CHAPTER 1

INTRODUCTION

1.1 Distributed Environment

Distributed environment is a collection of loosely coupled processors interconnected by a communication network that work together collaboratively. The location of the processor is referred as a site. So typically, a distributed system is a collection of sites.

There are four major reasons for building distributed systems namely resource sharing, computation speedup, reliability and communication. In a distributed environment, resources at one site can be accessed by the local as well as remote users. The resource can be a physical resource like high speed processor, printer etc., or a logical resource like files, databases etc. Computation speedup can be achieved by shifting the jobs from overloaded site to a slack site. This idea of load sharing among the sites improves the performance of the system. Apart from the performance, reliability is also increased by providing fault tolerance. When one of the sites fails due to hardware failure or software bug, its function is automatically taken over by another site. Though this requires redundancy of data, it improves the reliability of the system. The failure of one site does not affect the functioning of other sites in the system.

Distributed system can be centrally managed or truly distributed. In a centrally managed distributed system, one of the sites is chosen as the controller. The controller site is responsible for managing and coordinating the functions of all the other sites. This poses a bottleneck threat. I.e., everything relies on this controller. If the controller site fails, the entire system fails. So this setup is not favored much.

In the truly distributed system, each site allows autonomous management. Each site is responsible for managing itself. Therefore each site is unaware of the activities happening at other sites. They communicate only through the messages. The structure of a truly distributed system is given in figure 1.1.

In the figure, each site has a local Transaction Manager (TM) whose function is to serve the clients arriving at the site. It manages the execution of the transactions that access data stored in this site. The resources needed for the transaction can be accessed by requesting the Resource Manager (RM) local to the site. It has to provide
appropriate concurrency control techniques for the local as well as the remote transactions arriving at this site to access its resource. It is also responsible for getting access to remote resources on behalf of its local transactions.

**Figure 1.2** Client/server model in distributed systems

In a distributed system, the clients’ transactions are served by executing procedures stored in the server. The server accesses the data resources in the form of databases like relational databases and object oriented databases etc. It is the responsibility of the data resources to ensure their data consistency. So only the full fledged database management systems can be connected at the back end. The legacy file formats cannot be supported. Figure 1.2 shows the client/server model practiced in the distributed system.

### 1.2 Distributed Object Environment

The earliest and most relevant applications of distributed system were in business and administration. Recently several applications have cropped up that require support for the management of complex data types. There are several application areas that are limited by the restrictions of the relational data model and procedural
approach. Hence the object oriented approach is brought into the design of distributed systems. It has the features of encapsulation, abstraction, security and reusability.

The object model has structure and behavior. It encapsulates a set of attributes that contain the data and a set of methods that have a body of code to implement a service. The interface between an object and the rest of the system is defined by a set of allowed messages. The objects are interrelated by inheritance, aggregation and association relationships.

Typically objects exist in distributed systems in two forms namely Object Oriented Databases (OODBMS) and Object Oriented Distributed Systems (OODS).

1.2.1. Object oriented databases

The first choice for the applications that require support for complex data and long duration transactions is obviously object oriented databases. OODBMS is a collection of objects. The objects are classified into classes and instances. A class is a collection of instances. Objects represent complex data. Their complex relationships are defined by combinations of object relationships such as inheritance, composition (aggregation) and association. The support of complex objects imposes several requirements on both the object data model and object management. They are

- The object model should support structural modeling and interrelationships in a natural way.
- It must also support modeling of object behaviors and dynamic constraints.
- In addition in the intended application environments, the object structures, behavior and interrelationships may evolve over time [Bertino1991]. From the object management perspective, the following features are required.
  - Object versioning mechanisms to support evolution of objects.
  - Transactions can extend in time and involve large amounts of data. This requires the complex recovery and concurrency control mechanisms.
  - Due to the evolutionary nature of applications, extension of schema modification operations should be supported without requiring system slowdown or shutdown along with the execution of data requests.
  - Should provide security mechanisms.

