CHAPTER 1

INTRODUCTION\(^1\)

In this chapter we give a brief overview of the topics that are addressed in the study in the area of distributed and replicated databases.

For the past many years, many business concerns and industrial enterprises have introduced innovations in their data processing, which have come to be gradually supported by on-line systems. These on-line databases are intended to remain up-to-date and correct, at all times. Critical applications and other stringent requirements in many fields such as defence have eventually led to many significant advances in technology [HEV88, SEL87], such as hardware advances [SEL90], communication and networking advances, and software advances. These advances led to proliferation of distributed database system that provides the advantages of enhanced performance, reliability, availability, and modularity to an organization [CER84, OZS91].

Comparatively distributed database systems can be adapted more naturally in the decentralized structure of many organizations and also eliminate many of short comings of centralized database systems. However, distributed database systems are more difficult to design, control, use, and manage than centralized database systems. Thus, a great deal of research and development activity has taken place over the past two decades in order to find

\(^1\) An earlier version of a portion of this chapter has appeared in Proceedings of Indian Computing Congress (ICC'91) [RED91].
effective and efficient methods for designing, controlling, using, and managing distributed database systems. Some of the areas of research include transaction management, database design, directory management, concurrency control and deadlock resolution, and reliability. In this study, we focus on problems of concurrency control and deadlock resolution, and reliability.

In distributed database system, concurrency control mechanisms can be characterized in terms of the control disciplines they employ. One class of mechanisms involves some form of centralized control whereby all transactions pass through a single control point. At that point, the transactions can be validated and then distributed to the various database sites for application to the database copies. The other class of concurrency control mechanisms employ distributed control. For this class the responsibility for validating transactions and applying them to the database copies is distributed among the collection of database sites.

Mechanisms which use centralized control are attractive because a central control point makes it relatively easy to detect and resolve conflicts between transactions. The main disadvantage of such mechanisms is that database update activity must be suspended whenever the central control point is inaccessible. Such inaccessibility could result from failures in the communication network or at the site where the control point resides. Because a distributed concurrency control mechanism has no single point of control, it is possible to process transactions even one or more of the component sites are inaccessible. The problem here is that it is nontrivial to design a distributed update control mechanism, which can resolve conflicting updates in a way that preserves consistency and is deadlock-free. Centralized concurrency control algorithms is adequate for many applications. However,
there are database applications whose transaction performance requirements can be satisfied only by a system which uses distributed control mechanism.

In the subsequent chapters, we address ourselves to two subtopics. These concern concurrency control in distributed (without replication), and replicated environment. In chapter three, we propose a distributed concurrency control algorithm based on local access graphs, which is deadlock free and rejection free. In chapter four, we propose an algorithm to coordinate transactions in replicated database based on transaction data flow graphs. Both the approaches are distributed approaches. The algorithms treat all database sites as more or less equivalent. For example, a transaction can be initiated at any site. They share the same base, as both are graph based approaches.

The problem area of respective algorithms is being introduced in the following sections. Section 1.1, presents the background of the data flow graphs, and their applicability to concurrency control. Section 1.2, deals with the problem of deadlocks in distributed database systems. The motivation to data replication and a problem of concurrency control in replicated databases is discussed in the section 1.3. The objectives of the dissertation are explained in the section 1.4. The organization of the dissertation can be found in the last section.

1.1 Background of data flow graphs

In this section we present the background of graphical representations. In doing so, we investigate the possibility of using these representations to achieve concurrency control in a database system.
In non-programming disciplines, graphical representations are widely applied because they provide a more intuitive view of system structure. The graphical representations are also used in a variety of ways to represent diversified complex models [DEV82, AGE82]. Token models and structure models are two basic approaches for using graphs to represent data flow programs [DAV82].

Data flow programs are represented by directed graphs (called data flow graphs) wherein nodes represent the operations to be performed and arcs the data dependencies between them. Data items are viewed as flowing on the arcs from one node to another in a stream of discrete tokens [AGE82]. The scheduling of operations is constrained by the data dependencies identified by the graph. Operations in the same directed path must be scheduled serially. Operations in different paths can be scheduled independently, and different executions of the same operation acting on different tokens can be scheduled simultaneously. If a data flow graph containing all operations to be executed in a database system could be constructed, a data flow scheduling mechanism could execute operations based upon the dependency constraints defined by the data flow graph. In this way, concurrency control is achieved through graphs.

