CHAPTER 5

A STUDY OF DISTRIBUTED PROBABILISTIC RELATIONAL DATABASES

5.1 INTRODUCTION:

Owing to amalgamation of two diametrically opposite technologies namely database technology and computer networks, the concept of distributed database systems came into existence. The database systems integrate the operational data of an enterprise and provide the centralized access to that data whereas computer networks go against all types of centralization efforts. These two approaches can be synthesized to produce a technology that is more powerful and more promising than the either alone. The need for amalgamation arises due to the fact that the objective of database technology is integration and not centralization. Moreover, either of these terms doesn’t necessarily imply each other.

The term “Distributed Systems” is the most abused term in computer science and sometimes it has been called as a “concept in search of a definition and a name”. Certain other terms synonymously used with the distributed systems are: distribution functions, networks, multiprocessors and back-end processing etc.

There are several benefits of distributed processing. For example, distributed processing basically corresponds to the organizational structure of today’s widely distributed enterprises and such a system is more reliable and more responsive. Many current applications of computer technology are inherently distributed like e-commerce over the Internet, multimedia applications etc.

From a more global perspective, fundamental reason behind distributed processing is to be better able to solve the big and complicated problems by using a variation of the well-known divide-and-conquer approach. If necessary software support for distributed processing can be developed, it might be possible to solve these complicated problems simply by dividing them into smaller pieces and assigning them to different software
groups, which work on different computers and produce a system that runs on multiple processing elements but can work efficiently towards the execution of a common task. There are two basic advantages from the viewpoint of economics. First, distributed computing provides an economical method of harnessing more computing power by employing multiple processing elements optimally. The second advantage is that by attacking these problems in smaller groups, working more or less autonomously, it might be possible to discipline the cost of software development.

A distributed probabilistic database system can be defined as a collection of multiple, logically interrelated probabilistic databases distributed over a computer network. A distributed probabilistic database management system (DPDBMS) can be defined as a software system that permits the management of distributed probabilistic databases and makes the distribution transparent to the user. Probabilistic relations are distributed over a network and various aspects related to concurrency control, reliability etc can be studied.

5.2 DISTRIBUTED DATA PROCESSING:

In a uniprocessor system, CPU and I/O operations can be separated and overlapped. This type of separation and overlapping can also be considered as a type of distributed processing. However, there is no consensus on the definition of a distributed system but generally accepted definition is as follows:

"A distributed system consists of autonomous processing elements (PEs) that are interconnected by a computer network and these PEs cooperate in performing the assigned tasks."

Following are the four possibilities pertaining to the question as to what should be distributed in a distributed system:-

(i) Processing logic
(ii) Functions
(iii) Data
(iv) Control

Distributed systems can also be classified with respect to a number of criteria[50] as given below:

(i) Degree of coupling
(ii) Interconnection structure
(iii) Interdependence of components
(iv) Synchronization between components

Degree of coupling refers to a measure that determines how closely the processing elements are connected together. It can be measured as the ratio of the amount of data exchanged to the amount of local processing performed in executing a task. If the communication is done over a computer network, there exists a weak coupling among the processing elements. However, if components are shared then there is strong coupling. Shared components can be either primary memory or secondary storage devices. As for the interconnection structure, one can talk about those cases that have a point-to-point interconnection between PEs as opposed to those which use a common interconnection channel. The processing elements might depend on each other quite strongly in the execution of a task, or this interdependence might be minimal as passing messages at the beginning of execution and reporting the results at the end. Synchronization between PEs might be maintained by synchronous or by asynchronous means. It may be noted here that some of the criteria are not entirely independent. For example, if the synchronisation between PEs is synchronous, one would expect the PEs to be strongly interdependent, and possibly to work in a strong coupled fashion.

The fig. 5.1 shows a central database on a network.

![Diagram of a central database network](image)
The fig. 5.2 shows a DDBMS environment.

**Fig. 5.2**

Fig. 5.3 shows an application (as covered in Chapter 3) describing the applicability of probabilistic databases in a distributed environment.

**Fig. 5.3**
There are various research issues and problem areas if the probabilistic databases are distributed over a network as given below, explained with the help of fig. 5.4:

The Fig 5.4 outlines various research areas that should be resolved to realize the full potential of distribution when applied onto probabilistic databases. The first concept is that of the design of distributed probabilistic database.

