Chapter 3: Modelling and Analysis of Distributed Data Repositories

3.1 Objective

Enterprises are physically distributed over different locations like continents, countries, plants, divisions, laboratories, departments, work-groups and so on. Logically an enterprise can function as one unit of work. In order to support such logical unity among the isolated objects, a generic framework of distributed computing is highly needed. As there is no true global clock to synchronize such transactions in a loosely coupled distributed environment, one needs to device a cost effective deployable generic model for asynchronous distributed transmissions. As for example, in a business to business communication in e-commerce an ACH is a third party that is always present to record all financial transactions along with their time of incidence for effective control and audit purposes. Thus designing of a distributed communication protocol for transaction processing is considered to be one of the most critical multi faced task. The processes in a 2PC demands sequence of updates on the stable storages at different locations as a single complete unit of work. 2PC is a consensus protocol. It is a simple-sounding problem and amazingly been at the core of a theoretical distributed systems research for over last couple of decades. Consensus is solvable in synchronous systems, but not in case of asynchronous systems, when there is a possibility of failures, which is again time dependent. Indeed, most of the distributed systems are fabricated upon consensus. This chapter revisits the 2PC protocol of transactional technology for ensuring the consistent commitment of distributed transactions and then a framework of a high level net model is suggested to analyze transaction processing in a distributed environment.

3.2 Distributed System

Here, we describe the transaction processing issues of concurrent distributed data. In a distributed repository, user envisages dispersed records as a single unit on global schema, without having any federal control. In a centralised environment, a user accesses the database from remote locations via connected workstation. Here, data access and processing takes place at the central site. Figure 3.1 shows a centralised DB System with five distinct locations T1 to T5.

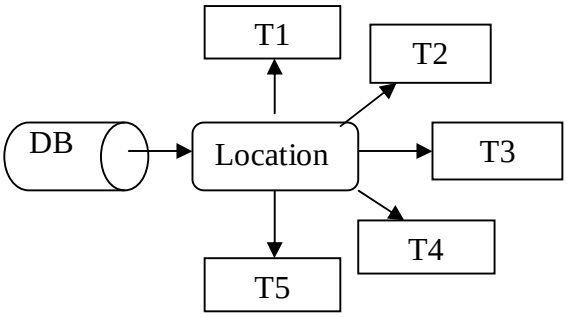

Figure 3.1 Centralized database System

In a distributed environment physical data is stored in different repositories across varied locations connected over some form of data communication network like LAN, WAN, MANET, etc. Besides, the devices and their operating environments may be of various types.

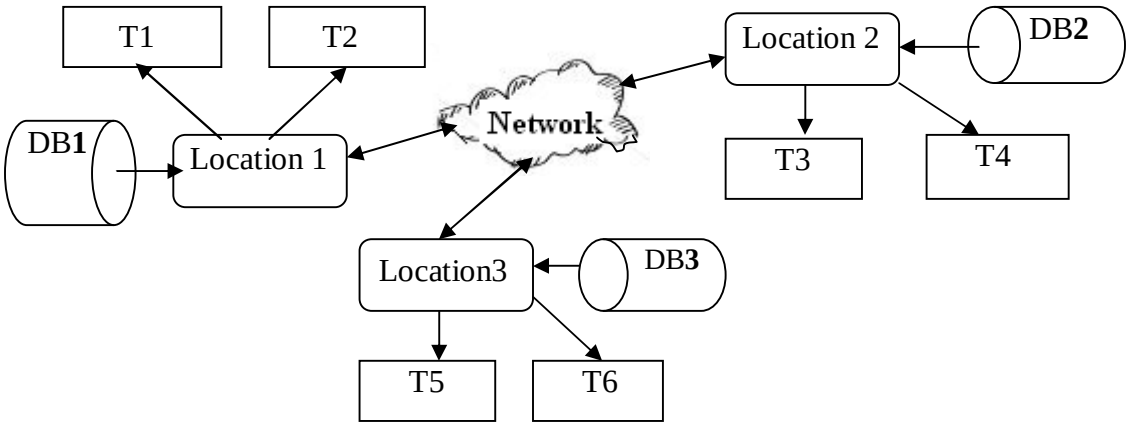

Figure 3.2: Example of a Distributed System

The applications or its fragments running from any location or ruining concurrently from many locations should be transparent before the user as a single source of information. Figure 3.2 is a distributed database (DDB) setup logically organised over 3 locations of a computer network with 3 different databases. Each site has a degree of autonomy, is capable of executing a local application, and also participates in the execution of a global application. The figure 3.1 can be extended in a top-down manner into a distributed setup [6] or figure 3.2 by federating existing database management through a bottom-up approach, which is also known as multi database systems [52]. There could be various degrees of autonomy from tightly-coupled sites to complete site independence. It is evolved to serve medium to large size organisations by interconnecting

existing databases, replicating databases, increasing reliability and adding new databases as new organisational units.

Transaction is a unit of work use to access the DDBS and suffers from diverse problems during transaction management and access control. It demands different rules to monitor, retrieve, update, and replicate into databases. Concurrency control is the activity of coordinating concurrent accesses into a multi-user multi site distributed database management system (DDBMS). The major obscurity in accomplishing this goal is to prevent database updates performed by one user from interfering retrievals and updates performed by another from the same or different sites. The problems can be listed as:

i. The lost update problem.
ii. The temporary update or dirty read problem
iii. The incorrect summary problem

Lost update: Occurs when a transaction say (B) updates the same data being modified by another transaction say (A) in such a way that (B) reads the value of (A) at time 2 prior to the write operation of A at time 3 and the updated value of A is lost.