On the basis of these observations, it appears reasonable to investigate the possibility of how such representations could be useful for concurrency control within database management systems. Precedence graphs are often used to determine serializability and wait-for-graphs are used to detect deadlocks [ULL82]. Acyclic precedence graphs within such indicate serializability, and acyclic wait-for-graphs imply freedom from deadlocks. The tools for serializability theory, transaction management coupled with database program graphs can
provide solutions, that are more efficient than conventional concurrency control techniques, such as, the two-phase locking.

Previous research activities have investigated the possibility of applying data flow concepts to data processing. The MEDLEY distributed DBMS project proposed a data flow approach to the scheduling of database operations in a distributed database environment [VIN80]. Recognizing that the query expression (parse) tree [MAI83] representation of database queries represents a type of data flow graph, Vines proposes decomposition of a single query into subgraphs based upon the location of data being accessed. These subgraphs are then executed using data flow scheduling mechanism at the various sites. Another approach to the merging of data flow and database concepts has been investigated in [BIC85]. With this approach, a database is stored in a network where each node is capable of processing database operations on the data at that node. The structure of the network is based upon the Entity-Relationship model [CHE76]. The network can be viewed as a data flow graph, and queries as a tokens being passed through the graphs. As queries are processed, the query tokens propagate through the network obtaining any results satisfying the query.

The application of graphs for concurrency control already have been studied by Eich [EIC88, EIC90, EIC88a] and, Katoh [KAT85] for centralized database system. The algorithm is explained in detail in chapter two. Data flow graphs representing individual transactions are merged together to form a data flow graph representing all active transactions and conflicting operations. The technique requires less information per transaction than does locking, and most of the processing overhead is incurred before and after transaction execution, rather than during processing. In systems that precompile and save commonly used queries, much of the
concurrency control overhead can be performed during precompilation of individual queries. A pre/post processor is used to maintain data flow graph as transactions enter and leave the system. This pre/post processing can be performed in parallel with transaction processing, thus reducing the overhead needed during transaction processing.

In this study, by applying data flow graphs, we have designed concurrency control techniques for distributed and replicated database environment.

In the next section, we give a brief introduction to the concurrency control and deadlock problem in the distributed databases. After that, we describe the problem of failures in distributed database system, and motivation to data replication.

1.2. Deadlock problem in distributed databases

The problem of deadlocks is examined in this section. We survey the existing deadlock detection strategies. At the end of this section, we give the overheads posed by deadlock detection mechanisms.

1.2.1. Locking and deadlock

In a distributed database system, several users may simultaneously make attempts to read and update a set of data items. In order to always provide users with a consistent view of data, it is necessary to have concurrency control. The concurrency control algorithm ensures that the schedule produced by a set of concurrent transactions will be serializable.

The research has led to the development of many concurrency control algorithms. Most of the proposed concurrency control algorithms is based on one of the following
mechanisms: locking, time-stamps [BER81, KOH81], commit-time validation (also called optimistic concurrency control or certification [KUN81]), and single version or multiple versions [CHA85, BER83].

In a distributed database system (where each copy of data is stored at one site only), two-phase locking is the most widely used algorithm for concurrency control. The principle characteristic of two-phase locking protocol is that all locks are acquired before a transaction starts releasing any locks. The deadlock problem is intrinsic to a distributed database system which employs locking as its concurrency control algorithm. A deadlock occurs when one transaction waits for locks held by another transaction which is waiting, directly or indirectly, for locks held by the first transaction. The following example illustrates a simple deadlock.

