数理解析研究所講究録
625巻 1987年 186-195

A Two-Phase Locking Mechanism Avoiding Deadlock for Read-Write Confliction

九州大学　仲　興国　(Xingguo Zhong)
九州大学　上林 彌彦　(Yahiko Kambayashi)

2-phase locking method is the most widely used approach for concurrency control in database systems. To detect deadlocks that may occur by mechanisms based on 2-phase locking protocol, a wait-for graph is usually used. By rolling back some transactions, the cycle corresponding to a deadlock can be removed. To improve system efficiency, it is very important to avoid deadlocks or to decrease the number of deadlock. The conflictions that may cause deadlocks can be classified into read-write, write-write and write-read situations. In this paper, we point out that there is functional redundancy between 2-phase locking protocol and a wait-for graph, and propose a mechanism that can always avoid deadlocks in case of read-write conflictions. That is, in our mechanism a read-lock request never cause deadlock. This property is very similar to some multi-version timestamp ordering mechanisms. In contrast to timestamp ordering mechanisms, no multi-version of data is necessary in this mechanism. Since for almost all database systems write-write conflictions are very rare to occur, it is very important for a concurrency control mechanism to make the read-write conflictions to be deadlock free. Combination of this mechanism with 2-phase commitment protocol seems to be very practical.

1. INTRODUCTION

In recent years, a lot of papers on concurrency control and recovery problems have been published and there are still many researchers on the subject. It has gradually become clear that most of the results can be classified into two kinds of approaches : two-phase locking [EG76] or timestamp ordering [RE78, BG80]. There are other methods, such as those realized by mixing the two approaches [BG81] and algorithms that depend on that the database itself have some data structures [SK80, SK82, MF85], but the general approaches classified in essence are only the two. It is well-known that each approach has different advantages and disadvantages. For example, multi-version timestamp ordering methods have the merit that read operations never get into trouble by reading a proper version of a data item [BG80]. However, there is very little research on methods for applying the advantages of one approach to the other.

2-phase locking based methods are most widely used for concurrency control in database systems. To solve the deadlock problem that exists in 2-phase locking methods, a wait-for graph is usually used [GR78]. By rolling back some transactions, the deadlock cycles can be removed. Timestamp ordering based methods have been proposed mainly for distributed database systems[BG80], since deadlock detection in 2-phase locking methods is very hard and cost consuming in distributed environments [MM79,GS80, OB82]. We will not discuss it in detail in this paper.

To improve system efficiency, it is very important to avoid deadlocks or to reduce the rate of deadlocks as less as possible. The conflicting situations of operation requests that may cause rollback of transactions can be classified into read-write, write-write and write-read. A read-write confliction means that a transaction requests a read operation on a data item that is written or held for writing by another transaction. Under 2-phase locking method, a read-write confliction occurs when a transaction requests a read lock on a data item that is locked by another transaction in write mode. Evidently, a write-write (write-read) confliction occurs when a transaction requests a write lock on a data item that is locked by another transaction in write mode (by other transactions in read mode). In this paper, we point out that there is functional redundancy between 2-phase locking protocol and a wait-for graph, and propose a mechanism that can always avoid deadlocks in case of read-write conflictions. That is, in our method a read lock request never causes deadlock, and thus it does not cause rollback of transactions. This property is just like the advantage of typical multi-version timestamp ordering mechanisms. In contrast with timestamp ordering mechanisms, no multi-version of data is necessary in this mechanism. Since for almost all database systems write-write conflictions are very rare to occur, it is, therefore,very important for a concurrency control mechanism to make read-write conflictions to be deadlock free. This mechanism is joined with 2-phase commitment protocol [GR78] in order to make it practical.

In the rest of this paper, we first give the basic concepts on concurrency control and 2-phase locking in Section 2. In Section 3, we point out the problem that exists in conventional methods and give a framework of the ideas proposed in this paper. We describe our mechanism in Section 4. Additional discussions are given in Section 5.

