RC-OCP: Reverse Coordinated Optimized Commit Protocol

Ganpat Singh Chauhan[1], Mukesh Kumar Gupta[2]
[1,2] Department of Computer Engineering
[1,2]Swami Keshvanand Institute of Technology Management & Gramothan, Jaipur
Email- [1]ganpatchauhan@gmail.com, [2]mukeshgupta@skit.ac.in

Abstract: **With the growth of trends in computing and requirement for developing a reliable computing environment the need for performing tasks in distributed manner has gained much importance. Atomic commit protocols are used to preserve the ACID property in distributed systems when several sites need to update their database with the same information. A variety of protocols have been proposed for last few decades, but among all Two-Phase Commit (2PC) are most widely used. 2PC is a blocking protocol, i.e. If a coordinator fails while the participants are waiting for a decision in its uncertain state, all the participants remain in a blocked state till the coordinator recovers and terminates the transaction. This is undesirable as these sites may be holding locks on the resources. In the event of message loss, the 2PC protocol will result in the sending of more messages. As an alternative, non-blocking atomic commit protocol has been suggested: This non-blocking protocol is Three-Phase Commit (3PC) protocol, which requires an extra phase (pre-commit) to remove blocking state.**

Although, the existing protocols are sufficient to ensure that ACID properties are maintained in a distributed transaction environment, but the substantial cost associated with the normal transaction execution adversely affects the performance of the system. These protocols require so many messages transfer from the coordinator to participants and vice versa during different phases and corresponding increase communication and time complexity and hold the locks acquire by different participants till the end of last phase, forcing other transactions also to be blocked just because of the objects that are locked. All the existing protocols give the same performance for both deferred and immediate consistency constraints databases. Lots of protocols are being proposed and mainly the concentration was to make the 2PC protocol non-blocking or to minimize the blocking possibilities in 2PC. There has been a renewed interest in developing and optimization of more efficient ACPs. This is also crucial in modern electronics and e-commerce environment which are characterized by high volume of transactions at several levels.

This paper is an effort to propose a new technique to optimize atomic commit protocols by optimizing voting phase based on deferred and immediate consistency constraints. Also an attempt has been made to propose reduction in communication complexity which will require less number of messages to be shared between coordinator and participants, the cost of execution as well as time delays.

Keywords: **Distributed Database System, Two Phase Commit Protocol (2PC), Three Phase Commit Protocol (3PC), Non-Blocking Commit Protocol, Deferred and Immediate consistency constraints.**

1. INTRODUCTION

It is found and proved by investigating the reliability of atomic commit protocols that blocking is unavoidable after certain site or network failures [1]. The existing protocols are sufficient enough to maintain ACID properties in the distributed transaction environment, but at a substantial cost during normal transaction execution, which adversely affects the performance of the system [2]. Atomic commit protocols require so many messages transfer from coordinator to participants and vice versa during phase1, phase2 and/or phase3 and corresponding increase communication and time complexity and hold the locks acquire by different participant till the end of last phase; this forces other transactions also to be blocked just because of the objects that are locked. Despite all the drawbacks of 2PC protocol like blocking, its high cost of logging and the number of messages [3]; it is supported by all commercial database systems. For these reasons, there has been a renewed interest in developing more efficient ACPs and optimization for modern e-commerce environment and electronics services that are characterized by high volume of transactions [4]

The structure of the paper: Section II, III and IV introduce review of the 2PC, 3PC and other protocols respectively. Section V shows survey concludes, Section VI gives proposed methodology, and Section VII gives performance comparison and analytical evaluation.

2. TWO-PHASE COMMIT (2PC) PROTOCOL

A. General Description

The most used protocol in distributed transaction processing systems is 2PC Protocol.In 2PC a single master (coordinator) is used to collect each slave's (participant's) status of the work; the coordinator decides to commit the transaction if all participants are ready to commit, otherwise the transaction is aborted.

B. Issues with 2PC

There are two major issues with 2PC:

1) Blocking: The 2PC goes to a block state by the failure of the coordinatorandallparticipantsareliveandnoparticipantishaving knowledgeofdecisionfromcoordinatorwhentheparticipantsare inanuncertainstate.Theparticipantskeeplocksonresourcesuntil

they receive the next message from the coordinator after its recovery.

2) State Inconsistency: Global State Vector (GST) in commit protocols works as a container of states for every participating node regarding a single transaction. The global transition state comprises this state vector and outstanding messages in the network. I call a state inconsistent when its global state vector contains both the commit and abort states. This inconsistency can be observed using a state vector, particularly when the participant is in its pre-commit state and fails. The coordinator shows the committed state after sending a commit message, but for the failed participant the protocol is declared no resilient in for assigning new state [3, 5].

3. THREE PHASE COMMIT PROTOCOL (3PC)

A. General Description

Three-Phase Commit Protocol (3PC) is a non-blocking protocol, contrary to the 2PC. Here a new state called "precommit" is introduced for the coordinator and participants. The coordinator gets to this "pre-commit" state only if all other participants have voted "YES" to commit. In case this state is not reached, the participant will abort and release the blocked resources after a specific time. When the coordinator gets the "pre-commit" state, then there is only one option to abort the transaction and that is a timeout, which corresponds to a failure of a participant, otherwise the transaction gets completed with an acknowledgement from the participants. It is also possible that the coordinator fails at this state; even then it will proceed for global commit.

B. Issues with 3PC

Even if 3PC present the non-blocking scenario as compare to 2PC but still 2PC is most popular atomic commit protocol. The major issues with 3PC are as:

3PC protocol is problematic only when there are multiple sites failures. For example, let's consider a case where the coordinator is in "pre-commit" state and fails just after sending a commit message and the participant also fails just before or after receiving this message. So by its failure, the participant moves to the aborted state, but according to the protocol specifications given in, the coordinator goes to the commit state, either it fails or receives acknowledgement. Hence, the coordinator moves to the committed state without receiving acknowledgement and the failed participant moves to the aborted state without sending the acknowledgement. In this way, coordinator and participant show different final states due to their failures. The 3PC is also problematic when overhead and time is crucial because it require extra phase to make protocol non-blocking, which require extra space and time to complete the task. It involves a great deal of overhead as compared to simple protocols and multiple logs forced write which increase latency. For short lived transactions, like Internet application performance of 3PC is again a trade off [3, 6].