Several advanced database products like Avance, Encore, Gemstone, Iris, O2, Orion and Vbase have provided support for the aforementioned features.
1.2.2. **Object oriented distributed systems**

Object oriented distributed system is the result of merging object oriented techniques with distributed systems technology. This approach makes objects as the unit of computation and distribution. It has the ability to encapsulate both data and operations in a single computational unit. It has the following benefits:

- Systems with diversified behavior for the same service request can be provided using polymorphism.
- It can support heterogeneous environments by its ability to separate their interface from their implementation.
- It can support the continuous evolution of business domains to suit their new requirements.
- Encapsulation can hide the implementation and therefore easily change the object behavior without affecting the users much.
- There is a uniform invocation irrespective of whether it is local or remote call using message passing.

In OODS, the objects are viewed as reusable data resources. Figure 1.3 shows the client/server model of the object oriented distributed systems.

![Client/server model in object oriented distributed system](image)

**Figure 1.3** Client/server model in object oriented distributed system

In this figure, the client’s request is satisfied by invoking the methods in the objects residing at server tier. So unlike distributed system which is procedure oriented, OODS is action oriented. The objects encapsulate the attributes and associated methods or member functions. The objects are related by inheritance, aggregation and association. Inheritance defines a ‘parent-child relationship’. Aggregation defines ‘has a’ relationship. The association is used to define all other relationships such as associated-with, using etc. The domain objects and their relationships are represented as a class diagram.

1.3 **Concurrency Control and Deadlock Handling Techniques**

In a distributed system, typically several transactions may arrive at a server site at the same time. These parallel transactions should not affect the consistency of the
data. They should not lead to dirty reads or dirty writes. The ACID property of the database has to be preserved. Ideally parallel reads are allowed. Mixed read and write operations are not allowed. Parallel write operations are also not allowed.

The business domains are continuously evolving in nature. They might want to improve their services to the clients. This requires modification of schema to reflect the changes in the business domain. Transactions would arrive to modify the schema. Then all the consistency requirements defined for data have to be extended for schema also.

Concurrency control mechanisms are applied to synchronize the transactions accessing the database to maintain data and consistency. Complex concurrency control mechanisms are needed for the object oriented system because of its complex nature. However, the concurrency control mechanisms should not affect the performance of the system. A good concurrency control mechanism should improve the throughput of the system by improving concurrency so that maximum number of transactions can run in parallel.

Timestamp Ordering, Optimistic Concurrency Control (OCC) and Locking are the commonly used concurrency control techniques. The timestamp ordering is based on the time stamps assigned to the transactions as they arrive at the system. The transactions are served in the system in First Come First Served (FCFS) order. But it is very difficult to implement time stamps in distributed systems.

OCC is a validation based protocol. Each transaction executes in two or three different phases in its lifetime, depending on whether it is a read or update operation. The phases are

1. Read phase- Reads all data items and write operations are performed on temporary local variables.
2. Validation phase- Checks the validity of updating the database with temporary variable values without any conflicts.
3. Write phase- If the validation phase is successful, the database is updated. Otherwise the transaction is rolled back.

This is a better scheme than timestamp ordering, but there is a possibility of starvation of long duration transactions.

Locking is the most common method of concurrency control. Among the various concurrency control mechanisms, locking is widely used because of its ease in implementation. It enforces the requirement of allowing a transaction to access a data
item only if it holds a lock on it. In lock based concurrency control scheme, a transaction has to acquire locks before accessing the database and release them after use. Locking technique uses compatibility or commutativity matrix to decide whether a new transaction can concurrently execute with those that are already executing without affecting consistency. If the lock modes are compatible, the transactions can execute. If the lock modes are conflicting, then the transaction that is requesting the lock will be blocked. It has to wait until the transaction currently holding the resource has released the lock or preempted. There are two possible lock modes.

1. Shared (S) – To read a data item. Write is not allowed. Several transactions can use this lock mode to share a data item.
2. Exclusive (X) - Both read and write operations can be performed on the data item. But it cannot be shared.

The compatibility matrix for S and X lock modes is given in the table 1.1.