Example 1.1: Consider the two site distributed database system, where data items X and Y resident at sites S1 and S2 respectively. Assume that, transactions T1 and T2 arrived at sites S1 and S2 respectively. Both transactions need locks (read and write) on both data items before starting execution. Transaction T1 locks data item X at S1, and sends the lock request for data item Y. Now if T2 locks data items Y and sends the lock request to data item X, there will be a deadlock. This is because, T1 will be blocked waiting for T2 to release its lock on data item Y while T2 will be waiting for T1 to release its lock on data item X. In this case, the two transactions T1 and T2 wait indefinitely for each other to release their respective locks. Figure 1.1, depicts the situation.
Locking algorithms can be divided into two classes: static and dynamic locking. Static locking is one where all lock requests of a transaction can be sent either simultaneously or serially to sites where data items reside, and execution may not start until all requests are granted. In a distributed database system, where transmission delay may be substantial, static locking schemes are preferable to dynamic ones. This is because static locking schemes allow concurrent transmissions of different lock requests, thus improving the response time of a transaction. A number of deadlock detection algorithms have been proposed for static locking [CHA82, CHA83, GLI80, MEN79, OBE82]. Two excellent surveys are available on this topic [ELM86, KNA87]. Dynamic locking is one where a lock request is issued, each time a particular data item is needed. In this dissertation we consider an approach where all accesses to the data items are granted prior to the start of the transaction.
1.2.2. Existing approaches

Detection of a deadlock is very difficult process in a distributed database system because no controller has complete and current information about the system and data dependencies. As in [KNA87], we classify existing deadlock detection algorithms by the techniques used. The first class of algorithms maintains an explicit global wait-for graph (WFG). The basic idea is to build some form of global WFG at each site. Each site sends its local WFG to a number of neighboring sites. The local WFG is updated and then passed to neighboring sites. This procedure is repeated until some site has sufficient information of the global situation to claim there is a deadlock. The common feature of this algorithms, namely, sending global WFG along paths to neighboring sites, leads to the name path pushing algorithms. Algorithm belonging to this class are described in [HO82, MEN79, OBE82].

The second class of the algorithms are edge chasing algorithms. The deadlock cycle in a distributed system can be detected by propagating special messages called probes along the edges of the cycle. Probes may be initiated when some conditions are satisfied. When a initiator of a probe receives a matching probe, it knows it is on a deadlock cycle. Approaches of this type are described in [CHA83, CHA82, MEN79, ROE88, SIN85]. A variation of this method is presented in [MIC84], where probes are sent in the opposite direction of the wait-for edge. The third class is characterized by diffusing computation. The basic idea is that a diffusing computation is activated by a transaction site which suspects a deadlock. The diffusing computation grows by sending query messages and shrinks by receiving replies. When a diffusing computation shrinks back to its root, it terminates. If the diffusing computation terminates, the initiator declares a deadlock. The characteristic feature of
diffusing computation is that the global WFG is implicitly reflected in the structure of the computation, but never built explicitly. Algorithms using diffusing computations are described in [CHA82, DIJ80]. In the algorithm proposed in [SHY90], the deadlocks are eliminated by reordering the lock requests of the data items, in such a way that no transaction needs to be aborted. The algorithm must be run regularly to detect deadlocks.

Almost all the resolution protocols associated with existing deadlock detection algorithms abort (restart) some transaction in the deadlock cycle detected. The restarted transaction must release all its locks, and send out the requests again. A large amount of information needs to be propagated from site to site [OBE82]. This may not be good idea if the transaction accesses many data items, since it may be very costly for the transaction to collect all these lock grants again when it is restarted. In addition, abortion is not necessary if none of the transactions in the deadlock cycle has started execution. As was shown in [SHY90], deadlocks ultimately degrade database performance dramatically. Although deadlocks occur rarely, however, if a deadlock occurs, the number of blocked transactions increases dramatically. Till the invocation of the next deadlock detection mechanism, propagation of enormous delays would continue. This implies that deadlocks should be detected and resolved immediately, though they do not occur very often. Based on the above observations, we design an algorithm, in which deadlocks are resolved instantly without abortions.

1.3. Failures and data replication

The problem of failures in a distributed database system is examined in this section.
This forms the motivation to adopt data replication and specialized concurrency control in replicated databases.

1.3.1. Failures problem in distributed databases

In case of a system failure at one of the sites, the processing comes to an end abruptly and the contents of volatile storage are destroyed at once. It can be due to hardware failure (i.e., processor, main memory, power supply, etc.) or due to a software failure (bug in the operating system or in the database management system code). As a result of such failure, a part of the database that was stored in main memory buffers is lost.