4. OTHER PROTOCOLS

The number of communication steps, the number of log writes and its execution time at the coordinator and at each participant influence the efficiency of a commit protocol. The blocking or no blocking nature and difference in recovery procedures are other important factors that have a vital impact on the overall commit protocol performance. Below given protocols are some of the variants of 2PC and 3PC to get equally good performance.

Presumed Abort (prA) and Presumed Commit (prC): Database research has been done on ways to get most of the benefits of the two-phase commit protocol while reducing costs by protocol optimizations and protocol operations saving under certain system's behavior assumptions.

Presumed abort or Presumed commit are common such optimizations. An assumption about the outcome of transactions, either commit, or abort, can save both messages and logging operations by the participants during the 2PC protocol's execution. For example, when presumed abort, if during system recovery from failure no logged evidence for commit of some transaction is found by the recovery procedure, then it assumes that the transaction has been aborted, and acts accordingly. This means that it does not matter if aborts are logged at all, and such logging can be saved under this assumption. Typically a penalty of additional operations is paid during recovery from failure, depending on optimization type. Thus the best variant of optimization, if any, is chosen according to failure and transaction outcome statistics.

Both PrA and PrC seek to reduce commit process overhead by reducing acknowledge messages and forced log writes in the decision phase, while the voting phase remains the same as for 2PC. PrA is preferable where the number of aborted transactions is more than the number of committed transaction; prC is preferred in systems where the number of committed transactions is more than the number of aborted transactions, a common situation considering present system reliability [7].

One Phase Commit: In One-Phase Commit (1PC) protocol the Early Prepare (EP) protocol forces each cohort to enter a prepare state after the execution of each operation. It makes cohort's vote implicitly YES and this protocol exploits the Presumed Commit as well. But a coordinator may have to force multiple membership records, because the transaction membership may grow as transaction Execution progresses. The main drawback comes from the fact that the log of each operation has to be written in the cohort's log disk per operation, it leads to a serious disk blocking time. Only if every sever has a stable storage so that log forces are free, EP can be considered to be used. Above all, 1PC is however rarely considered in practice because of strong assumption it requires from the distributed transaction system [3, 7, 8, 9].

Optimized Commit Protocol works on reducing the waiting time of lock by releasing the locks the current transactions hold. Even if the committing transaction is aborted there is no possibility of cascading aborts, since the lock releasing is done in a managed manner. Due to this reduction of the blocking due

to releasing of locks occupied on prepared data this protocol has a better performance [3, 4].

Extended Three Phase Commit Protocol is one of the important issue of distributed system is to improve concurrency control. Because of absence of global clock and lack of shared memory; concurrency control in distributed system is very difficult task. The basis of this protocol is the division of all sites into two groups depending upon the number of queries generated and importance of the queries at these sites. The sites where more queries are generated are considered as primary sites and those having less are considered as secondary sites. This protocol works only for transactions that access a single database object [10].

Reducing the blocking in 2PC employing backup sites In distributed database systems, the blocking phenomena reduce availability of the system since the blocked transactions keep all the resources until they receive the final command from the coordinator after its recovery. Although 3PC protocol eliminates the blocking problem, it involves an extra round of message transmission, which further degrades the performance of DDBSs. This protocol proposes a backup commit (BC) by including backup phase to 2PC protocol. In this, one backup site is attached to each coordinator site. After receiving responses from all participants in the first phase, the coordinator communicates its decision only to its backup site in the backup phase. Afterwards, it sends the final decision to participants. When blocking occurs due to the failure of the coordinator site, the participant sites consult coordinator's backup site and follow termination protocols. In this way, BC protocol achieves non-blocking property in most of the coordinator site failures. However, in the worst case, the blocking can occur in BC protocol when both the coordinator and its backup site fail simultaneously. If such a rare case occurs, the participants wait until the recovery of either the coordinator site or the backup site. BC protocol suits best for DDBS environments in which sites fail frequently and messages take longer delivery time [11].

5. PERFORMANCE AND SURVEY CONCLUSION

After reviewing all the above discussed commit protocols ,an evaluation is done as shown in TABLE 1 and Fig. 1 based on some performance parameters like message complexity, time complexity, number of log writes and blocking/non-blocking. Here n represent the number of participants.

It is found that the existing protocols are sufficient enough to maintain ACID properties in the distributed transaction environment [1, 12]. Protocols are being proposed and mainly the concentration was to make the two phase commit protocol non-blocking or to minimize the blocking possibilities in distributed transaction. Atomic commit protocols that are summarized in Table1 it is clear that they all require so many messages transfer from coordinator to participants and vice versa during different phases and corresponding increase communication and time complexity and hold the locks acquire by different participant till the end of last phase. This forces other transactions also to be blocked just because of the objects that are locked. It is observed that most the commit protocols give same performance for both immediate and deferred consistency constraints and some sort of optimization is required in existing protocols to decrease number of messages exchange, log writes and message complexity; This is crucial in modern application services and e-commerce environment that required high volume of transactions. [13, 14, 15].