**Table 1.1** Compatibility matrix for S and X lock modes

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>X</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

In the table, ‘Y’ indicates locks are compatible. ‘N’ indicates locks are incompatible. The locking technique requires a lock table along with the compatibility matrix. The lock table defines the current lock status of all the data items in the domain. The lock table has to be updated on every lock request and lock release message.

The performance of locking technique is further improved by Multi-Granular Lock Model (MGLM). It is a common technique for implementing concurrency control on transactions using the databases. Using MGLM, transactions can request the same database in different granule sizes varying from coarse granules to fine granules. This maximizes the concurrency while minimizing the number of locks. The main advantages of MGLM are high concurrency and minimal deadlocks. There are several multi-granular lock models proposed in the literature. Gray1978 has defined semantic MGLM for relational databases. In MGLM, intention locks are used to infer the presence of locked resources at a smaller granule level. The lock modes defined in Gray1978 are Intension locks –IS(Intension Shared) and IX (intension Exclusive) to
lock fine granules, S (Shared - Read), X (eXclusive – Write) and SIX (Shared Intension eXclusive – locks all in S mode but a few of them to be updated alone in X mode). The compatibility matrix proposed in Gray1978 is given in table 1.2.

Table 1.2 Compatibility matrix of Gray1978’s multi-granular lock model

<table>
<thead>
<tr>
<th></th>
<th>IS</th>
<th>IX</th>
<th>S</th>
<th>X</th>
<th>SIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>IX</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>S</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>X</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SIX</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

This multi-granular lock model is extended to object oriented environments also with suitable modification by applying the semantics of the object oriented paradigm.

Though concurrency control techniques ensure consistency of data and schema, they have the negative effect of introducing deadlocks. A system is in a deadlock state if there is a set of transactions such that every transaction is waiting for another transaction in the set in a circular way. None of the transactions can make progress in such a situation. This results in poor throughput and high transaction response time which are undesirable.

In distributed systems, the dependency among transactions can be described as a directed graph called as Global Wait-For Graph (GWFG). The graph consists of a pair $G = (V, E)$ where $V$ is a set of vertices representing active transactions, $E$ is a set of edges representing their dependency. When there is an edge $T_i \rightarrow T_j$, it indicates that $T_i$ is waiting for $T_j$ to release a data item that it needs. The presence of a deadlock in the system can be inferred from the presence of cycle in the GWFG.

There are two principal methods to deal deadlocks. We can use a deadlock prevention algorithm to ensure that the system will never enter a deadlock state. It is a pessimistic and proactive approach. Alternatively, we can allow the system to enter a deadlock state and then try to recover by using deadlock detection and recovery scheme. This is an optimistic and reactive approach.

There are two approaches to deadlock prevention. One approach ensures that no cyclic waits can occur by ordering the transactions or holding all the resources before execution. In resource ordering technique, an ordering is imposed on all the data
items. It insists on the transactions to lock the data items in a sequence consistent with the ordering rules. This scheme is easy to implement as long as the set of data items accessed by a transaction is known, when the transaction starts execution. The other approach is to roll back, whenever the wait could potentially result in a deadlock.

Several deadlock detection and resolution algorithms are available in the literature. One of them is the popular probe based deadlock detection algorithm presented in Chandy1983. It is very popular because of its easy implementation. In this algorithm, when a transaction suspects deadlock, it sends a probe along the edges of GWFG. If the probe comes back to the initiator, it indicates the presence of deadlock.

When the detection algorithm detects a deadlock existence, the system must recover from the deadlock. The most common solution is to roll back one or more transactions to break the cycle. Selection of a victim should be such that there is no starvation or degradation of desirable parameters such as throughput, response time and resource utilization of the system.

1.4 Motivation

Nowadays, business domains are implemented as distributed applications for reaching wider range of clients. In distributed systems, the business data and methods are scattered across various sites. Distributed databases are used to store the business data. Business methods are implemented as procedures. In order to avoid the inconsistency of the business data due to concurrent access by several clients, concurrency control techniques are applied on the databases in the database tier. The legacy data files cannot be used in the distributed systems because they are primitive and do not provide concurrency control. Further, distributed systems require separate concurrency control technique for each of the database models in which the business data are stored.

