データベース・システム 58－4
（1 9 8 7. 3. 1 6）

データベースの並行処理制御におけるロックの問題点

九州大学工学部
仲 興国 上林 弥彦

本稿では、データベースの並行処理制御におけるロック概念はロックの動機となるオペレーティングシステムにおけるロックの概念と根本的に異なっていることを指摘する。データベースに対する二相ロック方式に用いられるロックはロックの意味で使用されているのではなく、トランザクションの実行を制御するのに用いられている。従って、データが必要とする時間以上にロックされる。それぞれのデータに対してロックではなく適当な情報を記録することにより、より並行性の高い制御が可能である。この方法には次のような利点がある。

(1) 読み出し-書き込み矛盾によるデッドロックを無くしたアルゴリズムを作ることができる。

(2) 木構造の待ちグラフを用いて、読み出し-書き込み矛盾および書き込み-書き込み矛盾によるデッドロックを無くしたアルゴリズムを作ることができる。

(3) 書き込み-読み出し矛盾がデッドロックを生じない場合にデータの可用性を向上させることができる。

PROBLEM OF LOCKING IN DATABASE CONCURRENCY CONTROL MECHANISMS

Xingguo Zhong and Yahiko Kambayashi
Dept.of Computer Science and Comm. Eng., Kyushu University,

In this paper, we point out that the concept of locking in database concurrency control is essentially different from locking in operating systems. The lock concept used in 2-phase locking method for database is used for controlling the execution of transactions to make the schedules of transactions serializable and recoverable. The time of lock is usually much longer than the time required to operate on the data item. We propose that instead of locking, it is more reasonable to record proper information (called a schedule of requests) on each data item. By recognizing the difference between the function of keeping serializability and the function of locking for using data item, we can achieve several advantages. The major advantages we have obtained are as follows.

(1) We can produce a concurrency control mechanism to be deadlock free in case of read-write confliction.

(2) When we use a specialized structure for wait-for-graph called tree structure, we can make a mechanism to be deadlock free in cases of both read-write and write-write conflictions.

(3) We can improve the availability of data for write-read confliction when it does not cause a deadlock.

1.OVERVIEW

The concept of locking has been introduced from the field of operating systems (OS for short) by J. B. Dennis in 1966 [DE66], where locking is used to avoid incompatible use of resources. Under OS environment, there are many kinds of resources, memory, CPU, disk, program, data, and so on; and several kinds of locks, shared lock, exclusive lock and so on. When a task want to use a resource, it first locks the resource to ensure that the resource may not be, incompatibly, used by other tasks, since a simultaneous use of the exclusive resource may get the system into failure. There are two simple examples in the following.

EX 1 : A certain data object (which might be a word, an array, a list structure and so on) may be updated asynchronously by several tasks, which belong to different computations. Since the intermediate states of the data object is inconsistent, a simultaneous use would lead to erroneous computation.

EX 2 : When a task wants to use a disk driver that is just being used by another task, since the same driver cannot perform two works at the same time, a system failure may occur if there is no locking mechanism.

Lock and unlock operations are originally defined as meta-instructions in the operating systems [DE66]. Even a meta-instruction consists of a sequence of instructions, the system must ensure it to be an atomic one. J. B. Dennis had described lock and unlock operations using an one-bit lock indicator as shown in Fig. 1. Here w is an one-bit lock indicator. The resource is locked when w equals one, and the resource is free when w equals zero. We give the features of locking as follows.

(1) When a task wants to use a resource, it first locks the resource. Hence no other tasks may use this resource until it is unlocked by this task.

(2) When the task ends the use of that resource, it unlocks the resource.

(3) After this task unlocked the resource, other tasks can lock this resource.

(4) The locking mechanism must ensure that there is always at most one task keeping the lock.

With the wide use of lock concept, the deadlock problem has been becoming to be an important topic. Since a task may have to wait for a resource while holding other resources, a wait for cycle of tasks $T_1, T_2, \ldots T_k$, where T_i waits for T_{i+1} ($i=1, 2, \ldots k-1$) and T_k waits for T_1, may occur. To make the deadlock condition clear, E. G. Coffman has given four necessary conditions as follows [CE71].