Table: 3.1 Example on Lost Update Problem

Time	Transaction(A)	Transaction(B)
1	R(A)	
2	A=A+100	R(A)
3	W(A)	A=A-200
4		W(A)

This example in table 3.1 shows that the write operation at time 3 is overwritten by the write operation at time 4. The transactions are from the same or from the different sites or locations and the lost update problem returns an erroneous value of A.

Temporary Update (Dirty Read) Problem: This problem occurs when a transaction (A) aborts at some point of operation and by then the transaction (B) that had read or modified the data used by A. Here the example in table 3.2 shows that the transaction (A) fails at time unit 5, and (A) finally retains the initial value of (A). Mean while the transaction (B) reads the changed value of (A) at time unit 4 and performs the subsequent operations on the temporary value of A.

Table: 3.2 Example on Dirty Read Problem

Time	Transaction(A)	Transaction(B)
1	R(A)	
2	A=A+100	
3	W(A)	
4		R(A)
5	Rollback	A=A-200
6		W(A)

The transactions are from the same or from the different sites or locations and the temporary or dirty read problem return a wrong value of A.

The Incorrect Summary Problem: Transaction **(**B) read the value of A and 100 is added to it at time 2. Then at time 3, transaction (A) read the value of A and 200 is subtracted from it and write operation is performed, the value of 'A' gets changed at time 5. Effectively the operation of transaction (B) performed at time 2 is lost and subsequent Read and write operations at time unit 6 to 8 are performed in sequence. Finally, the read operation of A at time unit 9 shows the discrepancies of 100 units in the value of A as the result.

Table 3.3 Example on incorrect Summary Problem\

Time	Transaction(A)	Transaction(B)
1		R(A)
2		A=A+100
3	R(A)	
4	A=A-200	
5	W(A)	
6		R(A)
7		A=A+400
8		W(A)
9	R(A)	

Thus from the above examples, it is predominant that some form of concurrency control mechanism should be introduced to avoid such types of anomalies during interleaved transaction processing. It can be achieved by introducing a scheduler.

The goal of the transaction log is to ensure serializability, which enforces safe concurrent transaction operations by maintaining a history of transactions. The log is said to be serial if every transaction of one schedule execute before any of the operations from the other schedule. As for example the time sequences of the Transaction (A) and Transaction (B) of any of the

above three examples are schedules. If the log produces the same output with same effect on the database in case of serial execution of the same transactions then the set of transactions is called serializable. Existence of a serial log at some stage of concurrent processing ensures NP completeness and there is no known algorithm available to access in polynomial time whether any given log is serializable [53]. While serializability problem is NP complete, several subclasses of serializable logs are available with polynomial time like, class S, 2PL, BTO, CPSR, and SGT [54].

3.3 Distributed Transaction Processing

Among the concurrency control protocols 2PL is the most popular scheduling technique in a centralized DBMS. In case of a DDBMS, 2PL aggravate huge cost of communication due to deadlock problems. Let us visualize the DDBMS architecture during transaction processing. Figure 3.3 consists of a set of locations connected through a communication network, where each location is a DBS and each location of a DDBS runs a TM, a DM, and a scheduler module. The schedulers are distributed and at each site there exists a local scheduler to assist the consistency issues of the database.

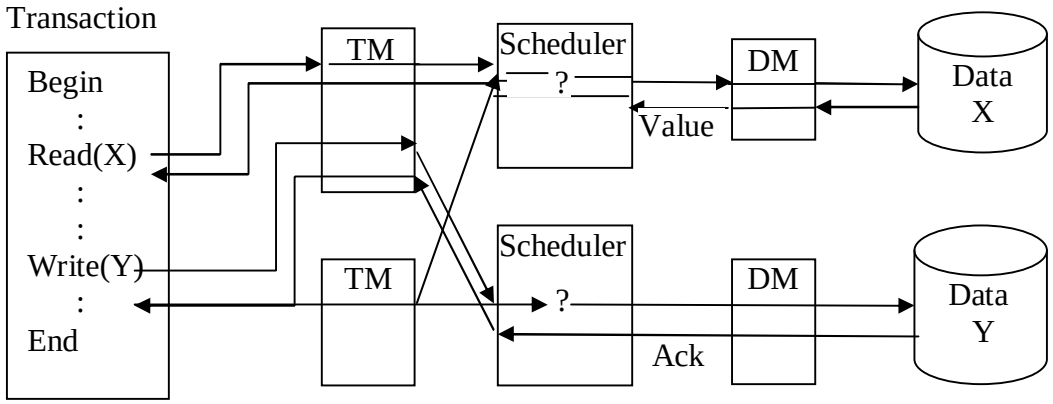

Figure.3.3 Distributed Transaction Execution Architecture

The distributed scheduler acts as a global scheduler in the system. TM performs the pre-processing operations. It receives transactions from the user and passes it to the scheduler to control the relative order in which these operations are supposed to be executed. DM manages the actual database. Therefore some transaction executes from more than one location must commit at all locations and it is not acceptable to have a transaction committed at one site and

aborted at another. Thus a single distributed transaction primarily demands two phases. In the first phase participants take the necessary steps for committing or aborting the transaction and in the second phase the coordinator decides whether to commit or abort the transaction. It notifies the result to all the cohorts to ensure atomicity across the locations. There may be resiliencies during commit or abort, so it is needed to include all such processes and model the 2PC protocol for distributed applications and analysis.

3.4 Modeling Distributed Systems Using Petri Net

Modelling of distributed systems at a behavioural level is critical and even more complex. Start with the 2PC protocol for distributed transactions are modeled with the help of a timed PN to analyze the ACID properties for consistent commitment of distributed transactions. In chapter 2 article 2.4 PN and its recent derivatives are already discussed. However, timing constraints plays a vital role in modeling and analyzing asynchronous concurrent real-time applications. There could be various temporal constraints in modeling such systems. The constraints could be a fixed or a variable type. When the transition is a fixed type constraint with a single time delay is known as Timed Petri Net [18] and it observes a strong firing mode. Instead of a single time delay, when a time domain is used and observes a strong firing mode is named as Time Petri Net. The additional delay time in temporal aspects of concurrent and distributed applications will restricts the dynamic behavior of the net. But by adding color to the tokens will help in identifying the objects uniquely. If a hierarchy is added then decomposition of a complex system becomes simpler. When all or some of the features like time, color, and hierarchy are added to a basic PN model the PN Model is known as High Level PN [56].