Hence, the possibility of shifting the concurrency control mechanisms from the database tier to the server tier is explored. In distributed systems, it is not possible to shift the concurrency control to server tier as it is implemented using procedures. This is possible in OODS because the client transactions to the database tier can be accessed only through the objects in server tier. This will help the OODS to support legacy file formats also. Moreover, it is sufficient to provide a common concurrency control mechanism independent of the nature of persistent storage. While exploring the requirements of concurrency control mechanisms for OODS, the concurrency
control mechanisms defined for OODBMS is a good place to look for as it is the only other distributed object environment.

The three common concurrency control mechanisms such as locking, timestamp ordering and optimistic concurrency control have been dealt in the literature for adoption in OODBMS. Locking is more widely used than the other two techniques, because of its ease in implementation. MGML- one of the types of locking, is a common technique for implementing concurrency control on transactions using the OODBMS. The other reason for the wide usage of multi granular locking is that it allows application of the semantics of object oriented paradigm to improve the performance.

Even though the concurrency control mechanisms in OODBMS can be considered, they cannot be adopted as they are in OODS. This is because query languages are used to request data from the OODBMS. But in OODS, object oriented programming languages like C++ and Java are used to make the client transactions. Then the lock types and the granularities of resources are to be ascertained from the client code using document tools. Document tool is a tool that parses the declarations and documentation comments in a set of source files and produces a set of HTML pages describing the classes, interfaces, constructors, methods, and fields.

The doc tools like docC++ and Javadoc can be used for identifying the method type and properties. After identifying the lock modes for all the classes used in the client code, the commutativity matrix defined in OODBMS can be extended in OODS.

All the business domains might eventually want to upgrade their services to clients. This evolution of business domain introduces the need to change the structure of the business domain. The domain is represented using the class diagram in OODS. Then the evolution of the business domain might require changes in the definitions of attributes, methods, classes and their relationships.

Usually, the object oriented distributed systems receive runtime transactions requesting the access of business data. Due to the continuous evolution of business domains, the systems may also receive transactions to modify the structure. These transactions are called as design time transactions. Both types of transactions can be either read or write transactions. This requires the application of concurrency control to protect the consistency of the objects. These two types of transactions may induce three different types of conflicts among the transactions to a class diagram: conflicts
among run time transactions, conflicts among design time transactions and conflicts between run time and design time transactions.

In OODBMS, both types of transactions can be executed in parallel as transactions are written in query languages. In OODS, the transactions are implemented in programming languages as mentioned earlier.

Then proposing a concurrency control mechanism for OODS involves two steps:

1. Define lock types and granularity for all types of methods defined in the classes that are related by relationships namely inheritance, aggregation and association and represent the business domain.

2. Propose a compatibility matrix based on class relationships that will address the conflicts mentioned above.

The existing semantic based MGLM for OODBMS can be classified as MGLM based on relationships and MGLM based on operations. Models based on relationships define lock modes for each class relationship separately. However, lock modes for the combinations of class relationships are not defined. Hence, they are not widely used.

The models based on operations have maximized the concurrency for runtime transactions by using access vectors. They have the following limitations:

1. The concurrency of runtime transactions is maximized at the cost of inconsistency of business data.

2. None of the lock models have addressed the conflicts among design time transactions and conflicts between runtime and design time transactions

3. The existing lock models have not fully exploited the semantics of attributes, methods and class relationships to maximize the concurrency of design time transactions.

4. They have also not proposed fine granularity lock modes for all types of design time operations defined in Bannerjee1987.

5. Design time transactions are executed in coarse granularity which reduces the concurrency and thus reduces the throughput also.

6. Run time transactions and design time transactions that are arriving simultaneously are blocked, even if they access different part of the class diagram. This affects the performance of the system.

So a MGLM has to be proposed which will improve the degree of concurrency for design time transactions and run time transactions by fully utilizing the semantics
of object oriented features. Separate lock modes covering all types of design time operations based on Bannerjee1987 have to be proposed. It should eliminate the inconsistencies in the existing models. An enhanced compatibility matrix has to be defined to provide high parallelism between design time transactions and run time transactions.