2. BASIC CONCEPTS

(1) Serializability

Concurrency control is very important in realizing database management systems. The most general way to discuss the concurrency control problem is to view a database as a set of data items, on which read and write operations are performed by programs. To ensure the consistency of the database, a common approach is to define transactions as the units of operations that preserve the consistency of the database [EG76, GR81]. A transaction always transforms a consistent state of the database into a consistent state when it is executed alone. The outcome of processing a set of transactions concurrently is required to be same as one produced by running these transactions serially in some order. A schedule that has this property is said to be serializable [EG76, BS79]. The basic work of concurrency control for database systems is to ensure the serializability of the schedule of transactions. In order to guarantee the serializability, some started transactions have to be aborted in some situations. This is called rollback of transactions. When a transaction is rolled back, all the changes that are caused by the execution of that transaction must be recovered.

(2) 2-phase commitment protocol

2-phase commitment protocol was first introduced by J. Gray for distributed database systems [GR78]. In fact, the same problem also exists in centralized systems,

if we do not have the assumption that the system never gets into failures and we do not allow any cascading of rollback of transactions.

By 2-phase commitment protocol, write operations of a transaction cannot reflect the new values that it writes to the database before it has been ensured to complete. This is realized by dividing the commitment of a transaction into two phases. In the first phase, the transaction writes the new values to logs without losing the old values of the corresponding data items in the database. After the transaction has completed all its write operations to logs, the completion of the transaction is guaranteed. In the second phase it reflects the logs to database.

(3) 2-phase locking protocol

2-phase locking protocol is a well known method in guaranteeing the serializability of a schedule of transactions. Before performing an operation on a data item, the transaction first locks the data item. There are read locks and write locks corresponding to read and write operations. More than one read lock can be imposed on the same data item at the same time. However, when a data item is locked by a transaction in write mode, no other transactions can lock it. The execution of a transaction is always divided into two phases. In the first phase, it does only lock data items and in the second phase it does only release locks. That is, for each transaction once it has released a data item, it will not lock any data item further. Under this protocol, the schedule of a set of executed transactions (or including the executing transactions up to their executed stages) is always serializable [EG76].

(4) Deadlock

The deadlock problem exists in the two-phase locking protocol. When transaction T_i requests to lock a data item D that is still locked by another transaction T_j (and at least one of them is a write lock), T_i can not obtain the lock and have to wait until T_j releases D. This situation is called a confliction, and we call T_i the conflicting transaction and T_j the conflicted transaction. When T_i is determined to wait for T_j, we say T_i is dependent on T_j and denote as $T_i \rightarrow T_j$. In this case, we call T_i the depending transaction and T_j the depended transaction. Furthermore, if $T_1 \rightarrow T_2$, $T_2 \rightarrow T_3$ . . . $T_{k-1} \rightarrow T_k$, T_1 is also dependent on T_k. Each transaction may be in one of two states. One is in the active state which means the transaction is in execution. The other is in the blocked state which means the transaction is blocked due to some lock request. Since a transaction can be waiting for a data item while holding other data items, the wait-for cycle as $T_1 \rightarrow T_2$, $T_2 \rightarrow T_3$ . . . $T_k \rightarrow T_1$ may occur. This situation is called deadlock. When a deadlock occurs, all the transactions in the wait-for cycle are blocked and cannot be executed further. The system must have some way to detect and resolve deadlocks [GR78].

(5) Wait-for Graph(WFG)

In order to detect deadlock, a directed graph called the wait-for graph(WFG for short) is usually used by the system. Each node of the graph corresponds to a transaction issued to the system. For simplicity, we use the same notation of transaction to the corresponding node in WFG. When transaction T_i conflicts with transaction T_j, that is when $T_i \rightarrow T_j$, an arc from node Ti to node Tj is appended to the WFG. There is a deadlock in the system iff there exists a cycle in the WFG [EG76]. There are mainly two strategies in handling deadlock detection. One is called

continuous detection and the other is called periodic detection. In continuous detection, a detection is performed whenever a new arc is required to be added to the WFG. In periodic detection, the detection is performed once in a period of time. The strategy used in this paper is continuous detection.