Timed workflow graphs are used for modeling the temporal aspects in a WFMS, where first order predicate logic is used for modeling, specifying and analyzing the temporal issues at design and execution times [57]. In a TPN model time is associated to transitions. Transitions represent activities which takes time. Timed Petri nets are similar to PN with the addition of a clock structure associated with each timed transition. Let us now define the time Petri net as a five tuple:

$TPN = (P, T, D^{-}, D^{+}, V)$…….. …….(2)

A timed transition t_j once it becomes enabled fires after a delay v_{jk}. Time delays are of deterministic, non deterministic and stochastic types. Deterministic delays allow for simple analysis methods with limited applicability. The models handling non-deterministic delays use time intervals to specify the duration of the delay. In these models each delay is described by a probability distribution function. Formally the Timed PN's allow strong firing mode, i.e., a transition, t_j, with a delayed time, T_{del}, will immediately fire at time when necessary tokens have arrived. During the time period from T_0 to ($T_0 + T_{del}$), the tokens are preserved for t_j so that no other transitions can use these tokens. At time ($T_0 + T_{del}$), the tokens must be removed from t_j's input to output places.

3.4.1 Timing Dynamics of a Timed Petri Net

The pseudo code with the current enabled state x, the PN timing dynamics may be expressed to evaluate the next enabled state x' in a timed Petri Net [18].

Let x is the current enabled state. 'e' is the transition that caused the PN into state x

t is the time that the corresponding event occurred

e' is the next transition to fire (firing transition)

t' is the next time the transition fires ($t'=t+t_{del}$)

x' is the next state given by x' = f(x, e') ,where f() is the state transition function.

3.4.2 Timing Dynamics of a Time Petri Net

In a Time Petri Net two times are associated with each transition [58]. Smallest and the largest of these times for any transition are marked as early-finish-time (EFT) and late-finish-time (LFT) respectively. The firing interval of the transition is the difference between EFT and LFT. States are pairs x = (M, I) in which M is a marking and I is a firing interval function. Firing a transition t, at time Ω from a state x = (M, I), is allowed subject to the following conditions:

i. The transition is enabled

ii. Time Ω is comprised between the EFT and the smallest of the LFTs among the enabled transitions. On firing t at time Ω from a state x= (M, I) moves to a state x' = (M', I') and is computed as follows:

1) The new marking M' for each place is defined for any place P, as in PN, such that
   M'(P) = M (P) - Previous (t, P) + Next (t, P)

2) The new firing intervals I' for transitions are computed as follows:

2.1. ∀ Transitions not enabled by the new marking x', then empty;

2.2. ∀transitions e enabled by marking M and not in conflict with t, then max (0,EFTe - Ω), or max(0, LFTe – Ω) , where EFT_e and LFT_e are the lower and upper bounds of interval I for transition e, respectively;

2.3. All other transitions have their interval set to their Static Firing Interval.

3.5 Two Phase Commit Protocol (2PC)

Let us imitate a simple real life consensus problem about a meeting on some issues for a group of ten persons with four friends. Call all the friends in succession and suggest some activity and a time. If that time is suitable for everybody then confirm the schedule, but if someone can't attend then call each person again and notify them that the activity is abandoned. At the same time suggest another event and date, which then starts another round of consensus. This is an example of a simple consensus and the two clear steps in the process are:

1. Speak to all participants and suggest a value and gather their answers
2. If everyone has the same opinions, contact every accomplice again to let them know, otherwise, contact every participant to abort the consensus.

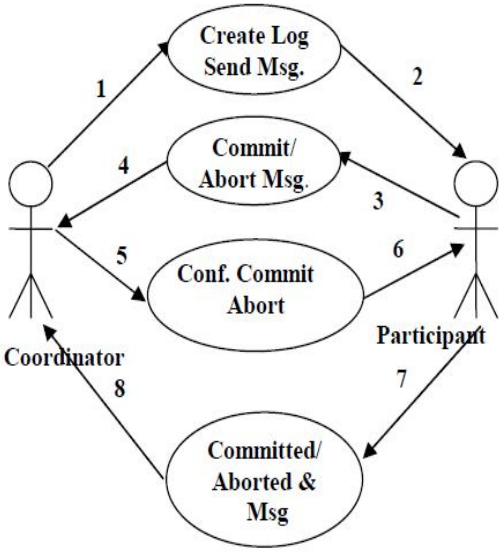

Figure 3.4: Use Case Diagram

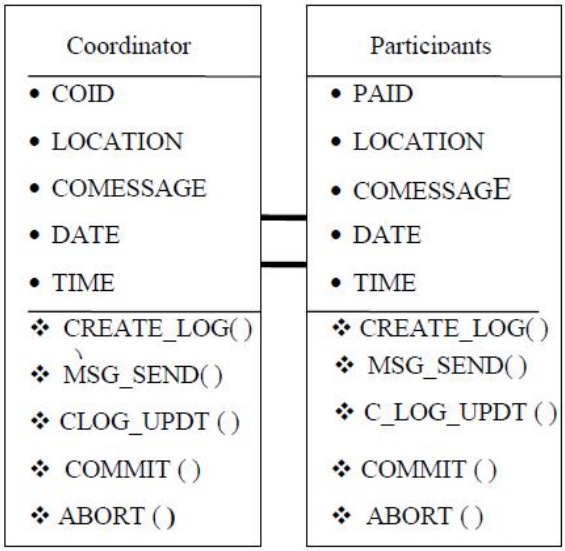

Figure 3.5: Class Diagram of 2PC

This outline describes the simplest, frequently used consensus algorithm called 2PC. As its name suggests, 2PC operates in two distinct phases. At first in the proposal phase for every participant in the system proposes a value and response is gathered. The second commit-or-abort phase communicates the result to the participants and tells them to go ahead and decide or abort the protocol. The process that proposes values is the coordinator, and does not have to be specially designated – any node can act as the coordinator and initiate another round of 2PC.