**Distributed probabilistic database design:** It relates to how the probabilistic database and the applications that run against it should be placed across the sites. There can be two possibilities: replication or fragmentation. In replication, one can either go for full replication or partial replication. In full replication, the entire probabilistic database is stored at each site. In partial replication, each fragment of the database is stored at more than one site, but not at all the sites. As far as fragmentation is concerned, two issues are to be addressed as follows:

- creation of fragments (whether horizontal or vertical)
- distribution (objective should be optimum distribution of fragments)
Distributed Probabilistic Query processing: Query processing deals with designing algorithms that analyze queries and convert them into a series of data manipulation operations. The problem is how to decide on a strategy for executing each query over a network in the most cost effective way. The factors to be considered are the distribution of data, communication costs etc. The objective should be to optimize the performance of executing the transactions subject to the constraints applicable with every system.

Catalog management: Catalog contains information about data items in the database. A directory may be global to the entire DDBMS or local to each site, there can be single copy or multiple copies.

Distributed concurrency control: Concurrency control involves the synchronization of accesses to the distributed database so that integrity of the database is maintained.

5.3 QUERY EXECUTION FOR DISTRIBUTED PROBABILISTIC RELATIONAL DATABASES

Consider the application scenario as covered in Chapter 3 regarding the Institute planning to celebrate the Diamond Jubilee. Let one more table be maintained at the Institute showing the ID, name, Designation of the person and his/her batch as follows:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Designation</th>
<th>Batch</th>
<th>ps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ram</td>
<td>President</td>
<td>1960</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>Shyam</td>
<td>CEO</td>
<td>1965</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>Bharat</td>
<td>CEO</td>
<td>1965</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Let the Institute want to send this consolidated information to various offices distributed throughout the zones of the country so that proper invitation letters may be sent through zonal offices. One of the approaches is to send the entire table alongwith the stamps of the probability attached to every office assuming that all the offices are interconnected with high speed networks. Sending this probabilistic relation to various offices is also necessary if it is assumed that all these offices will be authenticating the information and in all its certainty, will also try to eliminate or minimize the probabilistic information and attempt to convert it into a deterministic one.
5.3.1 Physical layout of the application scenario:

The structure of such a scheme can be looked upon as a pyramid in which various offices are located in Northern, Southern, Eastern and Western regions and Institute(I) acts as an apex node (or coordinator) of such a pyramid. However, physical layout necessarily need not be a pyramid. It can be any other structure depending upon the interconnection pattern of the network connecting various offices. If this situation is to be represented with the notions of graph theory, zonal offices can be represented by the nodes of the graphs and communication links among these offices can be depicted as edges between the nodes. The overall picture can be shown as in Fig. 5.5:

![Fig. 5.5]

In the next section the three questions related to horizontal fragmentation of the probabilistic relations have been addressed.

5.3.2 Issues of Horizontal fragmentation w.r.t Probabilistic Relations:

For distributing probabilistic relations over a network, two techniques can be adopted namely Replication and Fragmentation. In replication, entire probabilistic relation can be replicated to all the sites whereas in fragmentation original probabilistic relation can be fragmented and then these fragments can be distributed at various sites. The advantages of replication can be the accessibility to the relation despite failure at other sites apart from obtaining speed-up of query processing because of the fact that a query can be divided into subqueries which can be executed in parallel. The primary drawback of such a scheme would be that the information contained at one place may become inconsistent if some updates on some data item are made and the same changes are not
made at other sites which are having mirrored data. However, this drawback cannot be considered too severe if for everytime an update is made, i.e. the updating site becomes the coordinator (or master) of the network and it broadcasts the updation to all the other sites. In practical implementation, having such a floating master gives rise to various problems related to management and control of the sites as well as the data.

On the other hand, fragmentation can be done in one of the two possible ways. It can be either horizontal fragmentation or vertical fragmentation. In horizontal fragmentation, the probabilistic relation can be divided into horizontal fragments that are distributed to various sites. However it gives rise to three basic questions as given below:

(i) How much of the probabilistic relation should be decomposed (in the form of fragments)?
(ii) How can the correctness of decomposition be tested?
(iii) What techniques should be adopted for allocation?

The first point is related with the degree of decomposition wherein the probabilistic relation can be fragmented upto the level of individual tuple or not fragmented at all. Therefore a compromise between two extreme situations should be made. The second point is related to ensuring correctness of decomposition. It should be ensured that all the information contained in the original relation should remain in one of the fragments and that somehow original relation should be obtainable from the fragments after applying some suitable relational operator on them.

As far as allocation is concerned, the allocation of fragments to various sites depends on the network bandwidth and computational capabilities of the machines at various sites. Therefore, only the first two points are of particular interest as they fall in relational domain.

