

COMPATIBILITY AND COMMUTATIVITY IN NON-TWO-PHASE

LOCKING PROTOCOLS*

C. Mohan,¹ D. Fussell, A. Silberschatz

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

ABSTRACT

Research on concurrency control mechanisms for database systems has had as a primary goal the discovery of techniques for allowing increased levels of concurrent execution of transactions. In this paper, we study this problem in the context of non-two-phase locking protocols which are defined for data bases in which a directed acyclic graph structure is superimposed on the data items. We introduce a new lock mode, called INV, with properties fundamentally different from locking modes previously studied and show how this allows increased concurrency. Through the introduction of the INV mode of locking we have enunciated a new principle of the theory of data base concurrency control. This principle involves the separation of the effects of the commutativity and compatibility of data manipulation operations. We then examine how the introduction of such a lock mode affects the occurrence of deadlocks in a system. Certain conditions under which deadlock-freedom is maintained are identified, and simple methods for removing deadlocks in other situations are presented.

*This research was partially supported by ONR Contract N00014-80-K-0987 and NSF Grant MCS81-04017. C. Mohan was also supported by an IBM Pre-Doctoral Fellowship.

¹ IBM Research Laboratory, San Jose, CA 95193.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

1. INTRODUCTION

Data base systems usually allow concurrent operations (read and write) by different transactions in order to improve performance. Safeguarding the consistency of the stored/retrieved data is of great significance under such circumstances. The most widely accepted approach to dealing with this problem is to define a transaction as a unit that preserves consistency (i.e., it is assumed that each transaction, when executed alone, transforms a consistent state of the data base into a new consistent state), and require that the outcome of processing a set of transactions concurrently be the same as one produced by running these transactions serially in some order. A system that ensures this property is said to be serializable [1]. Another important issue in data base management that sometimes arises as a result of actions taken by the concurrency control to maintain serializability is the problem of deadlocks. A system which does not allow deadlocks to occur is said to assure deadlock-freedom.

Serializability can be ensured via a number of concurrency control mechanisms, the most common one being a locking protocol. Such a protocol can be simply viewed as a restriction on when a transaction may lock and unlock each of the data base items. Locking a data item in a certain mode inhibits certain kinds of concurrent activity on that item until the item is unlocked. The first useful locking protocol developed was the two-phase locking (2PL) protocol [1], which is characterized by the fact that a transaction is not allowed to lock a data base item after it has unlocked any other item. One drawback of the two-phase protocol is that it severely restricts the amount of concurrency allowed in a system.

This deficiency has led to the development of a number of non-two phase locking protocols [5, 6, 11, 12, 14]. One of the non-2PL protocols is the majority protocol (MP) presented in [5]. This protocol assures serializability and deadlock-freedom.

We introduce a new lock mode, called INV, in an extended version of MP and show how this leads to increased concurrency. Through the introduction of the INV mode of locking, which does not grant any access privileges (read/write) to the holder of the lock on the associated data item, we have enunciated a new principle of the theory of data

base concurrency control. This principle involves the separation of the effects of commutativity (which relates to serializability) and compatibility (which relates to deadlock-freedom) of data manipulation operations. Thus we depart from the traditional approaches to concurrency control which do not make this separation. We also see how the introduction of such a locking mode affects the problem of maintaining deadlock-freedom in a system and show how this problem can be handled.

This paper is organized as follows. In section 2, we introduce the invisible (INV) mode. In section 3, we discuss some existing protocols and motivate the need for the INV mode. Then, in section 4, we present the Super Majority Protocol (SMP), which allows transactions to lock data items in INV, S and X modes. SMP is shown to assure serializability. Section 5 treats the issue of deadlocks in SMP, which is shown to be deadlock-free for rooted trees but not for more general DAG's. Simple and effective methods for handling deadlocks when they occur are given.

2. LOCK MODES AND THEIR PROPERTIES

Let a data base consist of a set of data items V. We ignore the exact nature of the granularity of the items, but it should be noted that we do not consider in our work protocols supporting a variable granularity of locking [4, 8]. Associated with the data base is a set of consistency constraints, the exact nature of which is not of concern to us. A state of the data base is an assignment of values to the elements of V. A given data base state is said to be consistent if that state satisfies the consistency constraints.

One of the components of the data base system is a lock manager which receives and processes the lock requests of the transactions. This means that once a transaction issues a lock request, it cannot proceed until that lock has been granted. In all the protocols to be presented here, locks can only be obtained one at a time. Further, once a data item is unlocked by a transaction it cannot be relocked by that transaction.

We assume that the transactions are well-formed; that is, every action that a transaction performs is permitted by the locks that it holds at the time the action is performed and all locks held by a transaction are released by the transaction before it terminates. We also assume that every transaction when run alone on a consistent state of the data base transforms the latter into a consistent state.

The protocols that we present in this paper allow three modes of locking: X (exclusive), S (shared) and INV (invisible). These modes can be obtained via the LX, LS and LINV instructions, respectively. Locks can be released using the UN instruction. If a transaction holds an X mode lock on a data item, it can read and modify that data item, and if it holds an S mode lock, it can only read the data item. When holding an INV mode lock, the transaction can neither read nor modify the data item (i.e., the data item is invisible to the transaction). While the INV mode is a kind of intention mode in the sense that it conveys some

information about what kind of locks a transaction may acquire in the future, it bears no further resemblance to the intention modes of [4, 8], which are used to support variable granularity of locking. The motivation for the introduction of the INV mode will become apparent to the reader when we present an example in Section 3.

