CHAPTER 5

COMMIT PROCESSING IN THE DISTRIBUTED DATA
FLOW GRAPH ENVIRONMENT

5.1. Introduction

In the preceding chapters (three and four), we have proposed two algorithms for coordinating concurrent transactions in distributed environment. The algorithms assume incorporate the database updates that involve multiple sites. In this chapter, we have examined the problem of incorporating updates at multiple sites, and the use of commitment protocols in a brief manner. We have also investigated the possibility of integration of protocols, with the proposed techniques of concurrency control and the algorithms.

The study is divided into the following parts. In the next section, we describe the problem of reflecting the updates at multiple sites. After this, we discuss the reliability issues in a distributed database system. In section 5.4, the two-phase commit protocol is presented. In section 5.5, we examine the recovery aspects of the two-phase commit protocol from failures. A description of a three-phase (non-blocking) commit protocol is presented in section 5.6. In section 5.7, we explore the issue, as to how the commit protocols can be integrated with the proposed algorithms. The last section considers the recovery and partition handling in case of distributed and replicated databases.
5.2 Reflecting the updates at multiple sites

In this section, we study the problem of reflecting the updates at multiple data sites in a distributed environment.

In a distributed system there is a possibility of partial failures, where some sites may be working while others may fail. Also, failures in a distributed system include additional types of failures, such as the lost message cases, site failure, and communication link failure. Some of these failures may partition the network into two or more isolated parts.

In this environment, the transaction is treated as an atomic unit of computation. Which means, either all or none of its writes is performed. In transaction processing, one non-trivial problem is that of consistent termination. The commit or abort operation of a distributed transaction must be processed, at all sites where the transaction accessed data items. Ensuring this is complicated by the probability of partial failures.

The main technique for committing transactions in the presence of failures is based on the use of logs. A "log" contains information for undoing or redoing all actions which are performed by the transactions. To "undo" the actions of a transaction means to reconstruct the database as it was, prior to its execution. To "redo", means to perform its actions again [CER85].

An atomic commitment protocol is an algorithm that ensures consistent reflection of updates at multiple sites. In this chapter, we will give an exposition of the issues related to transaction commitment in the distributed environment. Before doing so, it is necessary to examine the nature of failures in detail.
5.3. Failures in DDBSs

The types of failures that often occur in the distributed database system have been discussed below.

5.3.1. Transaction failures

Transactions can fail for a number of reasons; for example it could be due to an error caused by incorrect input-data, or due to detection of a deadlock. In addition to these, some concurrency control algorithms do not permit a transaction to proceed or wait (as in static locking) if the data items for which they attempt to access are currently being accessed by another transaction. The usual action taken in case of transaction failure is to "abort" the transaction, thus resetting the database to its earlier state.

5.3.2. Site (system) failures

In our model of failure, a site is either working correctly (operational), or not working at all (down). It never performs incorrect actions. This type of behavior is called fail-stop [SCH82], because sites fail by stopping.

When a site experiences a system failure, processing stops abruptly and the contents of volatile storage are destroyed. When a site recovers from a failure, it executes a recovery procedure, which brings the site to a consistent state. After this, it resumes normal processing.

As each site is either functioning properly or has failed, different sites may be in different states. A partial failure is a situation where some sites are operational while others
are down. Partial failures are tricky to deal with. Fundamentally, this is because operational sites may not be able to ascertain about the state of the failed ones. The most important goal of atomic commitment protocols is to minimize the effect of one site's failure on other sites.

5.3.3 Communication failures

Communication failures are of many types. The most common ones are the errors in messages, improperly ordered messages, lost (or undeliverable) messages, and line failures. The first two types of errors are the responsibility of the computer network. The underlying computer network hardware and software ensure that, two messages sent from a process at site $S_i$ to another process at site $S_j$ are delivered without any error, and in the same order in which these were sent.

Lost or undeliverable messages are the consequence of communication line failures or (destination) site failures. If a communication line fails, in addition to lose of the message(s) in transit, it may also divide the network into two or more disjoint groups. If the network is partitioned, the sites in each partition may continue to operate. In this case, maintaining the mutual consistency of the database is a significant problem, especially if the database is replicated across the partitions.