Owing to advantages associated with distributed computing, probabilistic relations will be mapped to a distributed environment and various aspects of query execution in such an environment will be addressed.

A horizontal fragment can be obtained by applying a selection operation on the original relation whereas a vertical fragment can be obtained by projecting the attributes of the original probabilistic relation. Vertical fragmentation of probabilistic relations is not advisable because, in each fragment, primary key of the relation as well as probability
stamp is also kept wherein a lot of space is wasted without offering any additional benefits. Therefore the probabilistic relation is fragmented horizontally by selecting an appropriate selection condition.

The fragments can be obtained by applying suitable selection conditions on probabilistic relation at the apex node. The selection condition should not involve the ps attribute in its predicates because the resultant relation should be free from any other added uncertainty. A suitable selection condition that can be applied on the present relation and in the present situation could be on the basis of the City attribute where that fellow is living. Dividing the relation on the basis of City will further facilitate the invitation process. The predicate could be of the following form:

\[ \sigma <\text{selection-condition}> \]

where \(<\text{selection-condition}> ::= A \text{ relop value};\)

Here \(A\) is the name of the attribute, \(value\) is the value from the domain and \(\text{relop}\) is one of the operators from the set \{=,\(<\), \(<\), \(\leq\), \(>\), \(\geq\)\}.

The selection condition should be applied so that all the values pertaining to that attribute are stored at one place and in one fragment.

For determining the correctness of decomposition, we can take union operation on all the fragments as given below:

\[ \bigcup_{i=1}^{n} (r_i) \]

If we get the original relation then the decomposition is correct.

For allocation, network bandwidth and communication costs are to be taken into account and our objective should be to minimize these costs.

Now, consider the case where the Director of the Institute wants to invite the senior executives personally. For this, names and addresses of those pass-outs are required. Let the query generated at the Institute be as follows:

"Find the names and addresses of those persons where designation = 'President' or 'CEO' or 'Managing Director'."

This query requires joining various fragments with Table 5.1 at the Institute. A strategy is needed to be adopted that will minimize not only processing costs but also communication costs. The most obvious strategy for the execution of query would be to transfer all the fragments lying at zonal offices to the Institute and perform join of these
fragments with the Table 5.1 lying at the Institute. As an alternative, we can transfer only ID, Address, Batch and ps attributes of each fragment from zonal offices and then perform join operation at the Institute. This could be done by projecting these four attributes, storing them in a temporary relation and then transferring these temporary relations to the Institute. At the Institute, these temporary relations can be joined with Table 5.1 and results for the query could be obtained. This operation can be called probabilistic semijoin or psemijoin (because it contains a probabilistic attribute). The amount of data transfer will definitely be less in the later case. An algorithm is being proposed that will be useful in implementing this psemijoin operation. In the example query, the knowledge of what is required and how that can be obtained is already known. In a way it was clear what attributes should be projected and how and where these should be joined. For an arbitrary query, this operation needs to be automated. Given below is an algorithm that will be useful for performing psemijoin operation for any arbitrary query.

5.3.3 Algorithm to perform psemijoin:

Let database be stored at $N$ sites. The generalized algorithm is given below:

Algorithm psemijoin()
begin
For $i=1$ to $N$
begin
Compute at $S_i$
$\text{Temp}_i = \Pi r_i \cap I_i (R_i)$
// $S_i$ is the site
// $r_i$ is the schema of the relation at site $I$ and $I_i$ is the schema of the site relation
// where relation $R_i$ is required. The intersection of both the schemas is computed and put in the temporary relation, $\text{Temp}_i$
// that contains the projected
Transfer Temp_i to I;
Compute joins of various Temp_i s at I
end.
end.

It may be noted here that the computation of schemas of the sites with the Institute site at I_i makes sense because not all the tuples or attributes will be required to perform the join. Obviously, the result of query will need join of relations at both the sites and it makes sense to let off superfluous attributes or tuples. Since this operation is to be carried out with respect to a variety of sites, parallelism can also be used and this will further reduce the processing time although some complexity may be introduced in parallel processing of the requests. At the Institute, as a result of the execution of the query, the names and addresses of those pass-outs that have the designation of 'President' or 'CEO' or 'Managing Director' along with their associated stamps of probability can be had. The stamps of probability in the resultant relation will be the product of stamps of probabilities in the participating relation. However in its present form, this information is not useful as far as sending invitation letters is concerned. If the Institute wants to solicit some financial grants from those pass-outs, it needs to know with certainty where that pass-out is residing. To solve this problem, a threshold value of probability can be adopted that will suitably accept or reject some of the persons for invitation or for sending request for financial aid. The greater the value of this threshold probability, the fewer will be the number of invitees. Therefore this threshold value should be accommodated so that the errors can be minimized. The selection of this threshold value can be made depending upon some method of heuristic or adopting some statistical technique on the basis of suitably sampling the population.