Given a set of locking modes, we can define the compatibility relation among them as follows. Suppose that a transaction T_i requests a lock of mode A on item e on which transaction T_j currently holds a lock of mode B. If the lock manager is allowed to grant T_i 's request in spite of the presence of the mode B lock, then we say mode A is compatible with mode B. Such a relation, particularly on a relatively few locking modes, can conveniently be represented by a matrix. The compatibility relation among the 3 modes of locking used in this paper is given by the matrix, COMP (Figure 2-1). An element, say COMP(I,J), of the matrix has the value T if and only if mode J is compatible with mode I.

	X	S	INV
X			T
S		T	
INV	T		T

Figure 2-1: Lock Compatibility Matrix COMP

Note that INV mode is compatible with X mode, but not with S mode. At any time one X mode lock and zero or more INV mode locks can be simultaneously held (by different transactions) on a particular data item. A subsequent X mode lock request has to wait until the currently held X mode lock is released.

Another important characteristic of a set of locking modes is the commutativity relation among them. We say locking modes A and B are commutative if all operations which a transaction is allowed to perform while holding a mode A lock on an item and all operations which are allowed under a mode B lock are commutative. Thus, the results produced on an item by two transactions, one holding a mode A lock and the other a mode B lock, will be identical regardless of the order in which the transactions acquire locks on the item. As with the compatibility relation, the commutativity relation is conveniently represented by a matrix. The commutativity matrix COMM for the locking modes used here is given in Figure 2-2. COMM(I,J) = T iff modes I and J are commutative. For example, mode S is not commutative with mode X. If one operation reads data item v and a subsequent operation modifies v, then changing the order of these two operations must be assumed to result in a different value being read by the read operation. On the other hand, mode S is compatible with mode S since changing the order of reads will not change the value read. Notice that even though the S and

INV modes are incompatible, they are still commutative.

	X	S	INV
X			T
S		T	T
INV	T	T	T

Figure 2-2: Lock Commutativity Matrix COMM

In previous research on locking, the COMP and COMM matrices were assumed to be identical, as are the matrices presented in this section if we ignore the rows and columns corresponding to the INV mode. Because existing techniques for proving the serializability and deadlock-freedom of locking protocols depend on this fact, we have been forced to augment them with some new methodologies.

We now introduce some standard definitions which will be required in the subsequent technical discussion.

Definition 2-1: A history H is the trace, in chronological order, of the concurrent execution of a set of transactions $T = \{T_0, \dots, T_{n-1}\}$.

Definition 2-2: We define the \langle_e and \prec relations (the "precedes" relations) on a history H of a set T of transactions as follows:

$T_i \langle_e T_j \iff T_i$ has held an M_i mode lock on e initially and T_j has held an M_j mode lock on e later, and $\text{COMP}(M_i, M_j) \neq T$.

$T_i \prec T_j \iff \exists e [T_i \langle_e T_j]$

We note that the \prec relation (the "precedes" relation) pertains to the serializability of the history H . In the above case, it means that in an equivalent serial schedule T_i must appear before T_j . Note the close relationship between the commutativity relation on lock modes and the precedence relation on histories.

Claim 2-1: A protocol assures serializability if and only if for all concurrent executions of transactions following it the associated relation \prec is acyclic.

Definition 2-3: We define the \rightarrow_e and \rightarrow relations (the "wait-for" relations) as follows:

$T_i \rightarrow_e T_j \iff T_j$ is currently holding a lock on e in M_j which is incompatible with the mode M_i in which T_i has requested a lock on e (that is, $\text{COMP}(M_i, M_j) \neq T$).

$T_i \rightarrow T_j \iff \exists e [T_i \rightarrow_e T_j]$

Note that the relationship between the compatibility relation among locking modes and the waits-for relation on histories is similar to that noted above for commutativity and precedence.

Claim 2-2: A protocol assures deadlock-freedom if and only if for all concurrent executions of transactions following it the associated relation \rightarrow is acyclic.

3. EXISTING PROTOCOLS

The majority protocol (MP) for data bases in which the data items are organized as directed acyclic graphs (DAGs) was presented in [5]. It allows transactions to obtain only X mode locks. For simplicity of presentation we restrict our attention here to the version of MP for rooted DAGs.

Let V be the set of vertices of a rooted DAG with the vertex R being the root and each vertex a data item.

1. Any vertex may be locked first.
2. Subsequently, a vertex v can be locked only if locks are held on at least a majority of the parents of v .
3. Vertices may be unlocked any time.

Theorem 3-1: The majority protocol assures serializability and deadlock-freedom.

Proof: See [5]. □

In [6], MP has been extended to a protocol that we call MP' which allows both the X and S modes of locking. In this protocol, which also assures deadlock-freedom, the update (respectively read-only) transactions are allowed to acquire only X (S) mode locks. Further, the update transactions are required to start by locking the root of the DAG first in order for serializability to be assured. In this case, the two types of transactions follow the same majority protocol that we have described above. As a consequence of the requirements of MP' , the update transactions are serialized in the order in which they obtain a lock on the root.

4. THE SUPER MAJORITY PROTOCOL

Let us illustrate how some potential concurrency is lost due to the requirements of MP'. Consider the system consisting of two entities R and Q, where Q is the son of the root R. Suppose that, at a certain point in time, transaction T1, which needs to update only R, obtains an X mode lock on item R. Immediately after that, transaction T2 starts, and it needs to update only Q. T2 is required to acquire an X mode lock on R before it can lock Q. As a consequence of this requirement, T2 is forced to wait until T1 finishes its updating of R. Thus some potential concurrency is lost. What we would like to see happen is T2 being allowed to "overtake" (or "step-over") T1 and lock Q. In this way, both T1 and T2 will be able to perform their updates in parallel. It is only to provide this "overtaking" ability to update transactions, which could potentially lead to an increased level of concurrency, that we have introduced the INV (invisible) mode of locking. We now present a protocol which incorporates this capability into MP'. This protocol, which we call the Super Majority Protocol (SMP), is also intended for data bases organized as rooted DAGs.