Let us revisit the distributed transaction management issues like; Atomicity, Consistency, Isolation and Durability; ACID with reference to distributed data model to ensure that the state of the database remains consistent and deterministic [55]. A model on Transaction Management for Distributed Database using PN is designed and revisited the two Phase Commit (2PC) protocol to ensure the consistent commitment of sequence of updates on the stable storages in different locations or aborted as a single complete unit of work [BBS-9]. The uncertainties like happening of an event, synchronization, resource sharing and communications are further formalized from the industrial management perceptions [BBS-2].

Let us now model the coordinator, participant activity of 2-phase commit protocol with the help of a use-case diagram figure 3.4. The model consists of two distinct entities namely coordinator and participant. The occurrences of events are taking place between these two entities. There are four events in the process (create log, send message), (Commit/Abort message), (confirm-commit, confirm-abort) and (Committed/Aborted, send message) .The events are sequenced here numerically to describe the order of the occurrences. A class diagram is presented in figure 3.5. The pseudo code of the use-case functions following the timing sequence is as follows:

1. A LOG record is created at the stable storage with the coordinator marked as “prepare”.
2. MSG: Message to participants to “create” and activate time out.
3. A LOG is created at the participant’s stable storage and marked as “prepare. Write “Ready” or “Abort” message to the log at the participant’s log.
4. MSG: Message to coordinator “Ready” or “Abort”.
5. Coordinator acknowledges and checks all participants reply. Check out with time out parameter and write the decision of commit or abort in the log.
6. MSG: Send “Abort/Commit” message for confirmation to all Participants.

7. Participants acts according to the message received from "(6)" above. Write in the log. Execute the intended operation.
8. MSG: Send the acknowledgement message to all Coordinators.
9. On receipt of all acknowledgements from the participants write "complete" in the coordinators log.

3.5.1 Crashes and Failures in 2PC

Nodes can crash in numerous ways. They can crash and never recover called 'fail-stop' model of distributed system. Nodes may crash and at a later date recover from the failure and continue executing called 'fail-recover' model. However, when recovery is performed in a completely arbitrary way is called 'Byzantine failure'. How can we place a spanner to control such situations? Phase one, when no messages are sent out, the coordinator may crash. This means that 2PC never gets started. If a participant node crashes before the protocol is initiated then nothing bad happens. Now, after some of the proposal messages have been sent, and the coordinator crashes. Some of the nodes received the proposal are starting a 2PC round, and some nodes that are unaware that anything is going on. If the coordinator does not recover for a long time, the nodes that received the proposal are going to be blocked waiting for the outcome of a protocol that might never be finished. This means that no other instances of the protocol can be successfully executed. These nodes have to withdraw their votes from the coordinator without knowing that it has failed – and therefore can't simply timeout and abort the protocol as the coordinator might reawaken at a later time. Let us look at their 'commit' votes and start phase two of the protocol with a commit message. The protocol is therefore blocked on the coordinator, and can't make any progress.

On occurrence of a timeout situation at a node will be forced to finish off the active co-ordinator protocol and that node will contact all the other participants, as in a phase one message, and find out which way they voted. It's also possible that only one node might know the result of the transaction if the co-ordinator fails in phase two before all nodes are told to abort / commit decision. The same kind of arguments applies as broadly covered in phase two. If the co-ordinator crashes before the commit message have got to all participants, a recovery node is required to take over and safely guide the protocol to its conclusion. The worst case scenario is when the co-ordinator by itself is a participant and grants a vote on the outcome of the protocol.

Then a crash to the co-ordinator takes out itself and a participant guarantees that the protocol will stay blocked as a result of only one failure. Further to it the co-ordinator typically will log the result of any successful protocol in persistent storage, so that when it recovers it can answer inquiries about whether a transaction is committed. This allows periodic garbage collection of the logs at the participant nodes. The coordinator can direct the nodes that no-one will try to recover a mutually committed transaction and that can be erased from the log.

3.5.2 Sequence Diagram of the 2PC Distributed System

There are six different states of activities listed in sequence (Figure 3.6) as follows:

1→ Coordinator

2→Create Log and send MSG

3→ Commit message / Abort message

4→ Confirm Commit/ Abort process

5→ Committed / Aborted & send Message

6→ Participants

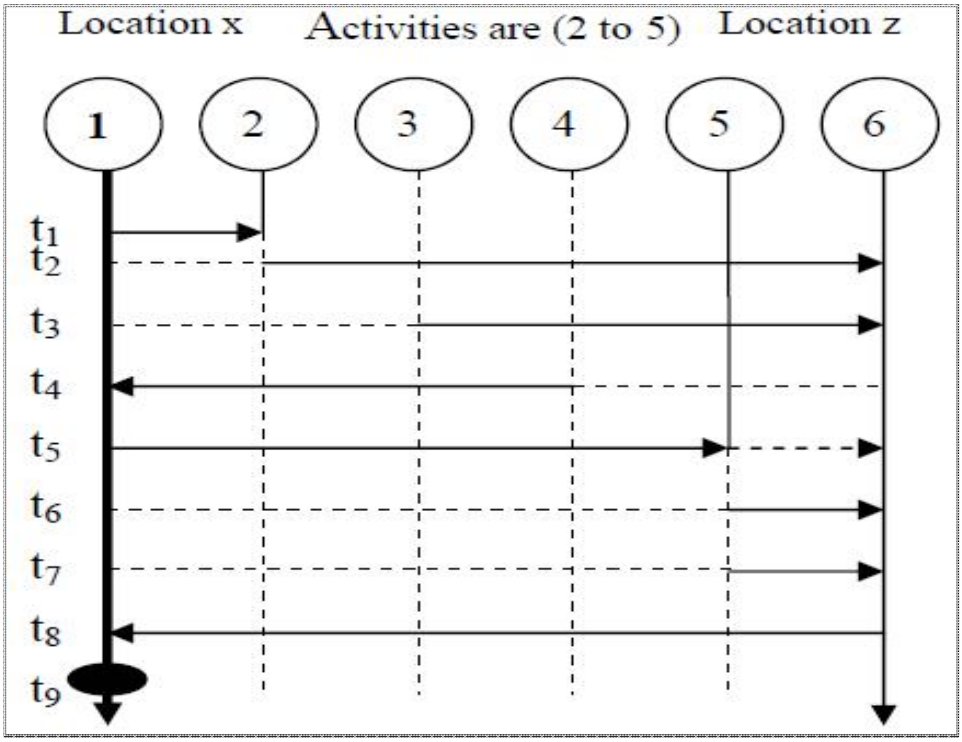

Figure 3.6: Time Sequence Diagram of 2PC

The time domain is from [t_1, t_2, t_3, ----------, t_9]. The Coordinator→1 represents the time intervals of the four activities (2 to 5). Dashed lines indicates the virtual lines, where from the activity should start at some later instant of time. Here t_1 is the starting time when the coordinator creates a log record at the stable storage and at time $t_1+\omega= t_2$ the message reaches to the participants. In the similar fashion the activities are executed till it reaches to t_8. At this point all participants confirm their activity and at $t_8 + \omega = t_9$ the activity will be completed. The following pseudo code represents the basic algorithm of a timed execution sequence of 2PC protocol.

```
Begin
   Activity ← initial t1
   Clock= 0
   Repeat
 For j= t1 to t9 do
   Begin
 If {tk is enabled} then
   <Perform the corresponding activity>
      ω = ω + clock -time
   Until Ωk <= ω [Ωk predefined time value]
End
```

If all the activities are performed within time Ω_k then the distributed 2PC commit process can be performed successfully otherwise a time out condition will be activated. There may be various reasons for possible time outs. Let us indicate some of the possible reasons due to which distributed transactions fails to commit are:

1. Deadlock Occurs
2. System site failures
3. Processor failure
4. Media failures
5. Power supply failure
6. Main memory failure

7. Main memory contents are lost, but secondary
8. storage contents are safe
9. Secondary storage devices Failure
10. Head crash / controller failure
11. Communication failures
12. Lost Messages or undeliverable messages due to
13. Network partitioning

3.5.3 Petri Net Model for 2PC Protocol

A brief overview of the distributed frame work and its 2-Phase Commit functions are presented in section 3.4. There are two different locations: coordinator location and participant's location distributed geographically and marked with doted vertical line in figure: 3.7 of the PN model consisting of nine places and eight transitions. Five places are marked for coordinator and four places for participants. Table-3.4 and Table-3.5 represents the places and transition functions of the coordinator and the participants respectively.

Table 3.4 Place & Transition of Coordinator

Place		Transition	
P1	DTM of coordinator	t_1	Message to DTM of the participating site
P2	Wait state for Commit /Abort	t_2	i) Commit at Global Log ii)"Commit" Msg: to participants
P3	Global Commit	t_3	i) Abort at Global Log ii)"Abort" Msg: to participants
P4	Global Abort	t_4	i) Commit or abort Global Log ii)Write Complete" to Global log
P9	Complete		

Table 3.5 Place & Transition of Participant

Place		Transition	
P5	DTM of participant	t_5	Abort Msg. to local log and coordinator to wait state (P2).
P6	Ready state for Commit	t_6	i) Commit Msg. at local Log ii) "Commit" Msg. to coordinator wait state P2
P7	Abort state	t_7	i)"Abort" Msg. to Coordinator ii)Abort transaction
P8	Commit state	t_8	i)Commit at local Log ii)Commit transaction iii)MSG: to Global log

DTM is the transaction manager at the coordinator side responsible for begin the transaction and it is enabled at P1 when there is at least one token present. The coordinator states are (P_1-P_4, and P_9) and each state has specific function. The corresponding transitions are marked t_1-t_4. The participant's states are (P_5-P_8) and each state has specific function. The corresponding transitions are marked t_5-t_8. The reachable place of a PN can be expressed by the Reachability graph figure 3.8 [59], which is a directed graph and the nodes of the graph are identified as markings of the PN R(N, M_0) ,where M_0 is the initial marking and the arcs are represented by the transitions of N. The graph is used to define a given Petri Net N and marking M, whether M ε R (N). Each initial marking M_0 has an associated peatability set .This set consists of all the markings that can be reached from M_0 through the firing of one or more transitions. In our case the Reachability graph starts the journey with initial marking M_0.