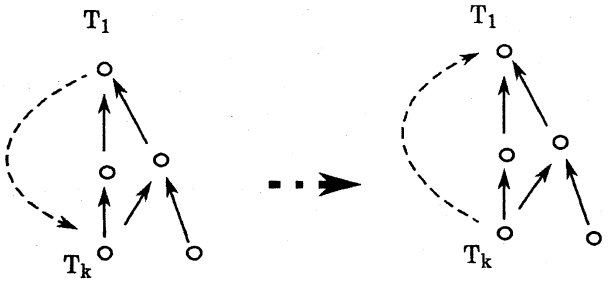
3. THE PROBLEM

We have introduced the basic concepts of the 2-phase locking protocol in the previous section. In order to give the reader a framework of the ideas proposed in this paper, we first describe the situation where a read lock is being requested by a transaction to see how we can handle it to be deadlock free without using multi-version of data.

The confliction caused by a read lock request only arose on the situation when a transaction T_1 requests to lock data item D in read mode that has been locked by another transaction T_2 in write mode. If T_2 does not depend on T_1 then no cycle in the WFG will be created when we append a new arc from node T_1 to T_2. Transaction T_1 can wait until T_2 releases D. However, when transaction T_2 depends on T_1, a cycle $T_2 \rightarrow \ldots, T_1 \rightarrow T_2$ will be created if we append the arc to the WFG. In such a situation, a deadlock occurs by conventional methods. Transaction T_1 or T_2, or maybe other transactions in the cycles have to be rolled back to break the deadlock.

To our viewpoint, when such a situation occurs, we need not to rollback any transaction. Since T_2 depends on T_1, T_2 must be in blocked state for some lock request, therefore T_2 has not yet reflected the new value of D being requested by T_1 to database (See 2-phase commitment protocol). Thus we can change $T_1 \rightarrow T_2$ to $T_2 \rightarrow T_1$ in the WFG. That is, transaction T_1 can read the existing value of D directly without destroying the serializability of the schedule of transactions.

Fig. 1(a) gives an example of the situation where transaction T_1 is requesting a



(a) An example of read-write confliction that causes a deadlock

(b) An example of avoiding deadlock caused by read-write confliction

Fig. 1

read lock that conflicts with a write lock of transaction T_k on D. By consenting T_1 to read the existing value of D, the deadlock can be avoided. The changed WFG is described in Fig.1(b). The dotted arcs in the figure express the dependency that is produced by the above operation.

Now we discuss the problem of conventional 2-phase locking methods. That is, there is functional redundancy between 2-phase locking protocol and the wait-for graph. In continuous strategy, no cycle is allowed to exist in the WFG. That means the schedules of transactions are always serializable up to the executing stage of each transactions. For this reason, it is not necessary to lock data items, in the conventional meaning, for a concurrency control mechanism. In conventional locking, when a data item is locked by a transaction, no other transaction can obtain a conflict lock on the data item. Therefore, the transaction that requested the data item earlier must perform its operation earlier than later transactions. Reading the existing value of data items as described above, shows that even if transaction T_1 requests its read lock later than T_2, it can do its reading before the writing of T_2.

With the development of database systems, the concepts of consistency and reliability are more strictly required. 2-phase locking protocol joined with 2-phase commitment have gradually become a standard technique in transaction processing. Under this environment, the period of a lock becomes long and a lock in write mode does not mean having a write operation on a data item , but means that the transaction will write the data item in the future when it is committed.