5.4 DISTRIBUTED CONCURRENcy CONTROL ALGORITHMS
FOR PROBABILISTIC DATABASES:
Serializability theory is the most widely accepted correctness criteria for concurrency control algorithms. A review of serializability theory is provided in the following section
and it is shown how the theory can be applied to distributed probabilistic relational databases.

5.4.1 A review of serializability theory:

A schedule (S) also called as history, is defined over a set of transactions \( T = \{ T_1, T_2, \ldots, T_n \} \) and specifies an interleaved order of execution of the operations of these transactions. Two operations accessing the same database item \( x \) are said to conflict if at least one of them is write. Since read operations do not conflict, therefore two types of conflicts are talked about: read-write (or write-read) and write-write. Intuitively if there is a conflict between two operations then it means that their order of execution is important. The ordering of two read operations is insignificant.

A complete schedule defines the execution order of all operations in its domain. A schedule, therefore, can be defined as a prefix of a complete schedule. Formally, a complete schedule \( S \) defined over a set of transactions \( T = \{ T_1, T_2, \ldots, T_n \} \) is a partial order \( S = \{ \Sigma_T, \leq \} \) where

1. \( \Sigma_T = \bigcup_{i=1}^n \Sigma_i \)
2. \( \leq \supseteq \bigcup_{i=1}^n \leq_i \)
3. For any two conflicting operations \( O_1 \) and \( O_2 \in \Sigma_T \), either \( O_1 \leq O_2 \) or \( O_2 \leq O_1 \).

The first condition states that the domain of the schedule is the union of the domains of individual transactions. The second condition specifies the ordering relation as a superset of the ordering relations of the individual transactions. By virtue of this condition the ordering of operations within each transaction is maintained. The last condition specifies the execution order among conflicting transactions.

Consider the transactions as given below:

- \( T_1: \) Read(x) x:=x+1 write(x) Commit
- \( T_2: \) Read(x) x:=x+1 write(x) Commit

A possible complete schedule \( S \) over \( T = \{ T_1, T_2 \} \) can be written as following partial order (where the subscripts indicate the transactions):

\[
S = \{ \Sigma_T, \leq \}
\]
where $\Sigma_1 = \{ R_1(x), W_1(x), C_1 \}$
and $\Sigma_2 = \{ R_2(x), W_2(x), C_2 \}$

therefore,

$\Sigma_T = \Sigma_1 \cup \Sigma_2 = \{ R_1(x), W_1(x), C_1, R_2(x), W_2(x), C_2 \}$
and $\leq = \{ (R_1, R_2), (R_1, W_1), (R_1, C_1), (R_1, W_2), (R_1, C_2), (R_2, W_1),$
$(R_2, C_1), (R_2, W_2), (R_2, C_2), (W_1, C_1), (W_1, W_2), (W_1, C_2), (C_1, W_2),$
$(C_1, C_2), (W_2, C_2) \}$

which can be specified with the help of a Directed Acyclic Graph (DAG) as given in the Fig. 5.6:

![Fig. 5.6](image)

The arcs implied by transitivity are omitted in DAG.

A schedule is defined as a prefix of a complete schedule. A prefix of a partial order can be defined as follows:

Given a partial order $P = \{ \Sigma, \leq \}$, $P' = \{ \Sigma', \leq' \}$ is a prefix of $P$ if

1. $\Sigma'$ is a subset of $\Sigma$
2. For all $e_1 \leq e_2$ if and only if $e_1 \leq e_2$
3. For all $e \in \Sigma'$ if there exists $e_1 \in \Sigma$ and $e \leq e_1$ then $e \in \Sigma'$

The first two conditions define $P'$ as a restriction of $P$ on domain $\Sigma'$, whereby the ordering relations in $P$ are maintained in $P'$. The last condition indicates that for any element of $\Sigma'$, all its predecessors in $\Sigma$ have to be included in $\Sigma'$. The definition of a schedule as a prefix of a partial order provides us with a mechanism that facilitates dealing with incomplete schedules. From the perspective of serializability theory we can deal with only those operations of transactions that deal with those operations of transactions that conflict rather than dealing with all the operations. From the perspective of failure, we need to be able to deal with incomplete schedules and this process is
facilitated by an incomplete schedule. The schedule as discussed in the preceding section is complete because it deals with the execution order of these two transactions. The following example gives a schedule that is not complete.

