransactionId ()***
The implementation of this method can be modified to meet the requirement of the transaction manager. It returns the integer value for the transaction id of the new transaction.
5. ***int tiggerDefferedConstraints (TransactionId)***
This function is invoked during the precommit phase. It can be used for invoking the code for checking the deferred constraints.
6. ***int CreateLog (TransactionId ,logType , message , coordinatorid)***
This function has been provided to be invoked from the functions for commit protocol. This has been done keeping in mind that the structure of log records may change with the actual implementation of the log manager. To avoid any modification in the code of the commit protocol, this method acts as an interface for creating and inserting the log records. With the change in the log structure only this function has to be changed.

3.5.2 LogManager

The log manager has been implemented keeping in mind the requirement of the actual log manager. The code for the logmanager can be reused as far as the insert, flush and delete are the requirements. The logs are maintained as a link list in the memory. The pointer to the last record inserted is maintained for the fast insertion of the log record. The records contain a pointer to the next log record for the same transaction. Thus the log records for all the transactions are in the same list, still tracing through the log records for a particular transaction is optimal and direct. It do not require traversing through the log records of other transaction. This makes flushing optimal. We have implemented three functions for logmanager.

1. *void insertLog (LogRecord)*

This function takes in the logRecord and inserts it into the log. Even if the structure of the log Record changes you need not change the implementation of the function.

2. *void flushLog (TransactionId)*

This function flushes all the logrecords for a given transaction Id on to the stable storage and frees the memory.

3. *void deleteLog (TransactionId)*

This function deletes all the log records in the memory for a transaction with the given transaction id.

3.6 Languages Tools and Libraries

3.6.1 Programming Language

ANSI C has been used for development of the code. C is most suited language for developing system software. The code has to be integrated with the existing code of the simputer DBMS, which has been written in C.

3.6.2 RPC Paradigm

Remote Procedure Calls have been used for communication between the Coordinating and Participating Sites. RPC is a powerful technique for constructing distributed applications. The Participating sites have RPC daemons, running on port. The transaction manager at the coordinating site, invokes the required methods on each of the site in the list of participating sites for a particular transaction. This is equivalent to sending a message (invoking a method) and getting ACK(results) in response. We have used Unix-C available **rpc** library for compiling stubs for communication.

3.6.3 Multi-Threading

In a distributed transaction scenario, where there are multiple transactions progressing at a site, with independent states, multi-threading is a necessity. We have used the Unix **pthread** Library and provided multi-threading support to both participants and coordinating sites. This facilitates independent messages for prepare, commit/abort to the participating sites and the participating site can also have multiple transactions going on at its end from different coordinators.

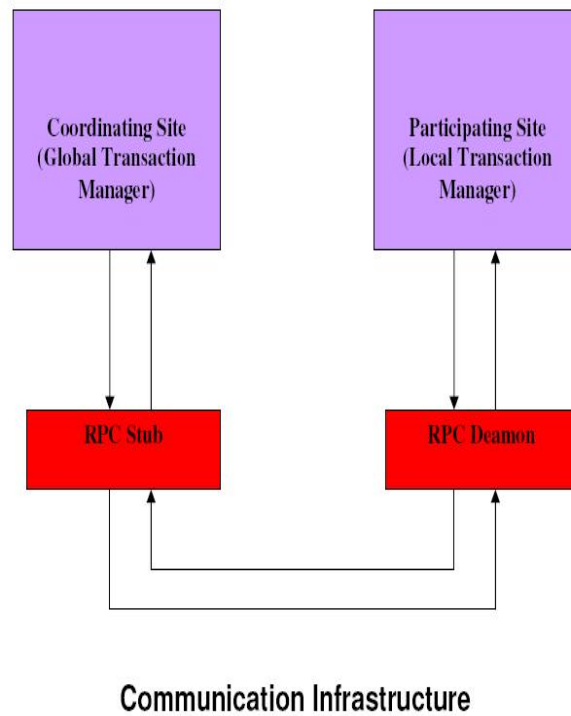


Figure 3: Communication Interface

3.6.4 Mutexes and Locks

We have taken care of all the synchronization aspects that can occur in a multiple transaction distributed environment, so as to ensure that all transaction commence without any interaction. For this purpose semaphores and locks have been used.

4 Testing and Build

In this section we discuss the various testcases and the way to compile the files .

4.1 Testcases

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 participating site for the participants once. and then you need to run the CoordinateSite 9 times for each of the testcases. We have written a simulation function in the participating site for demonstration that takes care of different testcases automatically. In that function we change the state of the transaction to Precommit 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.