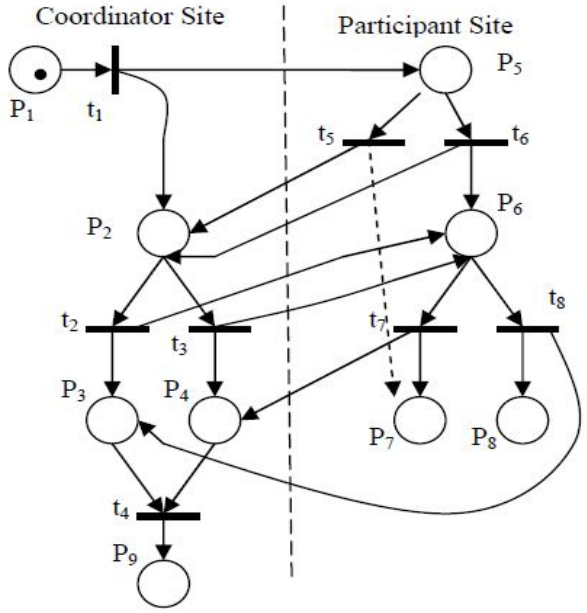

Figure 3.7: Petri net Model of 2PC

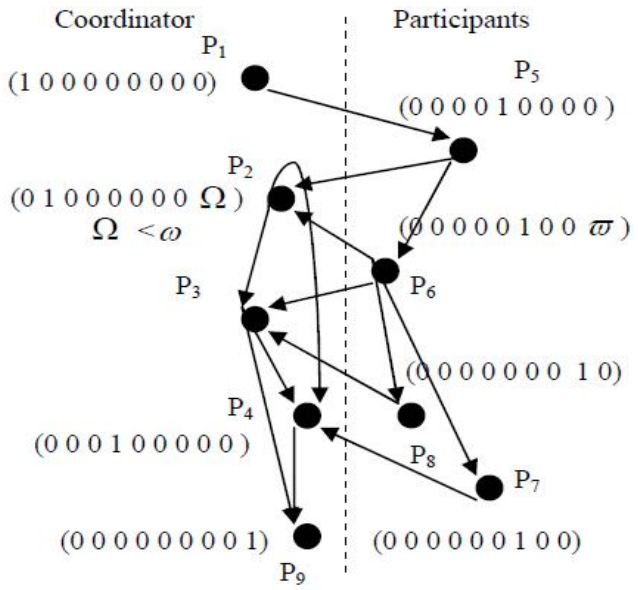

Figure 3.8: Reachability Graph

Reachability analysis is a basic dynamic property of any real-time system. Firing rule of an enabled transition changes the token distribution in a net according to the transition rule but the equality problem is still undecided [12]. Firing rule of an enabled transition changes the token distribution in a net according to the transition rule. In our case marking M_0 (Initial) = [1 0 0 0 0 0 0 0 0 0]^T and finally reach to state M_9(Final) = [0 0 0 0 0 0 0 0 0 1]^T. The place wise markings are presented in figure 3.8. The Reachability graph states some of the essential properties of the

model like safeness (place contains maximum token count 1 or 0), boundedness (place wise token count equals to one), conservativeness (Token count at each place should be same throughout the transitions), and liveness (No deadlock exists). Our model satisfies all the properties.

3.5.4 Reachability Analysis

Let us now describe of the properties and behavior of the model based on the Reachability Graph presented in Figure 3.8 [58].

Safeness: Any place of a Reachability graph is declared safe, if the number of tokens at that place is either 0 or 1. In our case the graph clearly shows that any of the places [$\mathbf{P}_1$ – $\mathbf{P}_9$] represents a combination of 0(no token) and 1(token), which implies that if the firing occurs there will be a token at the position bit otherwise no token. Thus it shows each of the places has a maximum token count 1 or 0 and is declared safe and as all the places in the net are safe, the net as a whole can be declared safe.

Boundedness: The boundedness is a generalized property of safeness. The limitation of token numbers in a place restricted to 1 in case it is safe is enhanced to some integer i, where i is known before hand for a place or we call it as a constraint to check the overflow condition at any stage calculated once at start. Boundary value for each place will be the maximum token count for that place. When there is no overflow at any place, then the design guarantees the boundedness of the mode. In our case at each stage from [P1- P9], i=1 and hence it is bounded.

Conservativeness: Conservation property of a Petri Net model checks the number of tokens remains constant before and after the execution. The process is to count the sum of all tokens at their initial markings. Next the Reachability tree is traversed and the sum of all tokens is calculated for each marking in the tree. In our case it is 1.If all the markings in the Reachability tree have the same sum of tokens, and then the Petri Net is declared to be strictly conservative. So our model is also strictly conservative. However, it will not be out of place to mention that in most cases due to process transformation explicit token counts are difficult to obtain to prove the conservation.

Liveness: The liveness property of a Petri Net is used to show continuous operation of the net model or in other words, it can be said that the system will not get into a deadlock state as the

process of commit or abort needs to perform some transaction processing activity. The possibility of deadlock or live lock cannot be ruled out and to be checked. In order to find whether or not the Petri Net is live; move along the markings of the Reachability tree. If any marking exists in the tree such that no transitions are enabled from that marking, then that marking represents a deadlocked state, and the Petri Net lacks the liveness property. Otherwise it is declared live. In our case there is no such deadlock situation appears in [P1- P9]. So we call the 2-phase commit protocol live. However, problems arise when a Reachability graph is used with loops in it. It may cause a particular place occupied with an infinite number of tokens. This would result in an infinite sized tree.

3.6 Concurrent Database Update

This example is found in operating systems and other multiple synchronization processes. A data repository is to be shared among several concurrent processes. Some of these processes may want only to read the database, while others may update the data repository. The processes are distinguished as readers, and writers. If two readers access the shared data simultaneously, no adverse affects will result. However, if two writers or a reader and a writer access the database simultaneously, chaos may ensue.