We classify the transactions that access the data base into two types:

1. Update transactions -- Those transactions that can issue INV, X and S mode lock requests (at least one X mode lock request must be issued).
2. Read-only transactions -- Those transactions that can issue only S mode lock requests.

The rules of the SMP protocol are:

1. Read-only transactions may start anywhere. They should obey the majority protocol rules stated in the last section.
2. Update transactions:
 1. Each transaction must start at the root; it may lock the root in either INV or X mode.
 2. Subsequently, a vertex v can be locked in mode

INV -- only if INV mode locks are being held on at least a majority of the parents of v, and the first X mode lock has not been acquired so far.

X -- only if X mode locks are being held on at least a majority of the parents of v, or, if this is the first X mode lock request, all the vertices so far locked (in INV mode) are predecessors of v and INV mode locks are

being held on at least a majority of the parents of v.

S -- only if X or S mode locks are being held on at least a majority of the parents of v.

3. Vertices may be unlocked any time.

Observations:

- All the vertices locked in INV mode, if any, will span a rooted DAG, with R as the root.
- All the vertices locked in X mode will span a rooted DAG, with the first vertex to be locked in X mode as the root.
- Once the first X mode lock request has been issued, no more INV mode lock requests can be issued.
- In contrast to the vertices locked in INV and X modes, all the vertices, if any, locked by an update transaction in S mode need not necessarily span only a single connected component.
- Any vertex locked in INV mode must be an ancestor of any other vertex locked in S or X mode. Thus no "useless" INV mode locking is allowed. The INV mode locks are "helpful" only to get the first X mode lock.
- All vertices locked in S mode by an update transaction must be successors of at least the first vertex to be locked in X mode.

We claim that SMP supports more concurrency than MP' for the following reasons. Any transaction which was designed to follow MP' (on a particular rooted DAG) can be run under SMP without any change (on the same DAG). In addition to those transactions, SMP permits other transactions also (update transactions with INV and/or S mode locks). Compared to MP', SMP permits an update transaction to "overtake" other update transactions. Unlike the former, it does not force the update transactions to be serialized according to the order in which they acquired a lock on the root. Moreover, even without the inclusion of invisible locks there may exist certain transactions which request both exclusive and shared locks which are allowed to run under SMP but not under MP'.

Note that in [7] it was shown that MP' is an example of a general methodology for extending serializable protocols allowing only X locks to serializable protocols with both X and S locks. The strategy of increasing concurrency in such protocols using new locking modes with non-equivalent compatibility and commutativity relations is thus a general strategy applying to a wide class of protocols rather than a single

instance of a protocol [9].

Theorem 4-1: The Super Majority Protocol assures serializability.

Proof: Before proceeding with the proof, we need to state some basic definitions and lemmas, the proofs of which are omitted for the sake of brevity.

Definition 4-1: Let $\#$ denote a vertex-enumeration function $\#: V \rightarrow \{0, 1, \dots\}$, where V is the set of vertices of the rooted DAG, such that if vertex e is a descendant of vertex f , then $\#(f) < \#(e)$.

Definition 4-2: Let $L(T_i)$ be the set of all vertices locked by T_i , $LINV(T_i)$ be the set of vertices locked in INV mode by T_i , $LX(T_i)$ be the set of vertices locked in X mode by T_i and $LS(T_i)$ be the set of vertices locked in S mode by T_i . A lock is held on each member of these sets by T_i during its execution.

When $L(T_i) \cap L(T_j) \neq \emptyset$, let the lowest common ancestor of T_i and T_j , denoted $LCA(T_i, T_j)$, be the vertex e such that $\#(e) = \min\{\#(a) \mid a \in \{L(T_i) \cap L(T_j)\}\}$.

Definition 4-3: When T is an update transaction, let $FX(T)$ be the first vertex to be locked in X mode by transaction T (If this vertex is e then $\#(e) = \min\{\#(v) \mid v \in LX(T) \cup LS(T)\}$).

When T is a read-only transaction, let $FS(T)$ be the first vertex to be locked in S mode (If this vertex is f then $\#(f) = \min\{\#(v) \mid v \in LS(T)\}$).

Let $FV(T)$ be the first vertex to be locked by transaction T in a non-INV mode. $FV(T) = FX(T)$, if T is an update transaction. Otherwise, $FV(T) = FS(T)$.

Lemma 4-1: If u and w are distinct vertices in $L(T_1) \cap L(T_2)$, then there exists a (undirected) chain in the underlying graph between u and w which lies entirely in $L(T_1) \cap L(T_2)$.

Lemma 4-2: If $LX(T_1) \cap LS(T_2) \neq \emptyset$ and T_2 is a read-only transaction then $FS(T_2) \in LX(T_1) \cup LINV(T_1)$.

Lemma 4-3: If $LX(T_1) \cap LS(T_2) \neq \emptyset$ and T_2 is an update transaction then $FX(T_2) \in LX(T_1) \cup LINV(T_1)$ and $FX(T_1) \in L(T_2)$.