If the messages can not be delivered, we will assume that the network does nothing about it. It will not buffer it for delivery to the destination site when the service is re-established, and will not inform the sender process that the message can not be delivered. In short, the message will simply be lost. So, the responsibility of detecting that, a message
is undeliverable is left to the application program. The detection can be facilitated by use of timers and time-out mechanism. This time-out interval needs to be set to a value greater than that of the maximum round-trip propagation delay of the message in the network.

5.4. The two-phase commit protocol

In the basic two-phase commitment protocol, the control coordinator has a special role to play. All the other agents which must commit together are called participants. The coordinator is responsible for taking the final "commit" or "abort" decision. Each participant corresponds to a sub-transaction which has performed some write action. It is responsible for performing the write action at its local database. These participants are nothing but the sites where the updates are to be reflected. The coordinator is the site which wants to commit the transaction. Normally, the coordinator is the home site of the transaction.

The basic idea of two-phase commit protocol is to determine a decision for all participants with respect to committing or aborting all the sub-transactions. If a participant is unable to locally commit its sub-transaction, then all participants must abort. This protocol consists of two phases. The goal of first phase is to reach a common decision and second phase is to implement this decision.

We first present the protocol in absence of failures, and then we discuss the recovery from all possible types of failures. The protocol proceeds as follows (see figure 5.1).
Coordinator:
1. Write the "prepare" record in the log;
2. Send the PREPARE message; activate time-out;

Participant:
1. Wait for PREPARE message;
2. If the participant is READY then
   begin
   Write sub-transactions records in the log;
   Write ready "record" in the log;
   Send the READY answer message to coordinator;
   end
   else
   begin
   Write the "abort" record in the log;
   Send the ABORT message to coordinator;
   end;

Coordinator:
1. Wait for message (READY or ABORT) from all participants or expire the time-out;
2. If time-out expires or the message is ABORT then
   begin
   Write "global-abort" record in the log;
   Send the ABORT command message to participants;
   end;
   else
   begin
   Write "global-commit" record in the log;
   Send COMMIT command message to all participants;
   end;

Participant:
1. Wait for command message;
2. Write the abort or commit record in the log;
3. Send the acknowledgment message to coordinator;
4. Execute the command;

Coordinator:
1. Wait for acknowledgment messages from all participants;
2. Write the complete record in the log;

Figure 5.1. The two-phase commit protocol
**Phase one:** The first phase starts when the coordinator asks all the participants to prepare for commitment. Each participant answers READY, if it is ready to commit and willing to do so. Before sending the first PREPARE for commitment message, the coordinator records on the stable storage, a log record called a "prepare" log record, in which identifiers of all sub-transactions are recorded. The coordinator also activates a time-out mechanism, which will interrupt the coordinator after the given time interval.

When a participant answers READY, it ensures that it will be able to commit the sub-transaction even if failures occur at its site. To do this, each participant has to record to its stable storage the following information.

1. All the information required for locally committing the sub-transaction must be recorded on a stable storage; and
2. the fact, that the sub-transaction has declared itself to be ready to commit (a ready "log" record), must be recorded on stable storage.

The coordinator decides to "commit" or "abort" the transaction as a result of the responses received from the participants. If all the participants sent READY message, it decides to commit the transaction. If some participant sent ABORT or there is a time-out, it decides to abort the transaction.
The state transition diagram of protocol is given in figure 5.2. The states are dentated by circles, and the edges represent the state transitions. The terminal states are denoted by concentric circles. The interpretation of the labels on the edges is as follows: The reason for the state transition, which is a received message, is given at the top, and the message that is sent as a result of state transition is given at the bottom.

(a) Coordinator

(b) Participants

Figure 5.2. State transitions in two-phase commit protocol
**Phase two:** The coordinator begins the second phase of the two-phase commit by recording its decision by writing a "global-commit" or "global_abort" record in the log. Which means that the distributed transaction will eventually be committed or aborted, despite failures. Further, the coordinator informs all participants of its decision, by sending the message.