In order to guarantee that these difficulties do not arise, it is necessary that the writers have exclusive access rights to the shared database. This is referred to as synchronization problem. There may be several variations involving priorities. The first reader writer problem states that no reader will be kept waiting unless a writer has already obtained permission to use the shared object. Otherwise it can be stated that no readers should wait for other readers to finish simply because a writer is waiting. There could be a different situation when the writer is ready to perform write operation as soon as possible. In other words, if a writer is waiting to access the object, no new reader may start reading. Any solution to either of the problem may lead to starvation. In the first case writers will starve; in the second case, reader will starve. Thus the problem can be summarized as:

- Any number of readers may simultaneously read the file
- Only one writer at a time may write to the file

- If a writer is writing to the file, no reader will read it

Diagrammatically this can be represented with help of read and write locks in order to protect the read write violations.

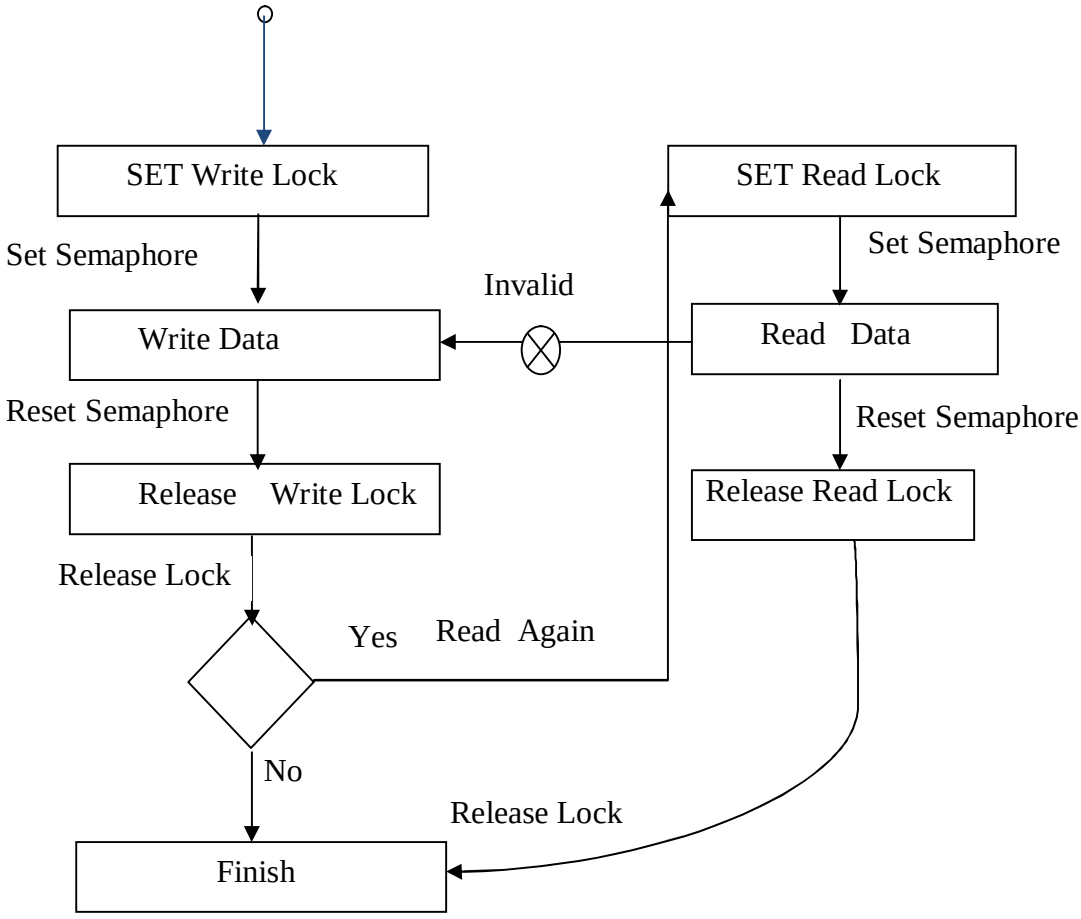

Figure 3.9 Block diagram of Read and Write

According to operating system terminology, mutex and semaphore are kernel resources that provide synchronization services. A mutex provides mutual exclusion; either reader or writer can have the key (mutex) and proceed with their work. As long as the writer is privileged, the reader needs to wait, and vice versa. At any point of time, only one thread can work with the *entire* buffer. The concept can be generalized using semaphore. It is a signaling variable having an integer value. It may be initialized to a nonnegative number. Signal operation increments semaphore value and the wait operation decrements the semaphore values. Acquire the write lock first and set the semaphore variable. Update the data base, then reset the semaphore,

releases the write lock. Now writing process may be finished or repeat the reading process after acquiring read locks and semaphore values. During implementation of the reader/writer problem to a database the following points may be noted.

1. The first reader blocks if there is a writer; any other readers who try to enter block on mutex.
2. The last reader to exit signals a waiting writer.
3. When a writer exits, if there is both a reader and writer waiting, which one gets next access depends on the scheduler.
4. If a writer exits and a reader goes next, then all readers that are waiting will fall through
5. Alternative desirable semantics:
   – Let a writer enter its critical section as soon as possible.

3.6.1 Petri Net Model of Reader/Writer

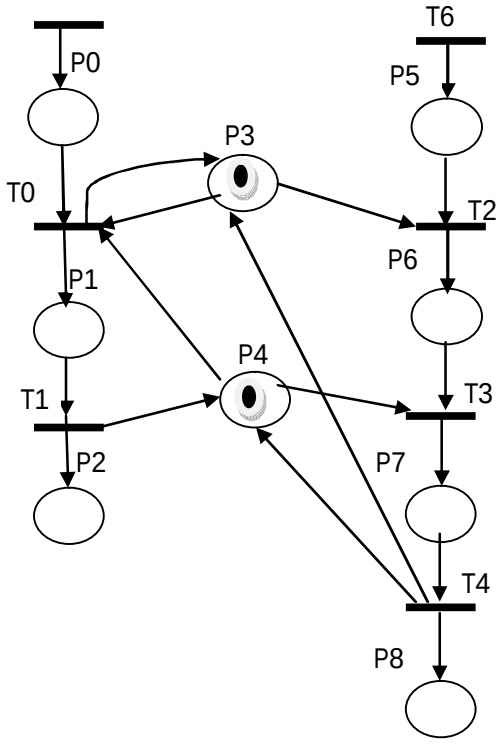

Figure 3.10 PN Model of Reader/Writer

Table 3.6 Positional Reference

Place	Functional Reference
P0	Reader is Ready to Read
P1	Reader is reading
P2	Reader finished reading
P3	Writer can Proceed to Write
P4	Reader can proceed to Read
P5	Writer ready to write
P6	Writer has blocked the reader
P7	Writer is writing
P8	Writer has finished writing