Thus we conclude that there is no sufficient reason to have locking in the conventional meaning on data items for concurrency control in database systems. It is more reasonable only to record proper information on a data item before a transaction read or determined to write it in the future. In this paper, we still use the word "lock" for succession, but the essence of the concept is different from the conventional one. Several lock modes on data items are defined in following.

4. A CONTROL MECHANISM THAT AVOIDS DEADLOCK IN CASE OF READ-WRITE CONFLICTIONS

In this section, we propose the control mechanism that always handles read-write conflictions to be deadlock free as described in Section 3. We first give a general processing model for transactions in Section 4.1. In Section 4.2, a lock mechanism is described by defining the lock modes on data items. The management of the WFG that does relate to the lock mechanism will be described in Section 4.3.

4.1 A processing model for transaction and data item operations

A transaction performs read and write operations on data items when it executes. When a transaction starts its execution, it is given a private work space by the system for buffering the data items it will read and write. All the read operations are performed by copying data items to its work space. Therefore, a transaction will not read same data item more than once. By 2-phase commitment protocol, write operations of a transaction always do not reflect their values until the transaction is committed. Therefore, under the above supposition no data item will be written to the database more than once by one transaction.

There are read lock requests and write lock requests that must be performed by a transaction before it wants to read and write a data item respectively. Once a transaction is blocked, it is not desirable for the transaction to lock other data items

further, since the increase of the number of data item which is held by a blocked transaction causes the increase of possibility of deadlocks in the system. Therefore even if a transaction could execute its actions in parallel, the lock requests are performed serially. In this way, a transaction can be blocked on only one data item even if several outgoing arcs might be created when a write lock request conflicts to several transactions that locked the same data item in read mode. When a transaction do both read and write operations on same data item, two distinct locks are performed. However, two locks by the same transaction are not judged to be a confliction. It is important to note that a write lock request is different from its write operation. The write operation can only be performed at the time when the transaction is committed. Write lock request, however, should be issued to the system as early as possible (see Section 6).

4.2 The lock mechanism

Under our lock mechanism below, a read lock request on data item D means that the transaction requests to read D. Once the read lock is granted, the transaction will read D soon. That is, when a write lock request conflicts with a read lock on D, we can think that the conflicted transaction has read D. A write lock request on D means that the transaction requests to write D in future. The write will be performed when the transaction commits. That is, when a lock request conflicts with a write lock on D, we cannot think that the conflicted transaction has written D. We define four lock modes, (a) Read lock, (b) Consent read lock, (c) Write lock and (d) reservation write lock in the following. We suppose that transaction T_i is now requesting a lock on D.

(a) Read lock : Transaction T_i can lock data item D in read mode when D is neither locked in write mode as (c) and nor preserved to be locked as (d) by any other transaction in write mode. When the read lock is granted,the transaction T_i can read D.

(b) Consent read lock : Data item D is still locked by a transaction T_j in write mode as (c) or reserved a write lock as (d). By referencing the WFG, we know that it will create a cycle in the WFG if we add an arc from node T_i to T_j. That is , making T_i waits for T_j will cause a deadlock. In this situation, a consent read lock is performed, since the existing version of D could be read. When the consent read lock is granted, transaction T_i can read D. The only difference between read lock and consent read lock is the current state of the data item when a transaction requests to lock it. After the consent read lock has been granted, there is no distinction between consent read lock and read lock.

(c) Write lock : Transaction T_i can lock data item D in write mode when D is not locked by any other transaction in any mode. When the write lock is granted, transaction T_i is determined to write D when T_i is committed.

(d) Reservation write lock : Data item D is only locked by transactions T_{jk} ($k=1, 2, \ldots n$) in read mode and by referencing the WFG we know that it will not cause any cycle in the WFG if we add arcs from node T_i to each T_{jk} ($k=1, 2, \ldots n$). That is, making T_i wait for T_{jk} ($k=1, \ldots n$) will not cause any deadlock. In this situation, transaction T_i can reserve a write lock. However, even having reserved the write lock, the transaction is not allowed to write D yet. The transaction should change its lock into write mode as in (c) when it is awaken to continue its execution.