In response, all the participants write a "commit" or "abort" record in the log. From this very moment, the local recovery procedure is capable of ensuring that the effect of the sub-transaction would not be lost.

All participants send a final acknowledgment message to the coordinator, and perform the actions required for committing or aborting the sub-transaction. On receiving the acknowledgment messages from all participants the coordinator writes a log record called "complete" record.

The two-phase commitment protocol is resilient to all failures in which no log information is lost.

5.5. **Recovery with two-phase commit**

The recovery log, in addition to information related to centralized recovery, also includes the log of messages transmitted between sites. Such a record enables the recovery system to detect the connectivity of a site within the network, or the extent of the site's interaction with the rest of the system. The recovery system also takes the note of the status of the sub-transactions running at the site. It determines which sub-transactions were committed, aborted, or blocked.

Communication link failures, in many cases, can result in the database system
becoming partitioned. Each of the partitioned systems could operate by marking the sites in the other partitions as down.

5.5.1. Site recovery

When a failed site resumes operation, it consults first the recovery log to find the transactions that were active during the interval. For local transactions, recovery actions are similar to the centralized database system which requires a simple "undo" or "redo" of the transactions. Global transactions are classified into two types: coordinator or participant.

Regarding all participant type transactions, if the log indicates that it has not sent the status message to the coordinator, then the later would abort all sub-transactions. Suppose the log for participant type transaction indicates that a READY message has been sent to the coordinator, from the recovery action, the site needs to inquire from the coordinator or some other participant about the outcome of the transaction. Once it receives the message, the recovery action performs a "redo" or "undo" operation. In the case of a participant for which the log indicates the receipt of a second message from the coordinator (to commit or abort), the recovery process can take appropriate action and ensure that an acknowledge message be sent to the coordinator.

For the second type of coordinating transaction, the recovery process examines the log to determine its status. If no PREPARE messages had been sent before the site failure, all participants would have aborted, whereupon the coordinating transaction can be aborted as well. If the coordinator sent PREPARE messages before crash, the recovery process must send new PREPARE messages. Participant sites that receive this signal would treat this as a
repeat message, and ensure that appropriate actions are taken, and send the required acknowledge message.

If the site failed after the coordinator wrote a complete transaction record in the log, no further actions would be necessary.

5.5.2. Lost message

The type of recovery operation to be performed depends on the message that gets lost. If the PREPARE message is lost, the participant would abort, which will lead to the abortion of the global transaction. If the status message from any one of the participants is lost, the coordinator would time-out and abort the global transaction, including all participant transactions. If the second (ABORT or COMMIT) message is lost, a participant would time-out and recover by consulting the coordinator. In the event that one of the ACKNOWLEDGE messages is lost, after expiry of time-out period the coordinator sends the request to retransmit ACKNOWLEDGE message to the participant.

5.5.3. Communication link failure

Suppose the failure of the communication link occurs in such a way that a subset of the participant sites are partitioned and are without a coordinator, as far as the coordinator is concerned, this is equivalent to the failure of a number of participant sites. If the failure occurs before the partitioned participants were sent the PREPARE message, the coordinator would have aborted the global transaction, including all non-partitioned sub-transactions. The partitioned participants would also abort after a time-out. If the failure occurs after the
participants have reported their status, the coordinator would have decided either to commit or abort. The partitioned sites recover by consulting coordinator.

5.6. The three-phase commit protocol

A problem with the basic two-phase commit protocol is that a sub-transaction which has entered its ready state could be blocked. The reason for blocking is a failure of the coordinator or of the communication network. The blocked sub-transactions must keep all their resources blocked, until they receive the final command during the recovery from the failure, because they must be able of eventually commit or abort. Therefore, blocking reduces the availability of the system in case of failures. This problem can be eliminated by modifying the two-phase commit protocol as the three-phase commit protocol [SKE81]. In this section, we briefly explain the protocol.