Multi granular lock models using access vectors provide high concurrency for domains which do not alter its services frequently (stable domains). Here the design time transactions are rare. These models require access vectors in addition to the compatibility matrix and lock table. But in the case of continuously evolving domains, the schema of the domain needs to be changed frequently to match the new changes in the domain. Then, the access vectors as well as lock table should be altered every time a schema change is made. Because of this, the maintenance overhead is more than the conventional locking technique. This introduced the need for a new concurrency control scheme to support continuously evolving systems with less overhead. This algorithm should provide same or higher concurrency than the existing models with nil or less overhead.

Concurrency control mechanisms provide consistency at the cost of introducing deadlocks. OODS transactions support AND model [Hac1989]. This means that a transaction can execute only when it gets all the resources. If there exist no alternatives for any of the resources and if the resource request model is AND model, then Holt1972 says that the necessary and sufficient condition for deadlock is the presence of a cycle in GWFG. Shaw1974 and Coffman1971 have shown that by resource ordering, deadlocks can be prevented in the single resource model. Then a novel resource ordering technique is to be proposed as a common resource ordering policy cannot be adopted for all the systems. A resource ordering policy exploiting the semantics of class relationships is more suitable. To eliminate starvation, an access ordering policy based on semantics of object oriented features is also to be proposed.

Though the existing distributed deadlock detection algorithms perform well, they are not fault tolerant. They enforce a constraint that the implementation environment is free from hardware, software and network failures. So there is a need for improved fault tolerant distributed deadlock detection algorithm.

Once a deadlock is detected, a victim transaction has to be selected to break the cycle. Selection of an optimal victim that incurs minimum cost dynamically is a NP-
Complete problem [Gary1979]. So a cost based victim selection algorithm has to be proposed which lets the selection of factors based on which the victim can be chosen.

Thus the object of high concurrency, low overhead concurrency control and deadlock handling techniques are to be proposed by exploiting the object oriented semantics of distributed object environments.

1.5 Objectives of the Research Work
- To enhance the existing semantic MGLM for OODBMS by proposing fine granularity for design time operations in stable domains.
- To propose semantic MGLM for OODBMS supporting transactions from continuously evolving domains with high concurrency and low maintenance overhead.
- To propose lock types, lock granularity and compatibility matrix for all class relationships namely inheritance, composition and association for distributed object oriented system.
- To propose a deadlock prevention technique for distributed object oriented systems.
- To propose a fault-informant algorithm to send colored probes to the initiator of deadlock detection about the status of sites.
- To propose a weight based victim selection algorithm for deadlock resolution.

1.6 Organization of Chapters in the Thesis
The thesis is organized as follows.

**Chapter 1** introduces the research problem by briefly describing the concepts of distributed object environment along with the challenges of the problem domain. The motivation and the objectives of the thesis are also formulated and explained.

**Chapter 2** is dedicated to review the related existing works for their merits and demerits. It gives a brief classification of semantic concurrency control techniques and deadlock handling techniques in distributed object environment. The representative algorithms are identified and reported. The limitations in the existing algorithms are summarized. At the end of this chapter, the proposed research work is defined and described.

**Chapter 3** describes an enhanced semantic MGLM for OODBMS supporting stable domains. The performance comparison of the proposed algorithm and the similar existing algorithms are reported in this chapter.
Chapter 4 proposes two semantic concurrency control mechanisms in OODBMS for continuously evolving domains namely semantic MGLM using lock rippling and semantic MGLM using access control lists. They are compared with the existing semantic MGLM.

Chapter 5 proposes semantic concurrency control for OODS by defining the lock types, granularity for all types of object relationships. A compatibility matrix combining all the relationships is also proposed.

Chapter 6 proposes a deadlock prevention technique based on resource ordering for OODS. The existing probe based distributed deadlock detection algorithm is enhanced to work in faulty environments also. A weight based victim selection algorithm is also proposed.

Chapter 7 concludes the thesis by highlighting the findings that facilitated to accomplish the objectives. The limitations of the research work have been identified to carryout the possible future research to make further improvement in this direction.