4.1.1 Two Phase Commit Protocol - Testcases

1. Transaction is successfully executed. It reaches the precommit state and votes yes. No delays takes place so no site fails. The output is PREPARE_T and COMMIT log in the coordinator sites log file and the READY and COMMIT message in the logs of the participating sites.
2. Transaction is successfully executed. It reaches the precommit state and votes yes. During precommit delay takes place at participating sites so participating site fails. The output is PREPARE_T and ABORT log in the coordinator sites log file and the READY and ABORT message in the logs of the participating sites.
3. Transaction is successfully executed. It reaches the precommit state and votes yes. During commit delays takes place so participating site fails. The output is PREPARE_T and COMMIT log in the coordinator sites log file and the READY and COMMIT message in the logs of the participating sites except for the sites where commit delayed.
4. Transaction is successfully executed. It reaches the precommit state, deferred constraints failed and it votes NO. No delays takes place so no site fails. The output is PREPARE_T log in the coordinator sites log file and the READY and ABORT message in the logs of the participating sites.
5. Transaction is successfully executed and reaches the precommit state. Deferred constraints failed and site votes NO. During precommit delay takes place at participating sites so participating site fails. The output is PREPARE_T log in the coordinator sites log file and the READY and ABORT message in the logs of the participating sites.
6. Transaction is successfully executed and reaches the precommit state. Deferred constraints failed and site votes NO. During abort delays takes place so participating site fails. The output is PREPARE_T log in the coordinator sites log file and the READY and ABORT message in the logs of the participating sites except for the sites where abort delayed.
7. Transaction fails and goes to abort state ,sites votes NO. No delays takes place so no site fails. The output is PREPARE_T log in the coordinator sites log file and the READY and ABORT message in the logs of the participating sites.
8. Transaction fails and goes to abort state ,sites votes NO. During precommit delay takes place at participating sites so participating site fails. The output is PREPARE_T log in the coordinator sites log file and the READY and ABORT message in the logs of the participating sites.
9. Transaction fails and goes to abort state , votes NO. During abort delays takes place so participating site fails. The output is PREPARE_T log in the coordinator sites log file and the READY and ABORT message in the logs of the participating sites.

4.1.2 One Phase Commit Protocol - Testcases

1. 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 coordinator site and the participant site.

2. Transaction is successfully executed and reaches the commit state. during commit delay takes place at participating sites so participating site fails. The expected output is the COMMIT log on the coordinator site.
3. Transaction fails and reaches the abort state. no delays takes place so no site fails. The expected output is the ABORT log on the participating site.
4. Transaction fails , and reaches the abort state . during abort delays takes place at participating sites so participating site fails. The expected output is the ABORT log on the participating sites.

4.2 Compilation

To compile the code we have given a make file. Steps to run the code are as follows:

1. Untar the tar file. It will create a folder name project.
2. Go inside the folder project. The two directories corresponding to 1PC and 2PC will be found.
3. For 2PC enter the 2PC directory. type make at the command prompt. This will compile the code and generate the EXE files.
4. For 1PC enter the 1PC directory. type make at the command prompt. This will compile the code and generate the EXE files.
5. For running the coordinator of 2PC go to the location project/2PC/coordinator/simpdb/bin. Modify the participatinglist.txt with the IP of the machines where participating sites are running.
6. Type ./coordinator.exe participatinglists.txt
7. For running the participating sites, enter the directory project/2PC/participant/simpdb/bin. Type ./participant.exe < Testcase No.>(optional).
8. similar steps for 1PC also.

5 Conclusion and Future Work

We have studied in detail the commit protocols for distributed database management systems. We have implemented the 1PC and 2PC protocols with its variants - Presumed abort and Presumed Commit. We have ensured that the communication between the participating sites and the coordinating sites is multithreaded for optimal performance of the commit protocols. The source code can easily be plugged into simputer DBMS while implementing the transaction manager. The Future work consist of developing the transaction manager and incorporating in it the combination of 1PC and 2PC i.e. 1-2PC protocol which can simply be implemented by reusing the code for 1Pc and 2PC developed by us, during the implementation of transaction manager.

References

- [1] Ashwini G. Rao:*Memory Constrained DBMSs with updates*, 2003.
- [2] Jayant Harista, Krithi Ramamritham:*Revisiting Commit Processing in Distributed Database Systems*.
- [3] Yousef J. Al-Houmaily, Panos K. Chrysanthis:*1-2 PC: The one-two phase atomic commit protocol.*, 2004.
- [4] Silberchatz, Korth, Sudarshan:*Database System Concepts- Fundamentals of Database*, Tata Mc Graw Hills.
- [5] Elmsari, Navathe:*Fundamentals of Database Systems..*