In this protocol, the participants do not directly commit the transaction during the second phase of commitment. Here, they reach a new "pre-commit" state. An additional third phase is required for actually committing the transaction. In this case, the command which is issued by the coordinator during the second phase is either the normal ABORT command or a new PREPARE-TO-COMMIT command. Once PREPARED-TO-COMMIT messages are sent, the coordinator enters a new entry before the commitment state. Each participant must send an YES message when it has executed this command. It is stored on stable storage. Finally, when the coordinator receives all YES messages, it enters the final commit state and sends the final commit command. This new protocol eliminates the blocking problem of the two-phase commitment protocol as shown below. The state transition diagram is shown in

115
1. If one of the operational participants receives the command and the command was ABORT, then the operational participants can abort the transaction. This is similar to the two-phase commitment.

2. If one of the operational participants receives the command PREPARED-TO-COMMIT, then all operational participants can commit the transaction. The failed participant would commit the transaction on its restart. The failed participant does not unilaterally abort the transaction, because it has sent a READY message. This case was correctly dealt with by two-phase commit.

3. If none of the operational participants has received the PREPARED-TO-COMMIT command, we have the case which can not be terminated for a two-phase commitment protocol. With this protocol, instead, the operational participants can abort the transaction, because the failed participant has not yet committed.

The new protocol allows termination for all possible failures during the second phase, which blocked the two-phase protocol. If a coordinator fails during third additional phase which has been added, the termination protocol will elect a new coordinator and commit the transaction, because all participants are in the "pre-commit" state when this phase begins.
The new protocol requires three phases for committing a transaction and two phases for aborting it. Several termination protocols for operational participants and restart protocols for failed participants are designed to operate correctly with the three-phase-commitment protocol [OZS91, CER85].
5.7. Integration with the proposed approaches

The two-phase and three-phase commitment protocols terminate consistently when a coordinator or a participant fails. By examining the concurrency control algorithms proposed in this study, it appears that, some of the stages of proposed algorithms can be merged with the commitment protocols. The primary advantage of the data flow graph based algorithms is that the participating sites can know the identity of the transaction which is about to be committed, by virtue of the data flow graphs constructed at sites. That is, once the local access graph, or transaction data flow graph is formed, the transaction knows about the preceding transactions.

In the local access graph based approach, when a transaction requests the data items, its local access graphs are constructed at respective sites. The identity of the transaction is present within the system until COMMIT or ABORT is sent by the home site (coordinator). So, sending of the lock requests to data resident sites is tantamount to the sending the of "PREPARE" messages to participating sites (in the two-phase commit). In return to this, (prepare messages), in the proposed approach, the local access graphs or access grants are sent to coordinator.

In the two-phase commit protocol, after sending the PREPARE messages, the coordinator sets the time-out period for receiving the READY or ABORT messages by participants. In this approach, instead of READY or ABORT messages, the coordinator receives the access grants. At the sites where the lock request contains preceding transactions in its LAG, the access grant may be delayed, for it gets an access to the concerned data items
after receiving the final commit of the all preceding transactions in its LAG. If the two-phase commit is followed, then after the expiry of time-out period, the coordinator sends an abort messages to all sites. To integrate with commit protocol, it is proposed that, in response to lock requests, the participating sites send either access grants or local access graphs. Receiving the local access graphs should not be taken as a READY message. It only helps the coordinator to extend the time-out period to have access grants. So, by examining these graphs, the coordinator sets the timer for remaining access grants from the sites where the local access graphs are constructed. Finally, participants send the data access grants to the coordinator. After executing the transaction the coordinator sends the final commit values to the participating sites. The step-wise description is given below.

Coordinator:

1. The coordinator sends its lock requests (PREPARE messages) to participants. It sets the timer and also writes the "prepare" record in the log.

2. The coordinator receives the local access graphs or access grants (READY) from all the participants. The coordinator sets the timer, according to number of preceding transactions in which it gets accesses to all data items requested at participants sites. Otherwise, it sends the ABORT to all participants.