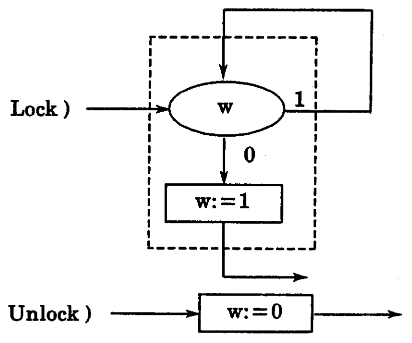


Fig. 1 Lock and unlock meta-instructions

1) Mutual exclusion condition (Tasks claim exclusion of the resources they require)
2) Wait for condition (Tasks hold resources already allocated to them while waiting for additional resources)
3) No preemption condition (Resources cannot be forcibly removed from tasks holding them until the resources are used to completion)
4) Circular wait condition (A circular chain of tasks exists, such that each task holds one or more resources that are being requested by the next task in the chain)

With the development of applications of computer systems, database (DB for short) has been gradually becoming a quite large field, where there exists the same problem of deadlock among transactions. The transactions in DB correspond to tasks in OS and data items to resources.

In this paper, we will point out that the concept of locking used in 2-phase locking method for database concurrency control is essentially different from locking in OS. A lock operation is performed in OS by a task when the task want to use a resource. Once the lock is granted, the task will use the resource instantly. When the task completes the use of this resource, it will unlock it. That is, the period between a lock and unlock pair of a task is exactly same as the period that the task operates on that resource.

In contrast, a lock operation in DB is very different from the lock operation in OS. The lock period is much longer than the period of performing the operation. By a 2-phase locking protocol (see Section 2), a transaction does not unlock any data item it locks until all of its locks are granted. Moreover, there is a typical technique in handling the execution of transactions, called 2-phase commitment protocol (see Section 2), by which when the write lock for a data item is granted, the transaction does not write the data item until the transaction is committed. Thus, it cannot unlock the item. In such a long period of locking, the data items it locks can not be used by any other transaction (see Fig. 2). Thus, we conclude that the lock operations in 2-phase locking method for database concurrency control are performed not for the consistent use of the data item itself, but for the consistency of the whole database.

In Section 2, we give the basic concepts of concurrency control for database systems. The difference between locks in OS and locks in DB then becomes clear. In Sections 3, we describe what advantages can be obtained by changing locking into recording proper information on data items. In Section 4, a framework of controlling the executions of transactions is described. Instead of 2-phase locking, we call our methods 2-phase requesting. Section 4 gives a rough description of these advantages.

2. BASIC CONCEPTS

The concept of locking is described in previous section. In this section, we discuss the basic concepts in database concurrency control which are used in this paper.

(1) Serializability

The concept of serializability is first introduced by K. P. Eswaran in 1976, about ten years after the introduction of lock concept [EG76]. As a general way in discussing concurrency control problem for database, the database is usually viewed as a set of data items, on which read and write operations are performed by transactions. The transactions are defined as the logical units of operations that preserve the consistency of the database [EG76, GR81]. A transaction is supposed to transform the database from a consistent state 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 under some situations. When a transaction is rolled back, all the changes that caused by the execution of that transaction must be recovered.

(2) 2-phase commitment protocol

The 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 assumptions that the system never get into failures and do not allow any cascading of rollback of transactions.

By the 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 in database. After the transaction completes to record all its write operations to logs, the completion of the transaction is guaranteed. In the second phase it reflects the logs to the database. The necessity of having 2-phase commitment is that when a transaction is determined to be rolled back or a transaction fails on some stage of its execution, the data items it has written, may be read by other transactions thus losing the recoverability of the system.