Another type of failures are the media failures. These refer to the failures of the secondary storage devices that contain the database. Operating system errors, as well as hardware faults such as head crashes or controller failures could cause media failures. In the event of media failure, all or a part of the database that is on the secondary storage is considered to be destroyed and becomes inaccessible for ever.

Elimination of failures that are related to hardware as well as software, can not be accomplished with the kind of technology available today. But, improving design and testing would reduce these failures to a certain extent. The main principle employed in all fault tolerant system designs is that of providing redundancy in system components. Data is maintained in two mirror disks that are located in close proximity to one another. Data is written into both copies, and read from any one. While this approach is very fast, it has certain deficiencies. The special purpose hardware is expensive and availability is limited by the fact that a single physical catastrophe can destroy the whole system.
Apart from these, failures do occur due to natural calamities like flood or fire; it may occur due to malicious social acts too. Eventually all these can cause an extensive damage. If the failure is due to environmental reasons, then maintenance of reliability of a data against such type of failures becomes a complex problem.

1.3.2. Motivation to data replication

The most important goal of networking is to provide high reliability by having alternate sources of supply. All files could be duplicated on two or three machines, so that, if one of them is not available (due to a hardware failures), the other copies can be used. In addition, the presence of multiple central processing units means that if one of them goes out of operation, the others may be able to take over its work. The ability to operate continuously in the face of hardware problems is of great importance especially for the sectors like defence, banking, and air traffic control.

An approach to design of highly available system is to store copies of data at several different geographical locations (sites) in a loosely coupled distributed computer system with independent failure modes. By storing the copies of data on processors, the probability that at least one copy of the data will be accessible, increases. Supporting failure free operations, requires maintenance of copies of data at more than one site. This approach is known as the data replication. The main advantage of distributed database system is that, possibility of supporting data replication as well as data distribution. In the same light, a specific area of new advance is that of replicated database management systems (RDMSs), that offers potential benefits, such as reliability, increased availability and growth possibility [GAR82].
1.3.3. Advantages of replication

In the presence of system failure, critical data items can be accessed from more than one copy, allowing the system to maintain reliable services while the failure is corrected.

In many cases, where the majority of accesses to the data items are read accesses only, the sites can process such queries in parallel. This results in improved performance [BER84, STO79]. By storing the copies of data items at various sites, the need for (expensive) remote read access is reduced. Hence data replication minimizes data accessing overhead. Data copies are located at sites [MIL89, CHU92], in such a way that it provides acceptable levels of availability to all system sites. Thus, multiple applications can access different copies of the same data simultaneously, improving system performance.

1.3.4. Problems and issues of replication

As discussed above replication enhances the reliability and data availability for read only access. But on the other hand, increased number of data copies imposes considerable overhead for processing the update transactions. In a RDMS, where several users concurrently update the data items, operations from different transactions may need to be interleaved and allowed to operate concurrently. In such cases, the interleaved execution of read and write operations of transactions may produce incorrect results.

In RDMS, notion of database consistency has two aspects. The principal goal of replication control mechanism is, to guarantee that updates are applied to copies of replicated data in a way, that preserves the mutual consistency of the collection of database copies,
as well as the internal consistency of each database [DEV85].

Mutual consistency requires that all copies of the database must be identical. This means that, whenever a data item is updated, it must be propagated to all sites containing the replicas. For example, in a banking system where account information is replicated in various sites, it is necessary that transaction management procedures must ensure that, the balance in a particular account agrees at all sites to a common value.

The internal consistency requires that each copy of the database remain consistent within itself. It concerns the preservation of invariant relations that exist among data items within a database.

Concurrency control techniques [BER81] are adopted for coordinating the concurrent access to the database in order to provide the effect that each request is executed as per requirements of the criteria of serializability [BER87]. The criteria for serializability in replicated databases is discussed in the following chapter.

In a distributed system, a combination of site and link failures can disable the communication between two groups of sites. This phenomenon is called network partitioning. In this situation, each pair of sites within a group can communicate with each other, but sites in a different groups can not communicate. Unless partition failures are detected and recognized by all effected sites, an independent and uncoordinated update may occur at a copy of data, thereby resulting in an inconsistency of database [BER87].