3. The coordinator gets the all lock grants (READY). It sends the COMMIT to all participants. Otherwise (NO) it sends the ABORT to all participants.
Participant:

1. The participant receives the lock request for the data items. It constructs the local access graph. In case there were no preceding transactions in the LAG then the access grants (YES message) are sent to the coordinator. Otherwise local access graph is sent to the coordinator.

2. After getting access to data items, the access grants (YES message) are sent to the coordinator.

3. When it receives the COMMIT or ABORT, it is recorded on a stable storage. It sends the acknowledgment to the coordinator.

Here, a change can be noticed in the first phase. When a coordinator sends the lock requests to all participants, the participants may in turn send local access graph or the access grants. The access grants are viewed as YES messages sent by participants. When local access graph is received by the coordinator, it is not treated as an YES message. The local access graph would give an estimate of additional time-out period, in which, the coordinator extends the duration of time-out period by setting the timer to have access grant from the participant. The other stages remain same as in the two-phase commitment. In the above protocol, each action is recorded on a stable storage, as it is with the basic two-phase commit protocol, which will be useful at the time of recovery.

For the transaction data flow graph based approach for replicated databases, the commit protocol can be totally integrated. As there is no risk of rejection due to conflict, the coordinator can send its transaction request to all participant sites as READY messages. The
participants send respective local graphs, to the coordinator. If the coordinator receives the local graphs from majority sites, then the TDFG can be constructed at the coordinator site and wait for preceding transactions' COMMIT or ABORT message. Following this, the transaction is executed and the update values can be sent as final commit messages. In case the coordinator does not receive the local graphs from majority of sites, an abort message has to be sent to all sites.

5.8 Network partitioning and commitment protocols

The termination protocols for network partitioning address the termination of the transactions that were active in each partition at the time of partitioning. This would imply that the sites in each partition can continue executing transactions despite the partitioning.

Unfortunately, in general it is not possible to design a non-blocking termination protocol, in which the transactions continue to have access in different partition groups, in the event of network partitions. In other terms, if the network partitioning occurs, normal processing is not possible in all partitions [SKE83]. This, limits the availability of the system. Hence the alternative strategies have to be developed. It is convenient to allow the termination of the transactions by at least one group of the sites, possibly at largest group, so that the blocking may be minimized.

5.8.1 Network partitioning in distributed databases

In the presence of network partitioning of distributed databases, the transactions which were active at the time of partitioning should be allowed to terminate. Any new transaction
that accesses a data item which is stored in another partition is simply blocked and has to wait for repair of the network partitioning. Concurrent accesses within the partition can be handled by the concurrency control algorithm. The significant problem is therefore terminate the transactions, that are active at the time of partition.

There are two approaches to this problem, the centralized approaches and voting based approaches.

**Centralized approaches:** In the case of primary site concurrency control algorithms, a site is designated as a primary site, and the group which contains it is allowed to terminate the transaction. This approach is vulnerable to primary site failures.

**Voting based approach:** As discussed earlier, voting [THO79] is a technique to manage a concurrent access to replicated data. The idea of majority voting has been generalized to voting with quorums. In the case of distributed databases, this involves the integration of the voting principle with commit protocols [SKE82]. Every site in the system is assigned a vote $V_i$. Let us assume that the total number of votes in the system is $V$, and the abort and commit quorums are $V_a$ and $V_c$, respectively. The following rules must be followed in the implementation of the commit protocol (see figure 5.4):

1. $V_a + V_c > V$, where $0 \leq V_a, V_c \leq V$.
2. Before a transaction commits, it must obtain a commit quorum $V_c$.
3. Before a transaction aborts, it must obtain an abort quorum $V_a$. 

122
The first rule ensures that a transaction can not be committed and aborted at the same time. The next two rules indicate the votes that a transaction has to obtain before it can terminate.