Table 1 : The Cost Of Different Protocols To Commit A Transaction

Protocol	2PC	3PC	prA	prC	1PC	Extended -3PC	2PC Backup
Blocking	High	Very Less	High	High	Less	Very Less	Very Less
Time Complexity (rounds)	4	6	4	3	2	6	4
Message Complexity (Complexity)	4n	6n	4n	3n	2n	6n	4n
Log Write	2+2n	2+2n	2+n	2+2n	2+n	2+2n	2+2n

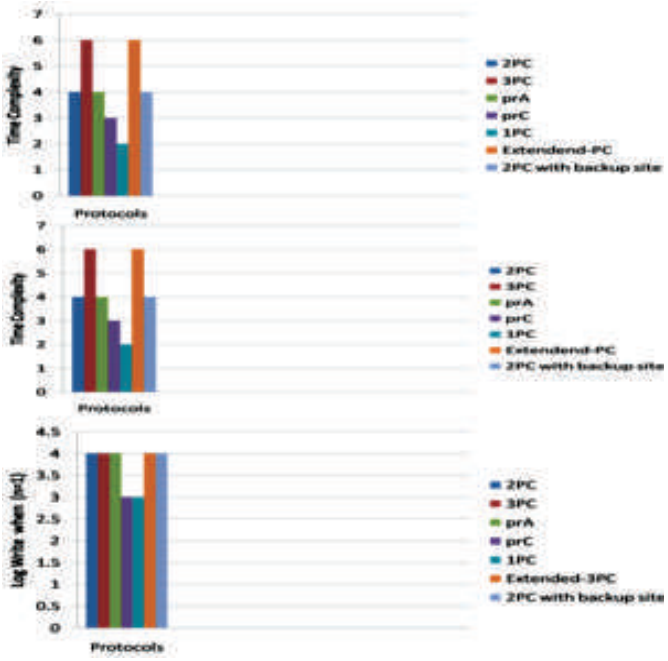

Fig. 1: The cost of different existing protocols to commit a transaction

6. PROPOSED PROTOCOLS

6.1 Objective

The 2PC protocol is one of the most widely used ACP. 2PC and 3PC ensures atomicity and interdependent recovery, but at a substantial cost during normal transaction execution, which adversely affects the performance of the system. This is due to the cost associated with its message complexity, time complexity and log writes. For this reason, there has been a re-newed interest in developing more efficient and optimized ACPs. This is crucial in modern application services and e-commerce environment that required high volume of transactions.

The proposed optimized protocols, which require less number of messages to be shared between coordinator and participants; corresponding decrease communication complexity and time

complexity for the transaction. The protocols are based on deferred or immediate consistency constraints where optimization of voting phase is done [16].

6.2 Optimized Two Phase Commit (O-2PC) Protocol

In Distributed Database Environment once the coordinator has initiated the transaction and distributed it among different participant nodes, the Local Transaction Manager (LTM) at each participant node is now responsible to deal with local sub-transaction. Once the Coordinator decides to commit the transaction, it asks for all the participants whether to COMMIT or ABORT the current running transaction. Both 2PC and 3PC requires initiating the step from the coordinator side (voting phase) to start commit protocol. This approach requires so many messages transfer from coordinator to participants and vice versa during phase1, phase2 and/or phase3 and corresponding increase overhead and hold the locks acquire by different participant till the end of last phase. This forces other transactions also to be blocked just because of the objects that are locked [17, 18].

The proposed solution is that once the transaction has initiated by coordinator and all the participant nodes are identified and their LTM is activated, the coordinator will then continue with its own work and never ask participants for their voting decision. As the coordinator maintains a log file for all the current transaction's participants, as soon as any participant finishes its works it will respond to coordinator with YES (for Commit) or NO (for Abort) decision message (in case of immediate consistency constraints). Otherwise if transaction needs deferred consistency constraints to be applied then all the participants are required to send their voting decision just after commit protocol is started. So when the transaction is completed and commit protocol is started there are two possible scenarios:

1. If the coordinator already has all the votes from all the participants with it (immediate consistency constraints); the coordinator decides on the basis of vote messages from all the participants.

2. Else (in case of deferred consistency constraints) Coordinator waits for the vote messages from all the participants. As soon as coordinator received all the vote messages then it will decides on the basis of vote messages.

In proposed protocols whether it is deferred or immediate consistency constraints; initiation of the commit protocol will be from participant side; that's why I called the proposed protocol as Reverse Coordinated Atomic Commit Protocols.

Basics Structure of O-2PC: assuming no failures O-2PC goes roughly as follows:

1. There are two possible scenarios:

a. If the coordinator already has all the votes from all the participants with it (in case of Immediate Consistency Constraints) then:

If all the votes are YES and the coordinator's own vote is YES, then the coordinator decides commit and multicast COMMIT message to all participants. Otherwise, the coordinator decides abort and multicast ABORT message to all participants that voted YES (those that voted NO already decided Abort). Goto step 3.

b. Else (in case of deferred consistency constraints) each participant will send its voting message to the coordinator: YES or NO. If the participant vote is NO, it decides abort and stops.

2. All the vote messages from all participants are collected by the coordinator. If all the votes are YES and the coordinator's own vote is YES, then the coordinator decides commit and multicast COMMIT message to all participants, Otherwise, the coordinator decides abort and multicast ABORT message to all participants that voted YES (those that voted NO already decided Abort)., then the coordinator stops.

3. All those participants that have voted YES wait for a decision (COMMIT or ABORT) message from the coordinator and when received, it proceeds accordingly and stops.