The task of transaction management leads to adoption of concurrency control, crash recovery and network partition handling in a replicated database environment.
1.3.5 Existing approaches

In recent years, there have been several studies which have come out with suitable techniques for storing and retrieving data reliably through replication, and a number of concurrency control schemes have been proposed [BER87, SON87]. Some of these proposals are based on voting based approach [THO79, GIF79], the primary copy approach [STO79], the locking based approach [BER84], use of tokens [MIN82], the semantics based approach [KUM88], and the update transport approach [SIN90] for processing update requests. The basic requirement for any replica control algorithm is that, databases should (mutually and internally) remain consistent. Besides this, the algorithm should be robust with respect to site and partition failures.

Replication management techniques can be divided as centralized and distributed. In centralized control techniques, all requests on replicated data objects pass through a single control point, at which the requests are validated first, and then distributed to all the sites that maintain replicated copies. In distributed control approach, the responsibility for validating requests and actual execution of these requests on the replicated copies is distributed among the collection of sites. The disadvantage of centralized techniques is the vulnerability of the system. The database activity must be suspended whenever the central control point is inaccessible due to the failure of the site where the control point resides.

Distributed control techniques achieve higher reliability than centralized control techniques.

Among the existing techniques, voting [THO79, GIF79] and special copy [STO79] are the major techniques for coordinating transactions in replicated data environment. In this
study, we concentrate on voting based approaches.

The first voting based approach is majority consensus approach [THO79] in which, a transaction can commit if a majority of sites give their consent. Each copy of the data item carries the time-stamp of the last transaction which updated it. The algorithm employs the time-stamp ordering technique both in the voting procedure and in the application of the accepted requests to the replicated copies. The detailed version of the quorum consensus algorithm is explained in chapter four.

Since each transaction must obtain O.K. from majority of sites, no two conflicting transactions can be executed and committed. Hence, if there is a conflict among a group of transactions, then one of the transactions is accepted and remaining transactions are rejected. These rejected transactions are resubmitted again. As the number of conflicts increases, the number of rejected transactions, also shows on increase.

1.4. Objectives of the study

The primary objective of this dissertation is to explore the data flow graph techniques for concurrency control in distributed and replicated database environment. In pursuing this objective we achieve the following accomplishments.

We extend the data flow graphs to order the transactions in distributed database system which is an alternative to distributed (static) two-phase locking. It forms local access graphs at data sites for each transaction. Transactions are ordered serially, as these arrive with the help of local access graphs. This algorithm does not rejects the transactions due to deadlocks. The deadlocks are removed instantly. We need not run separate deadlock
detection and resolution algorithms as the proposed algorithm itself handles the deadlocks without rejecting the transactions. It avoids resubmissions of transactions and hence reduces processing overheads within the distributed environment.

Apart from this, we also extend the data flow graphs to order the transactions in replicated database environment, which is an alternative to quorum consensus algorithm. In chapter four, we have considered a distributed system approach for generation of a transaction data flow graph for each submitted transaction. As said earlier, in case of conflicts, the voting based approaches reject transactions. In the proposed approach, the transaction request visits majority and forms transaction data flow graph, which contains all conflicting transactions. The transactions are ordered through these graphs. The possibility of a transaction rejection, or abort is removed.

1.5 Organization of the study

Chapter two, deals with the distributed system model, serializability theory, and the concept of data flow graphs. Serializability criterion for distributed and replicated databases is presented. Also, notion of time-stamps and transaction numbers is presented. A centralized concurrency control algorithm based on database flow graphs is described at the end of the chapter. A concurrency control algorithm for distributed database system based on local access graphs is proposed in chapter three. Simulation results were presented at the end of the chapter. Chapter four, presents the replica control algorithm based on transaction data flow graphs. The two-phase commit protocol which is to be employed to incorporate the updates into the database is presented in chapter five. It also explores the possibility of
integration of the commit protocols with the proposed algorithms. And, the last chapter gives the summary and conclusions. In this, we present some of the positive remarks of the proposed algorithms, and also give directions for future work.