(a) Coordinator

(b) Participants

Figure 5.4 State transitions in the quorum three-phase commit protocol
The integration of these rules into the three-phase commit protocol necessitates a minor modification of the third phase. Therefore, a quorum-based commitment protocol can be obtained from the three-phase commit protocol simply by including the quorum requirement (rule 2) to the third phase. For the coordinator to move from the PRE-COMMIT state to COMMIT state, and to send the global-commit command, it is necessary for it to have obtained a commit quorum from the participants. This would satisfy rule (2). This protocol obeys rule (3), without having to check explicitly for the existence of an abort quorum $V_a$, because all sites which are in the ready state agree to possibly abort the transaction. They therefore participate implicitly in building an abort quorum.

When a network partitioning occurs, the sites in each partition elect a new coordinator similar to the three-phase commit protocol in case of site failures.

5.9.2 Network partitioning in replicated databases

Network partitioning in the case of replicated databases is addressed by concurrency control algorithm. In the case of replicated databases, the replica control protocol has to be involved in mapping a read or a write on a logical data item to a read or a write on the physical data item copies. In the presence of network partitioning, the copies may be in different partitions and the replica control protocol has to be concerned with the management of network partitioning.

In the case of network partitioning, either by permitting all the partitions to continue
their normal operations, the database consistency may be compromised, or by guaranteeing the consistency of the database by employing suitable strategies that would permit operation in one of the partitions while sites in the other remain blocked. The strategies for dealing partition problems are classified as pessimistic or optimistic [DAV85]. Pessimistic strategies emphasize the consistency of the database, and would not permit transactions to execute in a partition if there is no guarantee that the consistency of the database can be maintained. On the other hand, optimistic approaches emphasize on the availability of the database even if this would cause inconsistencies.

Several partition processing strategies have been suggested that either relax correctness, or rely on compensating or correcting transactions to regain consistency once the partition is repaired [DAV84, GAR83, KOG87, LYN86, PAR83]. Other partition processing strategies have been suggested that pre-analyze transactions or use type-specific information to increase availability while guaranteeing correctness [BLA83, BHA90, HER86]. Since most of the optimistic approaches require extensive knowledge about what the information in the database represents, how applications manipulate the information, and how much undoing, correcting, or compensating inconsistencies will cost, we discuss the partition processing strategies that guarantee correctness. Here, we consider pessimistic approaches that guarantee correctness.

Most of the pessimistic replica control protocols are based on some version of voting. Besides initial algorithm by [THO79], an early suggestion to use quorum based voting for replica control is due to [GIF79].

In the case of network partitioning the proposed algorithm in chapter four, deals with
the partition failures in the same way as the quorum based protocols. In this, the partition which has the majority of sites, can process transactions. In this algorithm, the sites can be assigned different weights. With minor modifications of the algorithm, it is possible to design a partition handling algorithm which does not reject transactions in case of conflicts. Our aim has been to propose a concurrency control algorithm based on data flow graphs. In this light, we have examined and suggested to include the modification part, to suit the availability levels of voting based approaches in case of site and partition failures.

5.9. Conclusions

The primary goal of this chapter is to discuss commit processing in the data flow graph environment. The problem of reflecting the updates at multiple sites, need of commitment protocols and different kinds of failures have been discussed. In the following sections, two-phase commit protocol and the way it handles the transactions, in case of different types of failures is also explained. The integration of local access graph based approach with two-phase commit protocol is proposed. The three-phase commit protocol, and the way it eliminates the blocking problem of two-phase commit protocol is briefly described in the following section. The last section describes the partitioning problem, and also the way this problem is handled in the case of distributed, and replicated database system, respectively.

On the whole, the proposed algorithms need minor modification, if integrated with commit protocols. In the LAG based approach, in place of READY messages, the participants send, either access grants or LAGs. In case the coordinator receives the LAGs, it has to wait
for some more time to have the access grants. Our assumption is, that by knowing the preceding transactions the coordinator can decide the approximate extension period, for access grants. The formalization of this concept needs an additional investigation.

In case of replicated databases, we believe that, the weighted voting can be employed, with a little modification.

After presenting the issues connected with incorporating updates at multiple sites, the summary of the study has been presented along with directions for further research in the following chapter.