Lemma 4-4: If $LX(T_1) \cap LX(T_2) \neq \emptyset$ then $FX(T_2) \in LX(T_1) \cup LINV(T_1)$ and $FX(T_1) \in LX(T_2) \cup LINV(T_2)$ and either $FX(T_2) \in LX(T_1)$ or $FX(T_1) \in LX(T_2)$.

Lemma 4-5: When T_1 and T_2 are update transactions, if $LS(T_1) \cap LS(T_2) \neq \emptyset$, then either $FX(T_1) \in LX(T_2) \cup LS(T_2)$ or $FX(T_2) \in LX(T_1) \cup LS(T_1)$.

Lemma 4-6: (Composition lemma) If T_1 is an update transaction with $v \in LX(T_1)$ and no successor of $v \in L(T_1)$ and T_2 is a read-only transaction with $FV(T_2) = v$, then there exists a

transaction T_3 following SMP such that $L(T_3) = L(T_1) \cup L(T_2)$, $LINV(T_3) = LINV(T_1)$, $LX(T_3) = LX(T_1)$ and $LS(T_3) = LS(T_1) \cup LS(T_2) - \{v\}$.

Proof of Theorem 4-1: Assume by contradiction that SMP does not assure serializability. Then without loss of generality there exists a history of a set of transactions which follow SMP for which the precedence relation ($<$) contains a minimal cycle of the form:

$$T_0 < T_1 < \dots < T_{n-1} < T_0$$

Now remove the last lock instruction issued by T_0 along with its associated UN instruction from both T_0 and the history. Continue removing the last remaining lock-unlock pair until any further removals would break the cycle. The remaining pattern of locking instructions of T_0 , which we call T'_0 , still obeys SMP. Continue performing this truncation operation on each transaction in turn to obtain T'_0 through T'_{n-1} and a new history containing the cycle:

$$T'_0 <_{u_0} T'_1 <_{u_1} \dots <_{u_{n-2}} T'_{n-1} <_{u_{n-1}} T'_0$$

Remember that $T'_i <_{u_i} T'_{i+1}$ means that at least either T'_{i+1} or T'_i had locked u_i in X mode. The u_i s in the cycle have been chosen in such a way that (when $n > 2$) for no ancestor v of u_i , $T'_i <_v T'_{i+1}$.

Non-Distinct Vertices:

It is possible for u_i and u_{i-1} to be the same vertex u , giving us the following:

$$T'_{i-1} <_u T'_i <_u T'_{i+1}$$

This can happen only when T'_{i-1} and T'_{i+1} had locked u in S mode and T'_i had locked u in X mode. Note that no other vertex u_j can be equal to u since in that case either T'_i or T'_{i+1} must have locked $u (=u_j)$ in X mode, thus resulting in a shorter cycle with $T'_i < T'_j$ or $T'_j < T'_i$, contradicting the assumed minimality of the cycle.

Note that in the above case T'_i must be an update transaction, while either T'_{i-1} or T'_{i+1} must be a read-only transaction. The latter condition follows from Lemma 4-5 and the assumed minimality of the cycle.

We proceed with the proof by induction on the length (n) of the minimal cycle.

In the sequel, for brevity we drop the superscript ($'$) of the transactions and write T_i for each transaction T'_i . Thus the minimal cycle becomes:

$$T_0 <_{u_0} T_1 <_{u_1} \dots <_{u_{n-2}} T_{n-1} <_{u_{n-1}} T_0$$

Basis Step:

As the basis of the induction we make $n = 2$ and assume the existence of the cycle

$$T_0 <_u T_1 <_v T_0$$

Now consider the vertex w such that $\#(w) = \max\{\#(FV(T_0)), \#(FV(T_1))\}$. From Lemma 4-1 there must be a path, common to T_0 and T_1 , from u and v to w . From the rules of the protocol it should be clear that if a transaction had locked u or v in X mode then it must have locked w also in X mode. By

"pushing" the conflict $T_0 <_u T_1$ along the common path to w we will get $T_0 <_w T_1$. Similarly by "pushing" the conflict $T_1 <_v T_0$ we will get $T_1 <_w T_0$. Thus we will get a contradiction and hence the impossibility of a cycle of length 2.

Induction Step:

Notice that each u_i has to be either $FV(T_i)$ or $FV(T_{i+1})$ and that for each T_i , $FV(T_i)$ is u_i (only if $u_i = u_{i-1}$ or if u_{i-1} is a descendant of u_i), u_{i-1} (only if $u_{i-1} = u_i$ or if u_i is a descendant of u_{i-1}) or an ancestor of u_i and u_{i-1} due to the way each u_i is chosen. The proof of the induction step can be divided into two major cases.

Case 1: Cycles involving only distinct u_i s.

Pick that u_j for which $\#(u_j) = \max\{\#(u_k), k = 0 \dots n-1\}$. It must be the case that $u_j \in \{FV(T_j), FV(T_{j+1})\}$. From the way u_j has been chosen, we also have $\#(u_{j-1}) < \#(u_j)$ and $\#(u_{j+1}) < \#(u_j)$. This means that $FV(T_j)$ must be u_{j-1} or an ancestor of u_{j-1} and u_j , and $FV(T_{j+1})$ must be u_{j+1} or an ancestor of u_j and u_{j+1} . It is clear that neither $FV(T_j)$ nor $FV(T_{j+1})$ equals u_j . Thus we get a contradiction and the impossibility of the cycle.

Case 2: Cycles involving one or more pairs of non-distinct vertices.