A lock request is said to be granted when it belongs to the conditions of (a), (b) and (c). Otherwise it is said to be rejected. Read lock cannot be granted in other situation except (a) and (b). The transaction will be blocked until data item D is released. Write lock described in (c) is defined as conventional way. Reserving write lock as (d) is defined to avoid new read locks of transactions on D for the uniformity of response of transactions.

In this lock mechanism, it is allowed to have several locks on data item D. But there is at most one of them being in write mode or reserving write mode. Fig. 2 describes the lock states of data item D except the situation that there is no lock on D. Small circles express read locks and big circles represent reserving write locks. The arrows from write lock to a read lock show that the transaction locked D in write mode or reserving write mode depends on all the other transactions that lock D in read mode. We distinguish write lock from reserving write lock by describing them with real circles and dotted circle respectively. But there is no distinction between read lock and consent read lock in Fig 2. Condition (a) expresses that there are three transactions locking D. Condition (b) expresses that D is locked by a transaction in write mode. Condition (c) is such a state where several transactions locks D in read mode and one transaction reserves a write lock. This state must have be generated from state of condition (a), in which some of the read locks may be issued later than the reserving write lock. Condition (d) is the state that a transaction locked D in write mode and several read lock requests of transactions are consented to lock D. This state must generated from state of condition (b).

4.3 Management of WFG

The construction of the WFG is described in Section 2. As in the conventional way, the WFG is constructed by using nodes to express transactions and arcs to the dependent relationships between transactions. In this Section, we describe an algorithm for managing the WFG under the lock mechanism described in Section 4.2. The algorithm will be described by enumerating all the possible situations of lock requests and the corresponding operations on the WFG.

Algorithm :

1) A new node T_i is added to the WFG when transaction T_i is issued to the system.
2) When transaction T_i is committed, node T_i and arcs pointing to node T_i are deleted from the WFG .
3) No new arc is added when a lock request of T_i is granted as in the situation (a) and (c) in Section 4.2.
4) When a read lock request of T_i conflicts with a write lock or a reservation write lock of T_j on data item D, an arc from T_i to T_j is added, if no deadlock occurs. Otherwise, a consent read lock will be performed as situation (b) in Section 4.2 and an arc from node T_j to T_i is added.
5) When a write lock request of T_i conflicts to one or several read lock of transactions T_{jk} ($k=1, 2, \ldots n$) on data item D, the arcs from T_i to all of nodes T_{jk} ($k=1, 2, \ldots n$) are added if no deadlock occur.
6) When a write lock request of T_i conflicts with another write lock of T_j on data item D (at the same time, some other transactions may lock data item D in read mode also), an arc from T_i to T_j is added if no deadlock occur.

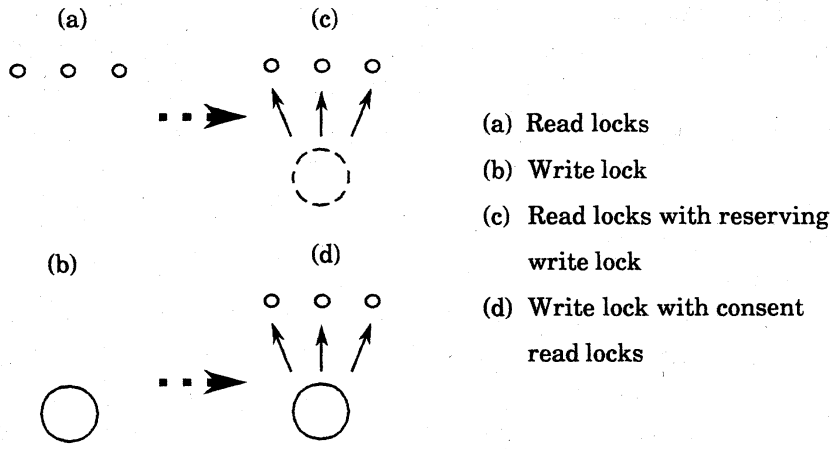


Fig.2 Lock states of data D

7) For the situations of 5) and 6), if a deadlock occurs then transaction T_i or other transactions will be determined to be rolled back in order to destroy the deadlock cycle as in the conventional way.
8) When transaction T_i is rolled back, node T_i and all arcs connected to T_i are deleted from the WFG and all the locks requested by T_i are released from each data item.