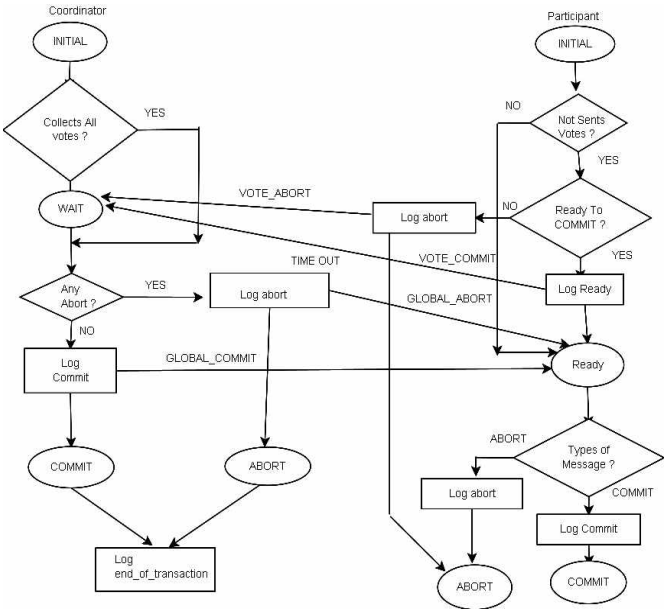

Fig. 2: Flow chart of Optimized- 2PC Protocol

The two phases of O-2PC (deferred consistency constraints) are the voting phase steps (1(b)) and the decision phase (steps (2) and (3)). For Immediate consistency constraints scenario voting phase is completely eliminated and we have only the decision phase (steps 1(a) and (3)). A participant's uncertainty period starts when it sends a YES to the coordinator and ends when it receives a COMMIT or ABORT. Since the coordinator decides as soon as it votes - with the knowledge, of course, of the participants votes the coordinator has no uncertainty period. Fig. 2 shows the flow chart of the above described steps for O-2PC.

6.3 State Transition Diagram for O-2PC:

Fig. 3 shows state transition diagram of O-2PC for immediate consistency constraints. In case of immediate constraints the coordinator decides whether to commit or abort the transaction as soon as commit protocol is started because the coordinator already has all the vote messages with it. Similarly each participant that has voted YES, enters into WAIT state and waits

for decision message from coordinator as commit protocol is started. It is clear that in case of immediate consistency constraints voting phase is completely eliminated and correspondingly reduces number of rounds, exchanged messages and log writes. Fig. 4 shows state transition diagram of O-2PC for deferred consistency constraints. In case of deferred constraints the coordinator enters into WAIT state as soon as commit protocol is start and waits for vote messages from each participant. The coordinator decides whether to commit or abort the transaction. As soon as all the vote messages are collected from each participant and multicast its decision to each participant that has voted YES.

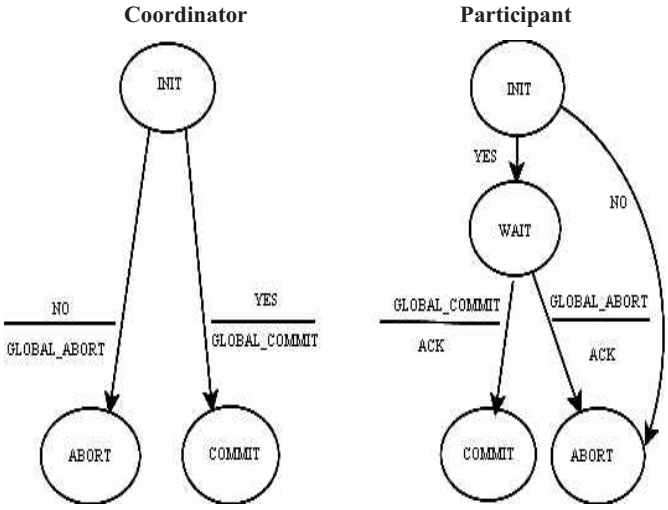

Fig. 3: State Diagram of O-2PC for Immediate Consistency Constraints

Similarly each participant that has voted YES enters into READY state and sends their vote message to the coordinator as commit protocol is started. It is clear that in case of deferred consistency constraints voting phase is partially eliminated and correspondingly reduces number of rounds, exchanged messages and log writes.

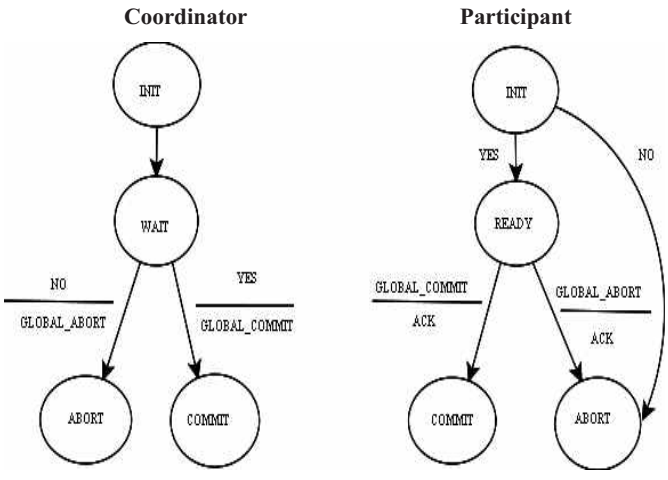

Fig. 4: State Diagram of O-2PC for Deferred Consistency Constraints

6.4 Timeout Actions

In Distributed system there are situations where a process needs to wait for message to arrive. To avoid the infinite waiting, the concept of timeout is used.Therefore; a process must keep some information in stable storage, specifically in the Distributed Transaction (DT) log.

There are two places in above protocol where a process is waiting for a message: in the beginning of (steps (2) and (3)).In step (2) the coordinator is waiting for YES or NO messages from all the participants and has not yet reached any decision. In addition, no participant can have decided Commit. Therefore, the coordinator can decide Abort but must send ABORT to every participant from which it received a YES.

A participant p1 that voted YES is waiting for a decision (COMMIT or ABORT) from the coordinator, in step (3). At this point p1 is uncertain. So, unlike the previous scenario (process unilaterally decision), here to find out what to decide the participant must consult with other processes. This consultation is by meant of a termination protocol.

The conventional termination protocol is the following: p1 remains blocked until it can re-establish communication with the coordinator. Then, the coordinator can tell p1 the appropriate decision. The coordinator can surely do so, since it has no uncertainty period. This termination protocol satisfies condition ACP5, because if all failures are repaired, p1 will be able to communicate with the coordinator and thereby reach a decision.

The drawback of this termination protocol is that p1 may be blocked unnecessarily, For example, suppose there are two participants' p1 and p2. The coordinator might send a COMMIT or ABORT to p2 but fail just before sending it to p1. Thus, even though p1 is uncertain, p2 is not. If p1 can communicate with p2, it can find out the decision from p2. It need not block waiting for the coordinator's recovery.