It is this interesting case that we consider below. The strategy that we adopt is, given a minimal cycle of length n , to show that by merging two adjacent transactions in the cycle we can produce a history with a minimal cycle of length $n-1$, thus contradicting the induction hypothesis.

Let us consider a pair of non-distinct vertices as described above. We have

$$T_{i-1} <_u T_i <_u T_{i+1}$$

and the history

$$\dots T_{i-1} \text{ LS } u; \dots T_{i-1} \text{ UN } u; \dots T_i \text{ LX } u; \dots \\ T_i \text{ UN } u; \dots T_{i+1} \text{ LS } u; \dots T_{i+1} \text{ UN } u; \dots$$

From the minimization performed above remember that the last lock acquired by T_i is the one (in X mode) on u . All subsequent instructions of T_i will be unlock instructions. Notice that u could not have been locked in a non-INV mode by any transaction other than the above three transactions.

Now we need to consider 3 subcases.

Subcase 1: $u = FV(T_{i+1})$. This immediately implies that T_{i+1} is a read-only transaction. u may or may not be $FV(T_i)$. Similarly u may or may not be $FV(T_{i-1})$ (Note that if u is not equal to $FV(T_i)$ then u must be $FV(T_{i-1})$, as pointed out before).

From Lemma 4-6 we know that we can merge T_i and T_{i+1} , and get, say, T'_i which follows the SMP. Now the above history, after the removal of one unlock of T_i and one lock of T_{i+1} , and appropriate renaming of the other instructions of the two transactions, will look as follows:

$$\dots T_{i-1} \text{ LS } u; \dots T_{i-1} \text{ UN } u; \dots \\ T'_i \text{ LX } u; \dots T'_i \text{ UN } u; \dots$$

This is still a legal history since during the interval I1 other transactions (if any) could have held u in only the INV mode and those transactions would still be able to hold those locks

simultaneously with T'_i since the INV and X modes are compatible.

By merging the two transactions we have obtained the relation

$$T_{i-1} <_u T'_i <_{u_{i+1}} T_{i+2}$$

and hence a cycle of length $n-1$.

Subcase 2: $u \neq FV(T_{i+1})$. We know immediately that u must be $FV(T_i)$. Let v be the vertex such that $\#(v) = \max\{\#(FV(T_{i-1})), \#(FV(T_{i+1}))\}$. Assuming that $v \neq u$, it can be shown that the history must be of the form:

$$<-.I6.-> T_{i-1} \text{ LS } v; \dots T_{i-1} \text{ UN } v; <-.I3.->$$

$$T_i \text{ LINV } v; <-.I5.-> T_i \text{ UN } v; <-.I4.->$$

$$T_{i+1} \text{ LS } v; \dots T_{i+1} \text{ UN } v; \dots$$

Briefly, the reasons as to why this should be the case are: We know that T_{i-1} and T_{i+1} locked v in S mode and that T_i locked it in INV mode. Without loss of generality assume that T_{i+1} is a read-only transaction and that T_{i-1} is an update transaction (i.e., $v = FV(T_{i+1})$). By inducting on the length of the path common to T_i and T_{i+1} from v to u , it can be easily shown that T_i must have locked v before T_{i+1} locked it. Now we need to show that T_{i-1} must have locked v before T_i locked it. Consider the path $v = w_0 \rightarrow w_1 \rightarrow \dots \rightarrow w_{k-1} \rightarrow w_k = u$, common to T_{i-1} and T_i . v must be the only parent of w_1 . Due to the fact that the T_i s were defined so that the $L(T_i)$ s had the minimal number of elements, it must be that the lock on w_j ($j = 1, \dots, k$) must have been obtained because T_{i-1} had a lock on w_{j-1} (in addition to other locks). From the rules of SMP, we can deduce that since T_{i-1} had locked v (i.e. w_0) in S mode it could have locked w_1 only in S mode. By repeating this argument we can show that all vertices on the common path must have been locked by T_{i-1} in S mode. Once this is known it can be easily shown that T_{i-1} must have locked v before T_i locked it.

Notice that during the time interval from the point of unlocking of v by T_{i-1} to the point of unlocking of v by T_{i+1} no transaction could have locked v in X mode. During intervals I3 and I4 some transactions could have locked u in S mode.

Within subcase 2 we need to consider two cases.

Subcase 2a: $v = FV(T_{i+1})$. This means that $u \neq v$.

We modify the history by truncating T_i so that it acquires locks only on the ancestors of v in INV mode, and a lock on v in X mode (instead of INV mode) and subsequently no other locks (i.e., after the acquisition of the X mode lock on v , T_i will consist of only unlock instructions). By doing this the history becomes:

$$\dots T_{i-1} \text{ LS } v; \dots T_{i-1} \text{ UN } v; \dots T_i \text{ LX } v; <-.I5.->$$

$$T_i \text{ UN } v; \dots T_{i+1} \text{ LS } v; \dots T_{i+1} \text{ UN } v;$$

We still maintain $T_{i-1} < T_i < T_{i+1}$.

This history is legal since we know that no transaction could have acquired an X or S mode lock on v during the interval I5. If any transaction T_j other than T_{i-1} or T_{i+1} had locked v in S or X mode then we will have a shorter cycle with $T_i < T_j$ or $T_j < T_i$. This would have contradicted the

induction hypothesis. If there is no such transaction then we have an instance of subcase 1 and we can take the steps outlined there to show the impossibility of a minimal cycle of length n .