In this algorithm, the WFG is managed as a completed description of dependent relationships of transactions. When a read lock request of transaction T_i conflicts with a write lock of transaction T_j on data item D and adding an arc from T_i to T_j will not cause a cycle in the WFG, we cannot let the read lock request of T_i to be consented, since we don't know if it is better to execute T_i before T_j or not. Furthermore, since T_j may be in active state, it is possible for T_i and T_j to operate on D simultaneously. A transaction usually issues its read lock request earlier than its write lock requests. When the above conditions occur, the probability that T_i should execute after T_j is greater than execute before it.

5. FURTHER DISCUSSIONS

We have described that conflictions are classified to be read-write, write-write and write-read situations. In our mechanism we have avoided deadlocks in the case of read-write confliction. Thus we have decreased the rate of causing deadlock.

For a database system, if the read-set of each transaction always covers its write-set then it is impossible for write-write confliction to cause a deadlock. It is even impossible to cause a write-write confliction when there is no duplication of data in the database. We give a proof of these assertions in the following.

We first describe the condition of a non-duplicated database. We can suppose that for each data item that is both read and written by the same transactions is always locked in read mode before the write lock is requested. Since the read-set of each transaction covers its write-set, before a write-write confliction occurs there must be a write-read confliction on the same data item. If the write-read confliction causes a deadlock and one of the transactions is rolled back, then the following write-write confliction will not appear at all. If it does not cause any deadlock, or even caused a

deadlock but the deadlock is broken by rolling back other transactions in the deadlock cycle, the write-read confliction then has determined the dependent relationship of the two transactions. The depending transaction is blocked until the depended one releases its lock on that data item. Thus the depending transaction cannot issue its write lock.

Now we consider the condition of a database with data duplications. We also suppose that read-set covers write-set for each transactions. A read lock request should lock one copy but write lock request locks all copies of the data item [TH79]. In such an environment, there may be write-write conflictions, but if a write-write confliction on some copy causes a deadlock, there must be a write-read confliction on one copy of the same data item causing the same deadlock (same cycle on WFG). It is because that the conflicted transaction (in the write-write confliction) still has locked some copy of the data item before its write lock is performed.

There might also exist some database systems in which transactions can update data item without referencing the old value of the same data item. That is, the read-set of a transaction does not cover its write-set. In such a situation, there may exist write-write conflictions that cause deadlock. How to make write-write conflictions to be deadlock free is also an interesting topic.

The improvement of performance can also be explained by the higher concurrency of data utilization. In conventional mechanisms when a data item is locked by a transaction T_i in write mode, no other transaction can use that data item even if the transactions can and should read it

There is an algorithm in which all transactions always make read lock on data items and does not upgrade its lock mode into write mode until it wants to perform the write operations. That algorithm gets lower performance than that locking data items in write mode directly when the transaction determines to write that data item. This assertion has been proved by a simulation experiment [CS84]. This result can also be explained in the following. Taking write lock to be delayed will increase the probability of deadlock. Since the delay of write lock allows other read lock requests of transactions issued lately to be granted, when the upstaging comes it will conflict to these read locks (write-read confliction) and might cause deadlocks.