Consider the case of the following three transactions:

<table>
<thead>
<tr>
<th></th>
<th>T1: read(x)</th>
<th>T2: Write(x)</th>
<th>T3: Read(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Write(x)</td>
<td>Write(y)</td>
<td>Read(y)</td>
</tr>
<tr>
<td></td>
<td>Commit</td>
<td>read(z)</td>
<td>Read(y)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>commit</td>
</tr>
</tbody>
</table>

A complete schedule is depicted in the Fig. 5.7:

![Fig. 5.7](image)

A prefix of the complete schedule is as shown in the Fig. 5.8:

![Fig. 5.8](image)

If in a schedule S, the operations of various transactions are not interleaved (i.e. the operations of each transaction occur consecutively), the schedule is said to be serial. The serial execution of the transactions shall maintain the consistency of the database. It also follows naturally from the consistency property of transactions: each transaction, when
executed alone on a consistent database, will produce a consistent database. Consider the following schedule:

\[ S = \{ W_2(x), W_2(y), R_2(z), C_2, R_1(x), W_1(x), C_1, R_3(x), R_3(y), R_3(z), C_3 \} \]

It is a serial schedule since all the operations of \( T_2 \) are executed before all the operations of \( T_1 \) and all operations of \( T_1 \) are executed before all operations of \( T_3 \). One common way to denote this precedence relationship between transactions is \( T_2 \rightarrow T_1 \rightarrow T_3 \).

Based on the precedence relationship introduced by the partial order, it is possible to discuss the equivalence of schedules with respect to their effects on the database. Two schedules \( S_1 \) and \( S_2 \) defined over the same set of transactions \( T \), are equivalent if they have the same effect on the database. Formally, two schedules \( S_1 \) and \( S_2 \) are said to be equivalent if for each pair of conflicting operations \( O_{jk} \) and \( O_{kl} \) \((i \neq k)\), whenever \( O'_{j} S O_{kl} \) then \( O_{ij} S O_{kl} \). It is called as conflict equivalence since it defines equivalence of two schedules in terms of relative order of execution of the conflicting operations in those schedules. The following schedule is conflict equivalent to \( S \) of previous example

\[ S' = \{ W_2(x), R_1(x), W_1(x), C_1, R_3(x), W_2(y), R_3(y), R_2(z), C_2, R_3(z), C_3 \} \]

A schedule is said to be serializable if and only if it is conflict equivalent to a serial schedule. It may be noted here that serializability refers to degree 3 consistency. The primary function of a concurrency controller is to generate a serializable schedule for the execution of the pending transactions.

Serializability theory can be extended to distributed databases. The schedule of transaction execution at each site is called a local schedule. If the database is not replicated and each local schedule is serializable, their union (called as global schedule) is also serializable as long as local serialization orders are identical.

The same theory can further be extended for distributed probabilistic databases by defining a probabilistic transaction and ensuring the joint and marginal probability distributions of the probabilistic transaction.

5.4.2 Taxonomy of concurrency control mechanisms:

Concurrency control mechanisms can be divided into two basic categories as follows:

Pessimistic algorithms and optimistic algorithms. Pessimistic algorithms synchronize the concurrent execution of transactions early in their execution life cycle, whereas optimistic
algorithms delay the synchronisation of transactions until their termination. The classification is as depicted in the Fig. 5.9:

In the locking based approach, the synchronization of transactions is achieved by employing physical or logical locks on some portion of the database. This class can be further subdivided according to where the lock management activity can be performed as follows:

1. In centralized locking, one of the sites in the network is designated as the primary site where the lock tables for the entire database are stored and is charged with the responsibility of granting locks to transactions.
2. In primary copy locking, one of the copies of each lock unit is designated as the primary copy, and it is this copy that has to be locked for the purpose of accessing that particular unit. For example, if lock unit x is replicated at sites 1, 2, and 3, one of the sites (say 1) is selected as the primary site for x. All transactions desiring to access x obtain their lock at site 1 before they can access a copy of x. If the database is not replicated (i.e., there is only one copy of each lock unit), the primary copy locking mechanisms distribute the lock management responsibility among a number of sites.