Subcase 2b: $v \neq FV(T_{i+1})$. This means that v must be $FV(T_{i-1})$ and hence T_{i-1} must be a read-only transaction. If $u \neq v$, just as in subcase 2a modify T_i so that it acquires an X mode lock on v . If after that the cycle becomes shorter then the proof is finished. Otherwise, whether u equals v or not, we need to do something more (unlike in the case of subcase 2a we do not now have an instance of subcase 1). Notice that this means that the vertex u_{i-2} must be a successor of v and that v had not been locked in X mode by any transaction and that v had been locked in S mode by only T_{i-1} and T_{i+1} .

We merge transactions T_{i-1} and T_i to get the transaction T'_i . T'_i is made to acquire an X mode lock on v . We modify the history by shifting all the lock and unlock instructions of T_i except the unlock v instruction to the very beginning of the history, removing T_{i-1} LS v and replacing T_{i-1} UN v with T'_i LX v .

These changes leave the new history in a legal state since during interval I_6 some other transactions (if any) could have locked v only in INV mode and by letting T'_i hold an X mode lock on v during that period we are not disallowing the former.

By merging the two transactions we have obtained the relation $T_{i-2} <_{u_{i-2}} T'_i <_v T_{i+1}$ and a cycle of length $n-1$, thus contradicting the induction hypothesis.

Thus we have proved the serializability of the SMP. \square

5. DEADLOCKS IN SMP

One of the consequences of the use of a locking protocol to assure serializability is the possibility of creating deadlocks. In this section we will show that on a DAG which is a rooted tree, SMP assures deadlock-freedom, while on an unrestricted DAG, deadlocks are possible. To illustrate this point consider the system depicted in Figure 5.1. The following sequence of execution results in a deadlock.

T_2 LINV R; T_2 LINV B; T_2 LINV C; T_2 LINV E;
 T_2 UN B; T_2 UN C; T_1 LS B; T_1 LS D;
 T_1 LS C; T_1 UN B; T_3 LX R; T_3 LS F;
 T_3 LX B; T_3 LX D; T_1 LS E; T_2 LINV F;

Before we present the proof of the former result, we present a general result concerning deadlocks in any protocol (not restricted to SMP) supporting the INV, X and S modes of locking:

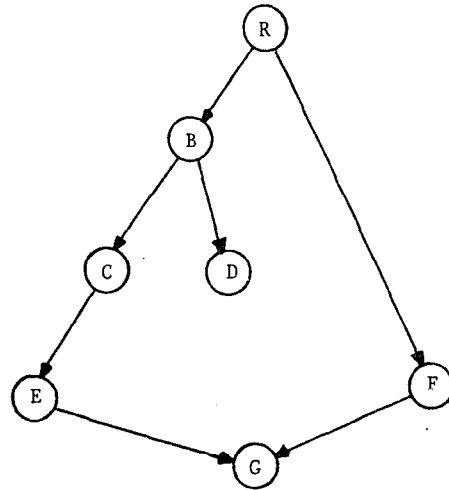


Figure 5.1

5.1 A General Result on Deadlocks

Definition 5-1: When we view a protocol as a set of allowed transaction patterns, a protocol P is said to be closed under truncation if for every locking pattern T of P , the instructions referring to the last item locked by T can be deleted from T to form a new pattern T' of P .

A locking protocol is called a T-protocol if it is closed under truncation. Most locking protocols that have been proposed in the literature are T-protocols.

One of the results of [14] is that if a set of transactions following a T-protocol P which assures serializability are involved in a deadlock of the form

$$T_0 \rightarrow_{u_0} T_1 \rightarrow_{u_1} \dots \rightarrow_{u_{n-2}} T_{n-1} \rightarrow_{u_{n-1}} T_0$$

then for any pair of transactions T_i and T_j involved in the deadlock, $T_i \not< T_j$ and $T_j \not< T_i$.

While [14] considered only the X mode, the above result is true even when we consider T-protocols which allow the X and S modes [2]. The result is not true in general if we consider any T-protocol which supports the INV, X and S modes. As a counterexample, consider SMP and the deadlock situation presented in Figure 5-1, where we have the cycle:

$$T_1 \rightarrow_E T_2 \rightarrow_G T_3 \rightarrow_D T_1$$

In this case the relation $T_1 <_B T_3$ is true at the time of the deadlock.

However, if we impose a few additional restrictions, then the above result will again hold even when INV mode locks are allowed.

Theorem 5-1: If a set of transactions following a T-protocol P which assures serializability and which supports the X, S and INV modes of locking get involved in a deadlock of the form

$$T_0 \rightarrow_{u_0} T_1 \rightarrow_{u_1} \dots \rightarrow_{u_{n-2}} T_{n-1} \rightarrow_{u_{n-1}} T_0$$

and if no u_i in the cycle is either held by T_{i+1} in INV mode or needed by T_i in INV mode, then for all pairs of transactions T_i and T_j involved in the deadlock, $T_i \not\prec T_j$ and $T_j \not\prec T_i$.

Proof: Similar to the proof of the result quoted earlier from [14]. \square

We make use of this result in the next section to prove the deadlock-freedom of SMP for rooted trees.

5.2 Deadlock-Freedom for Rooted Trees

Theorem 5-2: The Super Majority Protocol assures deadlock-freedom for rooted trees.

Proof: The technique that we use for proving deadlock-freedom is the following. Assume the existence of a minimal deadlock cycle involving n transactions. We perform an induction on m , the number of u_i 's which are either held or requested in INV mode by transactions in the cycle at the time of deadlock, to show that such a cycle is impossible.

Basis Step: $m = 0$.