6. CONCLUSION

As a new version of 2-phase locking methods, we proposed a mechanism in which deadlock caused by read-write confliction can always be avoided without using multi-version of data. This mechanism can be referred to as one in which the advantages of timestamp ordering methods are introduced to 2-phase locking. We have pointed out that there exists functional redundancy in the 2-phase locking protocol and the wait-for graph. Therefore, in a 2-phase locking mechanism with strategy of continuous deadlock detection, locks in conventional meanings should be changed to record proper information on data items to indicate that the data items was operated on or will be operated on in future. The information on each data item is just like a schedule of operations in contrast to the history of data item in timestamp ordering methods.

One problem is whether we can extend the mechanism proposed in this paper to distributed database systems. Many algorithms for distributed deadlock detection have been proposed in recent years. It is still far from knowing that which strategy will be a suitable one. Reviewing the published algorithms for distributed deadlock detections based on the wait-for graph, we find that about half of them belong to continuous detection. We hope that the proposed mechanism can be extended to distributed environment.

REFERENCES

[BG80] Bernstein,P. A. and Goodman,N. Timestamp-Based Algorithms for Concurrency Control in Distributed Database Systems. Proc. of VLDB. (Oct. 1980). pp. 285-300.

[BG81] P. A. Bernstein and N. Goodman, Concurrency Control in Distributed Database Systems. Computing Surveys, Vol.13, No. 2 (june 1981).

[BS79] Bernstein,P. A. Aan Shipman,D. W. Formal Aspect of Serializability in Database Concurrency Control. IEEE Trans. on Software Eng. Vol.SE-5, No.3 May 1979, pp. 203-216.

[CS84] M. Carey and M. Stonebreaker, The Perormance of Concurrency Control Algorithms for Database Management Systems. Proc. of VLDB Aug. 1984. pp. 107-118.

[EG76] K. P. Eswaran,J. N. Gray, R. A. Lorie and I. L. Traiger, The Notions of Consystency and Predicate Lock in a Database System, Comm. ACM Vol. 10, No. 19,pp. 624-633,Nov. 1976.

[GR78] J. N. Gray, Notes on Data Base Operating Systems, IBM Report RJ2188, 1978.

[GR81] J. Gray, The Transaction Concepts : Virtues and Limititions, Procedures on VLDB, 1981, pp. 144-154.

[GS80] V. D. Gligo and S. H. Shattuck, On Deadlock Detection in Distributed Systems, IEEE Trans. Softw. Eng. SE-6, No.5, pp. 435-440, Sep. 1980.

[MF85] C. Mohan, D Fussell, Z. Kedem and A. Silberschaz, Lock Conversion in Non-Two Phase Locking Protocols, IEEE Trans. on Softw. Eng. SE-11, No. 1, Jan. 1985, pp. 15-22

[MM79] D. Menasce and R.Muntz, Locking and Deadlock Detection in Distributed Databases, IEEE Trans. Softw. Eng., SE-5, No. 3, May 1979,pp. 195-202.

[OB82] R. Oberack, Distributed Deadlock Detection Algorithm, ACM Trans. on Database Systems, Vol. 7, No. 2, pp. 187-208, June 1982.

[RE78] D. Reed, Naming and Synchrinization in a Decentrolized Computer System. Tech. Rep. MIT/LCS/TR-205, Dept. Electrical Engineer and Computer Science,Massachusetts Institute of Technology, Sept. 1978.

[SK80] Silberschatz, A. and Kedem,Z. Consistency in Hierarchical Database Systems. Journal of ACM Vol.27, No.1 (Jun.1980). pp. 72-80.

[SK82] A. Silberschatz and Z. Kedem, A Family of Protocol for Database Systems That Are Modled by Directed Graphs, IEEE Trans. on Softw.Eng. Vol. SE-8, NO. 6, Nov. 1982, pp. 558-562.

[TH79] R. Thomas, A Majority Consensus Approach to Concurrency Control for Multiple Copy Database, ACM Trans. on Database systems, Vol. 4, No. 2, June 1979, pp. 180-20.