(3) 2-phase locking protocol

The 2-phase locking protocol is a well known method in guaranteeing the serializability of a schedule of transactions. The lock concept used in 2-phase locking protocol is almost same as introduced in previous section, if all the period of

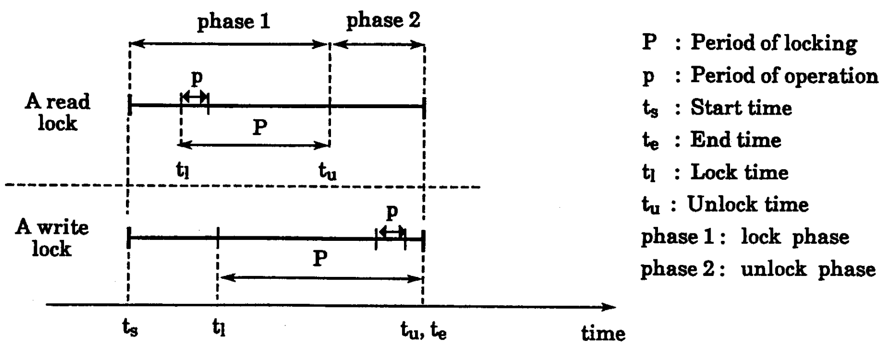


Fig. 2 Lock period and operation period in 2-phase locking method

locking data D is thought to be that the transaction is operating on D. There are read locks and write locks for read and write operations of transactions, respectively. More than one read lock can be imposed on the same data at same time. However, when a data is locked by a transaction in write mode, no other transaction can lock it in any mode. The execution of a transaction is always divided into two phases. In the first phase, it does only locks and in the second phase it does only unlocks of data items. That is, for each transaction once it has unlock 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].

By 2-phase locking methods, clearly, no data item can be unlocked by the transaction until the transaction arrives its second phase. Furthermore, by 2-phase commitment protocol, all the write operations cannot be reflected to database until the transaction is committed. Therefore, a write lock means that the transaction will write the data item it locks in future and the data item cannot be unlocked until the transaction completes. This is very different from locking in OS.

(4) Deadlock

Deadlock problem in database system is the same as in OS, which is discussed in previous section. When a deadlock occurs, all the transactions in the wait for cycle are blocked and cannot be executed further.

(5) Wait-for-Graph

In order to detect deadlocks, a directed graph called a wait-for-graph(WFG) is usually used by the system [GR78]. Each node of the graph corresponds to a transaction being executed by the system. For simplicity, we use the same notation of a transaction to its corresponding node in WFG. When transaction T_i issues a lock on data D that is locked by transaction T_j in conflict mode (at least one of them lock D in write mode), an edge from node T_i to node T_j 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 detecting deadlocks. One is called continuous detection and another is called periodic detection. In continuous detection, detection is performed whenever a new edge is required to be added to the WFG. In periodic detection, detection is performed once in one period of time. The strategy considered in this paper is continuous detection.

3. AN EXAMPLE OF AVOIDING DEADLOCK FOR READ-WRITE CONFLICTION

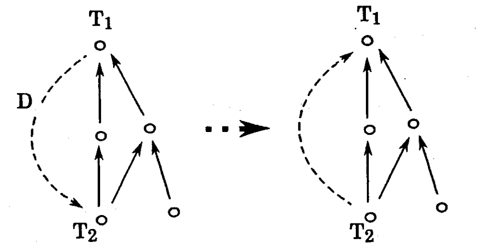
We have introduced the basic concepts of 2-phase locking protocol in the previous section, by which the difference between locking in OS and locking in DB is made clear. In this Section, we give an example on how to achieve the benefits by recognizing the difference of the two concepts.

A transaction issues read and write requests for data items when it is executed. There are three conflicting situations in requesting an operation, read-write, write-write and wirte-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. Similarly, a write-write (write-read) confliction occurs when a transaction requests a write operation on a data item that was written (read) or held for writing by another transaction.