This suggests the need for participants to know each other, so they can exchange messages directly (without the interference of the coordinator). Recall that our description of the atomic commitment problem states that the coordinator knows the participants and the participants know the coordinator, but that the participants do not initially know each other. It is the case that the coordinator attaches the list of the participant's identities with the transaction submission messages to each of them.

This discussion leads us to the cooperative termination protocol: A participant p1 that times out while in its uncertainty period sends a DECISION-REQ message to every other process, p2, to inquire whether p2 either knows the decision or can unilaterally reach one. In this scenario, p1 is the initiator and p2 a responder in the termination protocol. There are three cases:

1. p2 has already decided commit (or abort): p2 simply sends a COMMIT (or ABORT) to p1, and p1 decides accordingly.

2. p2 has not voted yet: p2 can unilaterally decide Abort. It then sends an ABORT to p1, and p1 therefore decides Abort.

3. p2 has voted YES but has not yet reached a decision: p2 is also uncertain and therefore cannot help p1 reach a decision.

With this protocol, if p1 can communicate with some p2 for

which either (1) or (2) holds, then p1 can reach a decision without blocking. On the other hand, if (3) holds for all processes with which p1 can communicate, and then p1 is blocked.

This will persist until enough failures are repaired to enable p1 to communicate with a process p2 for which either (1) or (2) applies. At least one such process exists, namely, the coordinator.

In summary, even though the cooperative termination protocol reduces the probability of blocking, it does not eliminate it. However, even with the cooperative termination protocol, this protocol is subject to blocking even if only site failures occur.

6.5 Recovery from failure

Consider a process p1 recovering from a failure; p1 must reach a decision consistent with that reached by the other processes - if not immediately upon recovery, then some time after all other failures are also repaired.

Suppose that when p1 recovers; it remembers its state at the time it failed. If p1 failed before having sent YES to the coordinator (step (2) of O-2PC), then p1 can unilaterally decide abort and this is part of Transaction execution. Also, if p1 failed after having received a COMMIT or ABORT from the coordinator or after having unilaterally decided Abort, then it has already decided. In these cases, p1 can recover independently [19].

However, if p1 failed while in its uncertainty period, then it cannot decide on its own when it recovers. Since it had voted YES, it is possible that all other processes did too, and they decided commit while p1 is down. But it is also possible that some processes either voted NO or didn't vote at all and abort was decided. p1 can't distinguish these two possibilities based on information available locally and must therefore consult with other processes to make a decision. This is a reflection of the inability to have independent recovery.

In this scenario, p1 is exactly in the same state as what it had timed out waiting for a decision (COMMIT or ABORT) from the coordinator. Thus, p1 can reach a decision by using the termination protocol. Since p1 may be able to communicate only with uncertain processes, it may be blocked

To remember its state at the time it failed, each process must keep some information in its site's DT log, which survives failures. Of course, each process has access only to its local DT log. Assuming that the cooperative termination protocol is used, here is how the DT log is managed.

1. When the coordinator is initiated by commit protocol, it writes a start_O-2PC record in the DT log. This record contains the identities of the participants.

2. If a participant votes YES, it writes a yes record in the DT log, before sending YES to the coordinator. This record contains the name of the coordinator and a list of the other participants (which is provided by the coordinator). If the participant votes NO, it writes no either before or after the participant sends NO to the coordinator.

3. Before the coordinator sends COMMIT to the participants, it writes a commit record in the DT log.

4. When the coordinator sends ABORT to the participants, it writes an abort record in the DT log. The record may be written before or after sending the messages.

5. A participant writes a commit (or abort) record in its DT log after received decision (COMMIT or ABORT).

In this discussion, writing a commit or abort record in the DT log is the act by which a process decides commit or abort.

At this point it is appropriate to comment briefly on the interaction between the commitment process and the rest of the transaction processing activity. Once the commit (or abort) record has been written in the DT log, the DM can execute the Commit (or Abort) operation. There are numbers of details regarding how writing commit or abort records to the DT log relates to the processing of the commit or abort operations by the DM. For example, if the DT log is implemented as part of the DM log, the writing of the commit or abort record in the DT log may be carried out via a call to the Commit or Abort procedure of the local DM. In general, such details depend on which of the algorithms used by the local DM.

A distributed transaction's state that is executing at S can be determined by examining its DT log as soon as S recovers from a failure:

- If the DT log contains a start_O-2PC record, then S was the host of the coordinator. If it also contains a commit or abort record, then the coordinator had decided before the failure. If neither record is found, the coordinator can now unilaterally decide Abort by inserting an abort record in the DT log. For this to work, it is crucial that the coordinator first insert the commit record in the DT log and then send COMMIT (point (3) in the preceding list).
- If the DT log doesn't contain a start_O-2PC record, then S was the host of a participant.

There are three cases to consider:

1. The DT log contains a commit or abort record. Then the participant had reached its decision before the failure.

2. The DT log does not contain a yes record. Then either the participant failed before voting or voted NO. (This is why the yes record must be written before YES is sent; see point (2) in the preceding list.) It can therefore unilaterally abort by inserting an abort record in the DT log.

3. The DT log contains a yes but no commit or abort record; then the participant failed while in its uncertainty period. It can try to reach a decision using the termination protocol. Recall that a yes record includes the name of the coordinator and participants, which are needed for the termination protocol.

A Fig. 5 gives the O-2PC protocol algorithm and the cooperative termination protocol, incorporating the preceding discussion on timeout actions and DT logging activity. We use multicast, send and wait-for statements for inter-process communication.

Although I have been presenting ACPs for a single transaction's

termination, it is clear that the DT log will contain records describing the status of different transactions relative to atomic commitment. Thus to avoid confusing records of different transactions, the start_O-2PC, yes, commit, and abort records must contain the name of the transaction to which they refer. In addition, it is important to garbage collect DT log space taken up by outdated information. There are two basic principles regarding this garbage collection:

GCC1: A site cannot delete log records of a transaction T from its DT log until at least after its RM has processed RM-Commit (T) or RM-Abort (T).