In this case each T_i in the cycle accesses both u_{i-1} and later u_i in non-INV modes. Moreover, T_i must access every vertex on the chain connecting u_{i-1} and u_i in the tree. This is obviously true of u_{i-1} is an ancestor of u_i . Otherwise u_{i-1} and u_i have a single common ancestor e in the chain. This ancestor cannot be locked in INV mode by any transaction obeying the protocol, and thus neither can any of its descendants on paths to u_{i-1} and u_i . Suppose that u_{i-1} is not an ancestor of u_i . Then T_i locks the father f of u_{i-1} in non-INV mode. As we have just seen, so does T_{i-1} , so that either $T_i \prec_f T_{i-1}$ or $T_{i-1} \prec_f T_i$ before the deadlock occurs. By Theorem 5-1 this cannot be the case, so u_{i-1} is indeed an ancestor of u_i . Since this is true for all transactions in the cycle, the tree must contain a cycle containing u_0 through u_{n-1} and we obtain a contradiction. Thus no deadlock involving only exclusive and shared locks can exist.

Induction Step: $m = k$.

From the minimality of the cycle we know that every vertex u_i is being held by only one transaction (namely T_{i+1}) at the time of the deadlock and that all the u_i 's are distinct.

Let us consider the vertex u_i such that

$$\rightarrow_{u_{i-1}} T_i \rightarrow_{u_i} T_{i+1} \rightarrow_{u_{i+1}}$$

There are two cases.

Case 1: u_i is held in INV mode by T_{i+1} . This means that u_i is being requested by T_i in S mode. It also means that T_{i+1} may be requesting an INV, X or S mode lock on u_{i+1} . Since T_{i+1} cannot obtain redundant INV locks, it must be the case that u_i is an ancestor of $FV(T_{i+1})$ and thus of u_{i+1} .

Now we take the following steps:

(i) Find the last transaction (call it T_j), if any,

in the history which locked the vertex u_i in a non-INV mode. If T_j locked u_i in X mode, remove the T_j UN u_i instruction so that T_j continues to hold u_i . By doing this we get a history in which

$$T_0 \rightarrow_{u_0} T_1 \rightarrow_{u_1} \dots T_i \rightarrow_{u_i} T_j \rightarrow_{u_j} \dots \rightarrow_{u_{n-2}} T_{n-1} \rightarrow_{u_{n-1}} T_0.$$

At least transaction T_{i+1} would have been eliminated from the cycle and u_i is no longer held in INV mode by any transaction due to the minimality of the cycle, thus giving us a cycle in which m is less than k and contradicting the induction hypothesis.

(ii) If no T_j as specified in the previous step exists, then modify T_{i+1} so that it acquires an X mode lock on u_i , instead of an INV mode lock. Then move all instructions of T_{i+1} which request locks for the vertices on the path in the tree from u_i to u_{i+1} to that point in the history where all transactions other than T_{i+1} are waiting for their requests to be granted.

Replace instructions requesting INV mode locks with those requesting X mode locks. It may be the case that T_{i+1} gets "stuck" even before it could request a lock on u_{i+1} . This will happen if a vertex, say v , which is predecessor of u_{i+1} is currently being held in X or S mode by a transaction, say T_j , in the deadlock cycle and T_{i+1} requests an X mode lock on v . This creates no problem since we then will have the relation:

$$\rightarrow_{u_i} T_{i+1} \rightarrow_v T_j \rightarrow_{u_j}.$$

This may result in the cycle length becoming smaller than n (we say "may" because it is possible that T_j and T_{i+2} are the same transaction). In the worst case T_{i+1} will get "stuck" requesting a lock on u_{i+1} . In any case, we will have reduced the number of u_i 's on which an INV mode lock is needed or being held by at least 1. There will be a reduction of 2 if T_{i+1} had requested an INV mode lock on u_{i+1} . A contradiction of the induction hypothesis is obtained regardless.

Case 2: u_i is being held by T_{i+1} in S mode. This means that T_i must be needing u_i in INV mode. It also means that u_{i-1} is being held by T_i in INV mode. Now if we consider the pair of vertices u_{i-1} and u_i then we have an instance of Case 1. Hence we can take the steps outlined under Case 1 to handle this case also. \square

5.3 Handling Deadlocks in DAGs

The simplest way to avoid the possibility of deadlocks is to force the transactions to lock vertices in an order (see Definition 4-1) consistent with their enumeration. Forcing the transactions to follow this rule could potentially lead to reduced concurrency since some vertices may remain locked for a time span longer than the time span during which they otherwise would remain locked.

In general, when deadlocks are allowed to occur, they can be resolved using a number of recovery schemes [3, 10, 13], which usually require one or more transactions to be rolled back (i.e., the effects of the transactions to be "undone"). Once

a transaction has been chosen for being rolled back it could be rolled back completely (all the actions of the transaction are "undone") or partially (the transaction is rolled back only to the point in its execution where it is about to acquire a lock on that item which is held by this transaction at the time of the deadlock and which is being needed by the neighboring transaction in the deadlock cycle - by doing this the neighboring transaction can be granted that lock and the deadlock cycle broken). Rolling back one transaction may force some other transactions which subsequently read values of data items produced by the former also to be rolled back. This phenomenon is called cascading rollback.

Once we know that deadlocks can occur and that they must be dealt with, it is important to see if cascading rollbacks can be avoided. The advantages of avoiding cascading rollbacks are:

1. Only one transaction needs to be rolled back to resolve any deadlock. This leads to a decrease in the amount of partial transaction executions that are repeated.
2. More importantly, transaction executions need not be monitored to keep track of transaction dependencies (information about which transaction read which transactions' output). In systems where cascading rollbacks are inevitable, such information is necessary to determine, when a decision is made to rollback a particular transaction, what other transactions must also be rolled back.
3. If cascading rollbacks are not avoided, then an increase in the delay between the time at which a deadlock occurs and the time at which it is detected could potentially lead to an increase in the number of transactions that need to be rolled back.