In 2-phase locking method, the confliction caused by a read lock request (read-write confliction) only arose on the situation when transaction T_1 requests to lock data D in read mode that has been locked by another transaction T_2 in write mode. In this situation, if appending a new edge from node T_1 to T_2 will not cause a cycle in the WFG, T_1 can wait until T_2 unlock D. However, if the appending of the edge from T_1 to T_2 will cause a cycle, by conventional methods, transaction T_1 or T_2, or may be other transactions in the cycle have to be rolled back.

In fact, when such a situation occurs, we need not to rollback any transactions. Since T_2 must be blocked for some lock request, it doesn't not yet reflect the new value of D being requested by T_1 to the database (See 2-phase commitment protocol). That is, transaction T_1 can read the existing value of D directly without destroying the serializability of the schedule of transactions. Since T_2 is blocked for some lock request, T_2 may not perform operation on D at the same time with T_1.

Fig. 3(a) shows an example of the WFG in this



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

Fig. 3 Wait-for graphs

situation where transaction T_1 is requesting a read lock that conflicts with a write lock of transaction T_2 on D and causes a cycle in the WFG. If we allow T_1 to read the existing value of D, the deadlock can be avoided. The resulting WFG is shown in Fig.3(b). The dotted edges in the figure express the relationship of T_1 and T_2 that is discussed. By this example, we find that even if T_2 locks D in write mode, since it doesn't perform the write operation, and in this situation it may not be performing the write operation, the benefit that allowing T_1 to read D can be obtained.

We further discuss the functional redundancy between 2-phase locking protocol and the WFG in conventional 2-phase locking methods. In continuous strategy of 2-phase locking, 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 transaction. For this reason, it is not necessary to lock data item that, in fact, hinders the data item to be used by other transactions. Reading the existing value of data D as described above shows that when transaction T_1 requests to read data D, the request might be allowed.

4. CONTROLLING THE EXECUTION OF TRANSACTIONS

We give a framework of controlling the execution of transactions. The main assertions are that we should use a lock mechanism (in OS meaning) to handle requests of transactions in a lower level. The 2-phase locking protocol for transactions is modified to be 2-phase requesting protocol in a higher level.

We have find that there is no sufficient reason to have locking in OS meaning on data for database concurrency control. It is more reasonable to only record proper information on each data item. When a transaction needs to read or determines to write a data item in future, it issues a request to modify the information on that item. The information on each data item (called a schedule of requests) is a sub-graph of the WFG which is constructed by the nodes with the corresponding transactions related to this data item.

In the remaining part of this paper, instead of locking we use the word "request" to express the function of locking for database concurrency control. We call the method 2-phase requesting method. Instead of two lock modes (read and write modes) in conventional way, two request modes on data item are needed. We define them as follows.

(1) Read request : When transaction T_1 needs to read data D, it first issues the read request to D. Once the request is granted, it reads D instantly.

No write request to D can be handled before T_1 ends the read operation on D.

(2) Write request : When transaction T_1 determines to write data D, it first issues the write request to D. If the write request is granted, the write operation will be performed when T_1 is committed.

There is the simultaneous operating problem that more than one request of transaction may call on the same schedule or even call on the WFG simultaneously. We need an additional lock mechanism to control the concurrency execution of the requests. This lock mechanism (a concurrency control mechanism for requests) is to ensure that the execution of the requests issued by each of the transactions are serializable. This is just such a problem of looking each schedule and the WFG as data items and each request as a transaction. An execution of a request have two results, one is to be granted and another is to be rejected. We show that there is no deadlock problem in such a control mechanism as follows.

The schedule on each data item and the WFG may be updated only when a request (lock request in conventional way) is issued. The behavior of handling a request of a transaction is as follows.

(1) The transaction is supposed to issue a request on data item D. When no confliction occurs, the request can be granted. The corresponding schedule on data D should be modified. Since such a work only locks one schedule for one data item (data D here), no deadlock will occur (see the necessary condition 2 in Section 1).

(2) When the request causes a conflict with request of other transactions, the request first locks the schedule on the data item and then locks the WFG. Since each of such work does not lock any other schedule, and each of them always lock the schedule of the data item before locking the WFG. Evidently, no deadlock may occur[SK82, FM85].