3. In decentralized locking, the lock management duty is shared by all the sites of a network. In this case, the execution of a transaction involves the participation and coordination of schedulers at more than one site. Each local scheduler is responsible for the lock units local to that site. Using the same example as above, entities accessing x must obtain locks on all the three sites.

5.5 LOCKING BASED CONCURRENcy CONTROL ALGORITHMS:
The main idea of locking based concurrency control is to ensure that the data that is shared by conflicting operations is accessed by one operation at one time. This can be accomplished by associating a "lock" with each lock unit. The lock is set by a transaction before it is accessed and is reset at the end of its use. Obviously, a lock unit cannot be accessed by an operation if it is already locked by another. Therefore, a lock request is granted only if the associated lock is not being held by another transaction. In this section a locking algorithm is presented that will help in controlling concurrency in distributed probabilistic relational databases. The proposed p-locking algorithm uses a global locking scheme to serialize conflicting operations of global probabilistic transactions. Global locking tables are used to lock data items involved in a global probabilistic transaction in accordance with the two-phase locking (2 PL) protocol. Maintaining a global locking table may require communication of information from the local site to the global transaction manager (GTM) regarding locked data items. The proposed concurrency control algorithm is based upon the following assumptions:
(a) There is no distinction between local and global probabilistic transactions at the local level.

(b) A local site is completely isolated from other sites.

(c) Each local system ensures local serializability and freedom from local deadlocks.

The global locking table can be used to create a global wait-for-graph which can be subsequently used to detect and resolve potential global deadlocks. The wait-for-graph can be defined as a directed graph that shows dependence/independence among global probabilistic transactions. Each node in the graph represents a probabilistic transaction and an edge $T_i \rightarrow T_j$, from node $T_i$ to node $T_j$, represents the fact that probabilistic transaction $T_i$ is waiting for transaction $T_j$ to release some lock. In a distributed environment, each scheduler at site $I$ can maintain a local wait-for-graph. In addition, the global manager also maintains a global wait-for-graph that is the union of all local wait-for-graphs.

The following definitions are used to describe the p-locking algorithm that can be used for distributed probabilistic relational databases.

(i)  **Operation**: can be one of the Begin Transaction, Read item, write item, abort or commit.

(ii) **DataItem**: A data item in the distributed probabilistic database.

(iii) **TransactionId**: A unique identifier assigned to each transaction.

(iv)  **DataVal**: A primitive data-type value (e.g. integer, real etc)

(v)  **SiteId**: A unique site identifier

(vi)  **Probability_stamp**: $ps$

(vii) **Dbop**: A quintuple of {a database operation from the application program}

$opn$: Operation  
$data$: DataItem  
$tid$: TransactionId  
$ps$: Probability_stamp

(viii) **Dpmsg**: A 3-tuple of {a message from the data processor}

$opn$: Operation
tid: TransactionId
res: DataVal

(ix) Sems: A 3-tuple of { a message from the scheduler}
opn: Operation
tid: TransactionId
res: DataVal

(x) Transaction: A 2-tuple of
tid: TransactionId
body: The transaction body

(xi) Message: A string of characters to be transmitted

(xii) Opset: A set of Dbop’s

(xiii) Siteset: A set of siteId’s

(xiv) WAIT(msg: Message)

begin
{ wait until a message arrives}
end

The formal algorithm is given below:

Algorithm p_locking_for_distributed_CC( )

Variable declaration:
msg: Message
dop: Dbop
Op: Operation
x: dataItem
T: TransactionId
pm: Dpmsg
result: DataVal
Set_of_op: OpSet

Begin
Repeat for each site participating in transaction execution:
Case : msg
Dbop:
Begin

Sid ← SiteId
Op ← dop.opn
X ← dop.data
T ← dop.tid
Ps ← Probability_Strip

Case : Op

Begin_Transaction:
Begin
Send dop to the data processor
End

Read_Item or Write_Item: // would require probabilistic locking
Begin
Find the lock unit, Ig, such that x lies within Ig and all occurrences
of x are locked without violating any of the axioms of probability.
If Ig is unlocked
Begin
Set lock in appropriate mode
Send dop to the data processor
End
Else
Put dop on a queue for Ig
End-if
End

Abort or commit:
Begin
Send dop to data processor
End

Dpmsg: // acknowledgement from data processor
Begin
Op ← pm.opn
result ← pm.result
T ← pm.tid
Ps ← Probability_Sign

If Op = Abort or Commit then
Begin
For each lock unit lg locked by T do // unlocking required
Begin
Release lock on lg held by T
End.
End.
End.