GCC2: At least one site must not delete the records of transaction T from its DT log until that site has received messages indicating that RM-Commit (T) or RM-Abort (T) has been processed at all other sites where T executed.

6.6 Coordinator's algorithm

```
write start_O-2PC and init record in local DT;
if all vote messages are not there with the coordinator /* for deferred constraints */
{
wait for vote messages (YES/ NO) from all participants;
if timeout /* in case of failure of any participant*/
{
let P[n] be the processes from which YES was received;
write GLOBAL_ABORT record in local DT;
multicast GLOBAL_ABORT to all processes in P[n];
return;
}
}
if all messages are YES and coordinator also voted YES
{
write GLOBAL_COMMIT to local DT;
multicast GLOBAL_COMMIT to all participants;
}
else / * only some of the processes voted YES * /
{
let P[n] be the processes from which YES was received;
write GLOBAL_ABORT record to local DT;
multicast GLOBAL_ABORT to all processes in P[n];
}
return;
```

Participant's algorithm

```
write init record in local DT;
if vote message is not yet sent to the Coordinator /* for deferred constraints */
{
if participant's vote is YES
{
write yes to local DT;
send YES to the coordinator;
}
else if participant's vote is NO
{
write no to local DT;
send NO to the coordinator;
return;
}
}
wait for decision (commit or abort) from coordinator
if timeout
{
initiate termination protocol;
}
if decision message received from coordinator is GLOBAL_COMMIT
{
write commit record to local DT;
}
else / *decision GLOBAL_ABORT was received from the coordinator * /
write GLOBAL_ABORT record in local DT;
return;
```

Fig. 5. Algorithm for Optimized 2 Phase Commit Protocol

This study of ACPs from the viewpoint of a single transaction has also hidden the issue of site recovery. When a site recovers, it must complete the ACP for all transactions that might not have committed or aborted before the failure. At what point can the site resume normal transaction processing? After the recovery of a centralized DBS, transactions cannot be processed until Restart has terminated, thereby restoring the committed database state. A similar strategy for the recovery of a site in a distributed DBS is unattractive, in view of the possibility that some transactions are blocked. In this case, the DBS at the recovered site would remain inaccessible until all transactions blocked at that site were committed or aborted.

7. EVALUATION AND RESULTS ANALYSIS

In this section I evaluate proposed atomic commit protocols and then their result analysis is being done.

Let me now examine how O-2PC fares with respect to resiliency, blocking, log write, and time and message complexity.

Resiliency: O-2PC is resilient to both site failures and communication failures, be they network partitions or timeout failures. To see this, observe that my justification for the timeout actions in the previous subsection did not depend on the timeout's cause. The timeout could be due to a site failure, a partition, or merely a false timeout.

Blocking: O-2PC is subject to blocking. A process will become blocked if it times out while in its uncertainty period and can

only communicate with processes that are also uncertain. In fact, O-2PC may block even in the presence of only site failures. To calculate the probability of blocking precisely, we must know the probability of failures.

Time Complexity: In the absence of failures, for deferred consistency constraints:

O-2PC requires three rounds: (1) all the participants send their votes to coordinator (2) the coordinator broadcasts the decision, and (3) all the participants acknowledge to coordinator. For Immediate consistency constraints: O-2PC requires only two round; (1) the coordinator broadcast its decision, and (2) all the participants acknowledge to coordinator.

If failures happen, then the termination protocol may need two additional rounds: one for a participant that timed out to send a DECISION-REQ, and the second for a process that receives that message and is outside its uncertainty period to reply.

Several participants may independently invoke the termination protocol. However, the two rounds of different invocations can overlap, so the combined effect of all invocations of the termination protocol is only two rounds. Thus, in the presence of failures it will take up to four rounds (Deferred consistency constraints) and 3 rounds (Immediate consistency constraints) for all processes that aren't blocked or failed to reach a decision. This is independent of the number of failures! The catch is that some processes may be blocked. By definition, a blocked process may remain blocked for an unbounded period of time. Therefore, to get meaningful results, I exclude blocked processes from consideration in measuring time complexity.

Message Complexity: Let n be the number of participants (so the total number of processes is n + 1). In each round of O-2PC, n messages are sent. Thus, in the absence of failures, the protocol uses 2n messages for deferred consistency constraints and n messages for immediate consistency constraints. All participants that has voted YES invoked the cooperative termination protocol but don't receive COMMIT or ABORT from the coordinator. Let there be m such participants, $\leqslant m \leqslant$ n. Thus m processes will initiate the termination protocol, each sending n DECISION-REQ messages. At most n-m+1 process (the maximum that might not be in their uncertainty period) will respond to the first DECISION-REQ message. As a result of these responses, one more process may move outside its uncertainty period and thus respond to the DECISION-REQ message of another initiator of the termination protocol. Thus, in the worst case, the number of messages sent by the termination protocol (with m initiators) will be:

$$nm + \sum_{i=1}^{m} (n-m+i) = 2nm - m2/2+m/2$$

Elementary calculus shows that this quantity is maximized when n = m, that is, when all participants timeout during their uncertainty period. Thus, the termination protocol contributes up to n(3n + 1)/2 messages, for a total of n(3n + 6)/2 (in case of deferred consistency constraints) and for a total of n(3n + 4)/2 (in case of Immediate consistency constraints) for the entire O-2PC protocol.

Table 2: The Cost Of Different Protocols To Commit A Transaction

Protocol	2PC	3PC	O-2PC		O-3PC	
			Immediate Consistency Constraints	Deferred Consistency Constraints	Immediate Consistency Constraints	Deferred Consistency Constraints
Blocking	High	Very Less	Less	High	Very Less	Very Less
Time Complexity	4	6	2	3	4	5
Message Complexity (Overhead)	4n	6n	2n	3n	4n	5n
Log Write	2+2n	2+2n	2+n	2+2n	2+2n	2+2n

Log Writes: In absence of failure O-2PC requires 2+2n logs writes in case deferred consistency constraints and 2+ n logs writes for immediate consistency constraints.