By the above descriptions, we give a summary that the lock mechanism for requests supports an interface of controlling execution of transactions as follows.

(1) When a transaction T_1 issues a write request to data D that conflicts with a read request of transaction T_2, the read operation of T_2 on data D is considered having been performed already. That is, a read request does not unlock the schedule it locks until the read operation is over.

(2) When a transaction T_1 issues a read/write request to data D that conflicts with a write request of T_2, the write operation of T_2 on D is not considered having been performed, if T_2 is blocked for some other request.

5. DISCUSSION ON ADVANTAGES FOR DATABASE CONCURRENCY CONTROL MECHANISMS

In this section, we propose three techniques as follows.

(1) A concurrency control mechanism which handles read-write conflictions to be deadlock free.

(2) A concurrency control mechanism which handles both read-write and write-write conflictions to be deadlock free, in which the tree structure for WFG is used.

(3) An improvement of availability of data for write-read confliction.

For space limitation, we only give the basic ideas and the rough descriptions for each of these techniques. For the description of our techniques, we first give a general processing model for processing transactions and operation requests.

5.1 A processing model for transaction and data operations

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

There are read request and write request that must be performed by a transaction before it wants to read and write the data respectively. Once a transaction is blocked, it does not request other data further. Even if a transaction could execute its actions (data accesses, computations, communications and so on) parallely, the requests of one transaction for data items are performed serially. In this way, a transaction can be blocked by only one data item. When a transaction does both read and write operations on same data, two requests are issued distinctly. However, two requests by the same transaction are not judged to be a confliction. It is worthwhile to note that a write request is different from its write operation. The write operation can only be performed at the time when the transaction is committed. However, write request is hoped to be issued to the system as early as possible, since requesting write operation earlier will decrease the possibility of causing deadlock.

5.2 A mechanism avoiding deadlock for read-write confliction

The basic idea of the mechanism that avoids deadlock for read-write confliction has been given in Section 3. We have described it with some discussion in [ZK87]. In the follows, we only give the procedures for handling read and write requests, in which transaction T_i is supposed being requesting data D.

(a) Read request:

a-1 : If Data D is not requested by any other transaction in write mode, the read request of T_i can be granted.

a-2 : If Data D is still requested by transaction T_j in write mode, the mechanism references the WFG to see if it will cause a wait-for cycle when adding an edge from node T_i to T_j. If so, the read request is granted with adding an edge from T_j to T_i to the WFG (avoiding deadlock for read-write confliction).

a-3 : Otherwise, an edge from T_i to T_j is added and Ti is blocked.

When the read request is granted, transaction T_i read D instantly.

(b) Write request:

b-1 : If data D is not requested by any other transaction, the request of T_i is granted.

b-2 : If data D is still requested by transactions T_{jk} ($k=1, 2, \ldots, n$) in which some request is in write mode (there exists a write-write confliction), the request of T_i is rejected and T_i is blocked. When T_i continues its execution, it requests D again.

b-3 : If data D is still requested by transactions T_{jk} ($k=1, 2, \ldots, n$) in only read mode and it will not cause any cycle in the WFG when adding edges from node T_i to each T_{jk} ($k=1, 2, \ldots n$), then the request of T_i is intended to the schedule on D and Ti is blocked. When T_i continues its execution, it requests D again.

b-4 : If adding edges from node T_i to each T_{jk} ($k=1, 2, \ldots, n$) will cause some cycles in the WFG, a real deadlock occurs and some transactions in the cycles have to be rolled back.

When the write request is granted, transaction T_i is determined to write D when T_i is committed.

5.3 A mechanism avoiding deadlock for both read-write and write-write confliction

The basic idea of this mechanism is very similar to the previous one. To avoid deadlock for write-write confliction, the tree structure of the WFG is used [AC83]. The tree structure of the WFG is first proposed by Agrawal et. al, which gets the detection of deadlock being very cheap as only to take a directed walk start from the node of the requesting transaction to the root of the tree. We show it as follows.