The Transition T2 is always enabled when a token arrives at P5 (unless P3 tokens are present on P6 and P7), since transition T0 is instantaneous. Hence, a writer can always place a token on P6. The first invariant assures that no new token can arrive on P1 since T0 is now disabled until all tokens from P6 and P7 return to P3. Therefore, writers prevent readers from starting. A token on P7 guarantees that the number of tokens on P1 is zero via the second invariant, so writers exclude readers. Similarly, the number of tokens on P4 is also zero, so T3 cannot be enabled, and therefore the number of tokens on P7 is restricted to one; i.e. writers exclude one another. Transition T0 is enabled if a token is on P0 when no tokens are on P6 and P7. If tokens are present on P7, transition T4 is enabled; when it fires, either T3 or T0 is enabled. Successive firings of T4 and T3 will clear any tokens from P6 and P7 eventually enabling T0. When T0 is fired, T1 is enabled. Transition T2 is disabled only if P3 tokens are on P6 and P7, so clearing one by the sequence above enables T2. Again, T3 is disabled only when tokens are on P1 or P7. In that case, T1 and T4 are enabled, so firing them as needed will enable T3. T5 is always enabled if a token is present on P7. Since every transition is live, the entire net is live, assuming that tokens will be placed on P1 and P5 by other transitions in the net. If this net is modified to accommodate the restriction that a place cannot be both the input place and output place of a transition, the arc from T0 to P3 would be replaced by a place and another transition. In general, the behavior would be the same, except that the guarantee that T2 is always enabled when a token arrives at P5 would not be strictly true [99,100]. Thus it can be concluded that if writer is ready to write, no reader may start to read until all waiting writers have finished writing.

3.6.2 Reachability Graph of Reader/Writer

The Reachability Graph of Reader/Writer problem depicts the dynamic property of the synchronization problem [58]. Firing rule of an enabled transition changes the token distribution. The initial marking Mo is [P0=0,P1=0,P2=0,P3=1,P4=1,P5=0,P6=0,P7=0,P8=0]. Firing rule of an enabled transition changes the token distribution P0 (initial)= [1 0 0 0 0 0 0] and P0(Final) = [0 0 0 0 0 10]. The place wise markings are presented in figure 3.11. The Reachability graph states some of the essential properties of the model like safeness (place contains maximum token count 1 or 0), boundedness (place wise token count equals to one), conservativeness (Token count at each place should be same throughout the transitions),

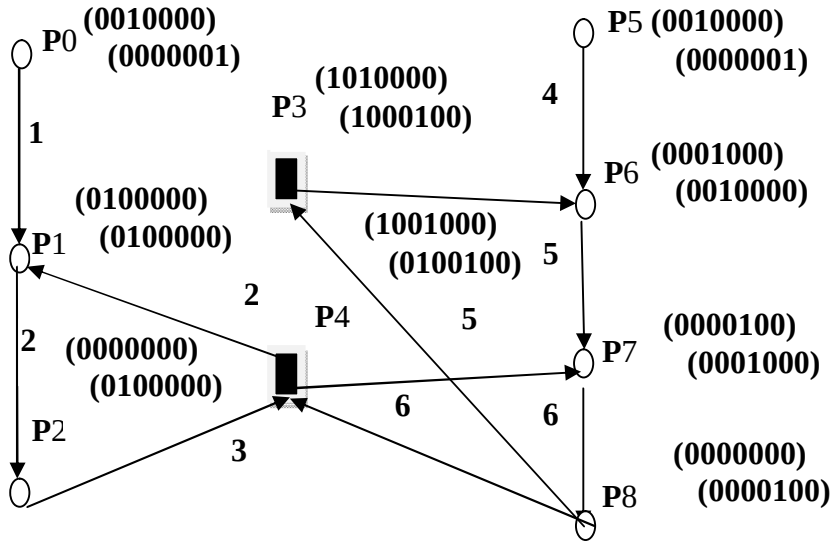

Figure 3.11 Reachability Graph of Reader/Writer

and liveness (No deadlock exists). In our case the net is not bounded, neither safe but the organization of the net with the inclusion of mutex and semaphore the model is free from deadlock nor does the reachability graph clearly show that the reader/writer problem establishes the synchronization mechanism.

3.7 Conclusion

The analysis in section 3.5 and 3.6 shows that the PN model for two phase commit protocol (2PC) and process synchronization in this section are technically sound. Besides, the proposed models reflect the behavioural dynamics of a transaction processing system (TPS). However, this has limited potential for further analysis and improvement of such system. It will not be out of place to note that the work included in this chapter is being cited by 9 other research instances out of which two citations are mentioned in [CT2, CT3], in which our work is cited to describe the typical nature of distributed transactions, while modelling the 3PC protocol and a concurrency control mechanism in a Distributed Databases using hierarchical coloured Petri Net. In the rest of the thesis, we have considered two application domains in the form of Business Process Re-engineering (BPR) and Supply Chain Management (SCM) and modeled using PN towards analyzing the performance of systems and for improving the same.