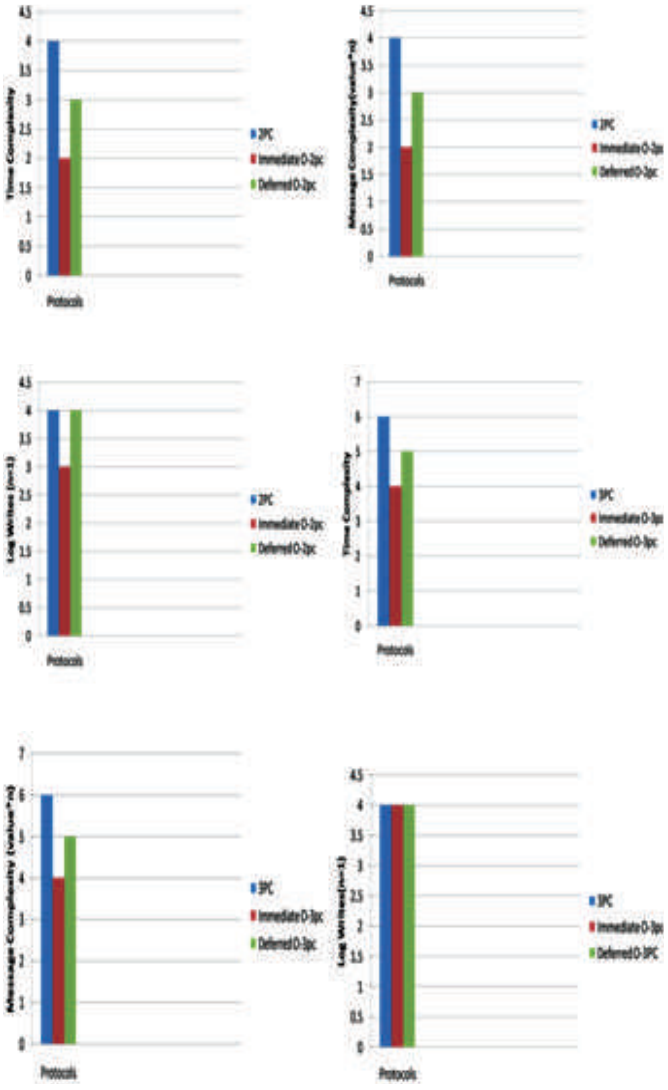

Fig. 5: The cost of different protocols to commit a transaction

Table 3: The Cost Of Different Protocols To Abort A Transaction

Protocol	2PC	3PC	O-2PC		O-3PC	
			Immediate Consistency Constraints	Deferred Consistency Constraints	Immediate Consistency Constraints	Deferred Consistency Constraints
Blocking	High	Very Less	Less	High	Very Less	Very Less
Time Complexity	4	4	2	3	2	3
Message Complexity (Overhead)	4n	4n	2n	3n	2n	3n
Log Write	2+2n	2+2n	2+n	2+2n	2+n	2+2n

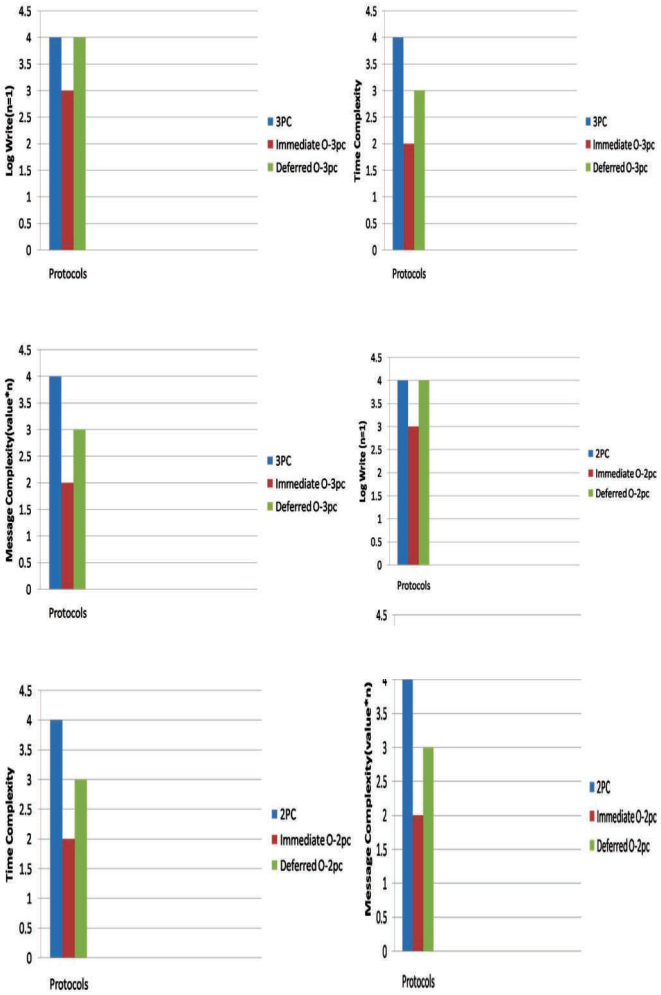

Fig. 6: The cost of different protocols to abort a transaction

Result Analysis: TABLE 2 and TABLE 3 compare the costs of different protocols for the commit and abort case on per transaction basis, respectively. It is clear that O-2PC improved performance but blocking problem is still there; so O-3PC is also compared here.

The comparison is done on four parameters as: Blocking, Time complexity, message complexity and no. of log write. Here n is number. of participants used in distributed transaction. One thing to make clear here is that proposed protocol is not to optimize blocking problem rather our concentration is to optimize overhead as compare to basic 2PC and 3PC protocol. By analysis all the value in above tables and Fig. 5 and Fig 6 it is clears that both O-2PC and O-3PC gives equal performance for most of the parameter.