When transaction T_i issues a read request on data D that was still requested in write mode by transaction T_j, T_i has to wait and an edge from T_i to T_j is generated. When transaction T_i issues a write request on D that was still requested by transactions T_{jk} ($k=1, \ldots n$) in read mode, instead of adding n edges $\{ Ti \rightarrow T_{jk} \mid k=1, 2, \ldots n \}$ to the WFG, only one node T_j among $\{ T_{jk} \mid k=1, 2, \ldots n \}$ is selected, to which an edge from T_i is added. When transaction T_j is committed or aborted for some reason and T_i can continue its execution, T_i issues its write request again on D. If D is still requested by some transactions, a new edge will be added to WFG as above. If there is no other transaction locking D, the request of T_i on D can be granted. In this way, the WFG always appears as a collection of trees. That is, each transaction may only depend on only one other transaction directly in the WFG. Only transactions with its node in the WFG being a root node of a tree are in active state.

By using the tree structure of the WFG, when a new edge is appended to the WFG, it joins two trees of the WFG into one. When a wait-for cycle occurs, it must occur within one tree with adding a new edge from the root node to some other node. For space limitation, we omit the detail of this mechanism in this paper .

5.4 Improving the availability of data for write-read confliction

In multi-version timestamp ordering mechanism (another typical method for concurrency control [RE78, BG80]), when a transactions T_1 issues a read request on data D that was still requested by another transaction T_2 with smaller teimstamp than T_1, T_1 have to wait until T_2 reflects the new value of data D to the database. But when transaction T_1 issues a write request on D that has been read by T_2, T_1 need not to wait even if T_2 is not committed yet.

In contrast, in conventional 2-phase locking method, the write lock of T_1, which corresponds to the write request of T_1 in multi-version timestamp ordering method of the above, has to wait until T_2 unlock D.

By changing lock into recording proper information on each data item, such a wait of transaction T_1 for T_2 may be avoided. That is, with adding an edge from T_1 to T_2 in the WFG, instead of being blocked, T_1 can go on its work, thus the availability of data D is improved. Of course, the commitment condition of transaction T_1 must be considered. We will not discuss it in this paper.

6. SUMMARY

In this paper, we have described the difference between locking in OS and locking in 2-phase locking method for database concurrency control. Several advantages are proposed by changing locking into recording proper information on each data item.

This idea lays a foundation in realizing a mechanism in which rollback of transactions may occur only when a write request of a transaction conflicts with a read request of another transaction and causes a deadlock (write-read confliction). The transactions issued to the system are independent with each other, and the set of data items, a transaction will operate on, are not determined before starting its execution. Under this assumption, deadlock cannot be avoided. When such a deadlock occurs, since the read operation has been done, a rollback of a transaction have to be performed. We say this, "What has been done has been done". Since such deadlocks are non-solvable and only such deadlocks are not solved under our method, we may say, by making the lock concept clear, we have minimized the possibility of rollback in database concurrency control.

REFERENCES

[AC83] R. Agrawal, M. Caray andd D. Dewitt, Deadlock Detection is Cheap, UC Berkley ERL Memorandum M83/5 (1983).

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

[BS79] Bernstein,P.A. AND 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.

[CE71] E. G. Coffman, M. J. Elphick and A. Shoshani, System Deadlocks, Computing Survey Vol. 3, No. 2, (June 1971) pp. 67-78.

[DE66] J. B. Dennis nad E. C. Van Horn, Programming Semantics of Multiprogrammed Computations, Comm. ACM Vol. 9, No. 3, (Mar. 1966) pp.143-155.

[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, Proceedings of VLDB, 1981, pp. 144-154.

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

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

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

[ZK87] X. Zhong and Y. Kambayashi, Two-Phase Locking Mechanisms Avoiding Deadlock for Read-Write Confliction, LA Symp., Kyoto, Japan, Feb. 1987.