8. CONCLUSION AND FUTURE WORK

Commit protocols are used to provide reliable methods to enforce ACID properties in database transactions. Current data applications demand much lower execution time and enhanced reliability even in the event of failure and concurrency. There has been a lot of work done on commit protocols and recently there is renewed interest in searching for efficient and reliable commit protocols which can fulfill the needs of present mobile computing and real time computing. Modern Internet database applications such as those commonly found in electronic commerce and electronics services require coordination protocols with reduced overhead. In this paper we critically analyzed two phase commit protocols and three phase commit protocol both on the basis of messages to share, time and cost and also on log force write .This is in order to increase customer satisfaction through enhanced system's throughput. To this end, the proposed protocols can get the performance advantages of 2PC and 3PC in optimized way.

Optimized protocols further enhancement, such as to analyzed performances in failure environment and multilevel transaction execution model. These enhancements and extension are left as part of future work in this direction.

This work can be extended in other possible directions as follows: A more realistic simulation would allow evaluating the performance of studied optimized protocols empirically in order to reveal any hidden costs that cannot be captured analytically.

REFERENCES

[1] Eric C. Cooper," Analysis of Distributed Commit Protocols", Computer Science Division – EECS, University of California ACM O-89791-073-7/82/006/0175, 1982.

[2] Yousef J. Al-Houmaily, "On Interoperating Incompatible Atomic Commit Protocols in Distributed Databases", Department of Computer and Information Programs, Institute of Public Administration, 978-1-4244-0012-6 , 149 - 156 ,14-16 Nov, 2005.

[3] Shishir Kumar, Sonali Barvey," Non-Blocking Commit Protocol", Department of CSE, IJCSNS International Journal of Computer Science and Network 172 Security, VOL.9 No.8, August 2009..

[4] Y. J. Al-Houmaily and P. K. Chrysanthis," 1-2PC: The One-Two Phase Atomic Commit Protocol", Proc. Of the ACM SAC, 684-691, 1-58113-812-1, 2004.

[5] Muhammad Atif, "Analysis and Verification of Two-Phase Commit & Three- Phase Commit Protocols", Emerging Technologies, 2009. ICET 2009. International Conference, Department of Mathematics and Computer Science, Technische Universities Eindhoven, 326 – 331, 978-1-4244-5632-1/09, 19-20 Oct. 2009.

[6] Y. J. Al-Houmaily and P. K. Chrysanthis, "Atomicity, with Incompatible Presumptions", Proc. of the 18th, ACM PODS, pp. 306-315, 1-58113-062-7, 1999.

[7] V. Manikandan1, R.Ravichandran1, R.Suresh1, F. Sagayaraj Francis," An Efficient Non Blocking Two Phase Commit Protocol for Distributed Transactions", Vol.2, Issue.3, pp-788-791 ISSN: 2249-6645. 2012.

[8] M. Abdullah, R. Guerraoui and P. Pucheral, "One-Phase Commit: Does it make sense? ", Proc. of the Int'l Conf. on Parallel and Distributed Systems, ISBN: 0-8186-8603-0, ISSN: 1521-9097, 182 – 192, 14-16 Dec 1998.

[9] Inseon Lee, Heon Y. Yeom, "A Single Phase Distributed Commit Protocol for Main Memory Database Systems", Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS.02), 0-7695-1573-8, 15-19 April 2001

[10] Poonam Singh, Parul Yadav, Amal Shukla and Sanchit Lohia, "An Extended Three Phase Commit Protocol for Concurrency Control in Distributed Systems", International Journal of Computer Applications (0975 – 8887) Volume 21– No.10, May 2011

[11] P.Krishna Reddy and Masaru Kitsuregawa, "Reducing the blocking in two-phase commit protocol employing backup sites", Cooperative Information Systems, 1998. Proceedings. 3rd IFCIS International Conference, Inst. of Ind. Sci., Tokyo Univ., Japan, 0-8186-8380-5 , 406 - 415 ,20-22 Aug. 1998.

[12] Constantinos V. Papadopoulos, "On the Heterogeneity of Distributed Databases Integrating Commit Protocols", Dept. of Comput. Sci., Piraeus Univ., Greece, 0-8186-5840-1, 380 – 386, 21-24 Jun 1994.

[13] Sylvia, Vibha, R. B. Patel IEEE Member, "Queue Sensing Distributed Real-time Commit Protocol: A New Dimension for Distributed Database System", 2009 International Conference on Advances in Recent Technologies in Communication and Computing, 978-1-4244-5104-3, 829 - 834, 27-28 Oct. 2009.

[14] Udai Shanker, Nikhil Agarwal, Praphull Goel, Shalabh K. Tiwari, Praveen Srivastava,"Real Time Commit Protocol-ACTIVE", 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 978-1-4244-8538-3 , 314 - 319 ,4-6 Nov. 2010.

[15] Ivana Popovic, Vladislav Vrtunski, "Formal Verification of Distributed Transaction Management in a SOA Based Control System", 2011 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, 206 - 215, 978-1-4577-0065-1, 27-29 April 2011.

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

[17] Alexander Thomasian, "Distributed Optimistic Concurrency Control Methods for High-Performance Transaction Processing", IEEE Transactions on Knowledge and Data Engineering, Vol.10, No. 1, ISSN: 1041-4347, 173 – 189, 06 August 2002.

[18] Khake Asha, Gojamgunde Ashwini, Shastri Ashlesha and Biradar Usha, "Transaction Management in Distributed Database" BIOINFO Transactions on Database Systems, Volume 1, Issue 1, pp-01-06, 2011.

[19] D. Skeen and M. Stonebraker, "A formal model of crash recovery in a distributed system," IEEE Trans. Software Eng., vol. 9, no. 3, pp. 219–228, 1983.

◆ ◆ ◆