It turns out that with a very simple condition (which does not restrict the transactions' ability to lock anything that the unrestricted version of the protocol allows) such rollbacks can be avoided. Even the rollback that needs to be done for a single transaction turns out to be a very simple one. It does not require the restoration of any item's value. Before we discuss that simple condition we need to state a property that holds for any possible deadlock cycle under SMP.

Theorem 5-3: If the concurrent execution of a set of transactions following SMP results in a deadlock cycle of the form

$$T_0 \rightarrow_{u_0} T_1 \dots \rightarrow_{u_{i-1}} T_i \rightarrow_{u_i} \dots T_{n-1} \rightarrow_{u_{n-1}} T_0$$

then there exists some u_j in the cycle which is either held in INV mode by T_{j+1} or needed in INV mode by T_j .

Proof: Let us assume that there is no such u_j . Then from Theorem 5-1 we know that at the time of deadlock

$$\forall j \forall k (T_j \nless T_k \text{ and } T_k \nless T_j)$$

This would mean that $u_i \notin \{FV(T_{i+1}), FV(T_i)\}$ and that $FV(T_i)$ is a predecessor of u_i and u_{i-1} or is equal to one of the latter two vertices if that vertex is a predecessor of the other vertex (Note that since we have a deadlock cycle all u_i s will be distinct). With this information in hand, we can follow the steps given in the proof of Theorem 4-1 (basis step and Case 1 of the induction step) to show that such a cycle is not possible. Hence a contradiction and the theorem is proven. \square

Next we state the simple condition which guarantees freedom from cascading rollbacks.

Definition 5-2: We say a transaction satisfies The INV Lock Release (ILR) Condition if it releases all INV mode locks immediately after the acquisition of its first X mode lock (before any reading/modification of the item locked in X mode takes place and before any other locks are acquired).

Theorem 5-4: If all update transactions obey the ILR condition, in addition to the rules of SMP, then when a deadlock occurs there will exist at least one transaction which can be rolled back to break the deadlock without causing any cascading effect (i.e., without other transactions being forced to be rolled back as a consequence) and without having to restore the value of any data base item.

Proof: Let there be a deadlock cycle. Then choose some u_j meeting the conditions of Theorem 5-3.

Case 1: u_j is needed in INV mode by T_j .

From the rules of SMP we know that T_j could have acquired only INV mode locks so far, and thus no updates can have been performed. Therefore u_j can be rolled back by merely unlocking the INV mode locks it holds.

Case 2: u_j is held in INV mode by T_{j+1} .

If T_{j+1} is requesting an INV mode lock on u_{j+1} then we have an instance of Case 1. Otherwise, note that due to the ILR condition T_{j+1} could only be requesting an X mode lock on u_{j+1} and that it must be its first X mode lock request. Since so far T_{j+1} would have acquired only INV mode locks, it can be rolled back by releasing those locks. \square

The interesting thing to notice is that cascading rollbacks are avoided even if we choose to rollback an INV transaction completely (as opposed to partially). This is in contrast to what happens in the case of the pitfall protocol, where only a partial rollback of one of the transactions in the deadlock cycle will avoid a cascading rollback (see [9]).

Injudicious use of rollbacks can, while removing deadlocks, result in a set of transactions becoming involved in a situation in which each transaction in the set in turn causes another transaction in the set to be rolled back. Such a state of affairs has the potential to continue to occur indefinitely in the absence of outside interference, resulting in a dynamic analogue of deadlock which we call potentially infinite mutual preemption. In particular, if every deadlock is resolved only by

rolling back the INV transactions, which is what prevents cascading rollbacks, then it is possible for such a phenomenon to occur, but once such a transaction manages to get its first X mode lock then it is guaranteed to finish (assuming that the scheduling algorithm is fair in granting pending lock requests), although it might become involved in deadlocks after that point in time. Due to the way deadlocks are resolved here, the transaction will not be rolled back as a consequence of the latter deadlocks.

6. CONCLUSION

In this paper, we have given an example of how concurrency can be increased in data base locking protocols by separating the effects of the commutativity and the compatibility of multiple concurrent accesses of data items. In particular, we have introduced a new locking mode, INV, which is used solely for concurrency control and which does not grant any access privileges (read/write) to the holder of the lock on the associated data item. Thus the "operations" performed by a transaction on an item which is locked in INV mode are commutative with respect to any other transactions' operations on that item, even though the S and INV modes are incompatible. As a result, acquisition of an INV mode lock by a transaction on an item does not imply that the transaction be ordered in an equivalent serial schedule with respect to other concurrently active transactions which have locked (in whatever mode at whatever time) the same item.

We have presented a protocol that supports the INV mode of locking. We have examined the effects of this locking mode on the occurrence of deadlocks and have given some effective mechanisms for handling them. These results indicate that in some cases it is useful not to equate the commutativity and compatibility of locking modes as has always been done in previous models of concurrency control.

7. REFERENCES

1. Eswaran, K.P., Gray, J.N., Lorie, R.A., Traiger, I.L., "The Notions of Consistency and Predicate Locks in a Database System," Comm. ACM 10, 11 (November 1976), 624-723.
2. Fussell, D.S., Kadem Z., Silberschatz, A., "A Theory of Correct Protocols for Database Systems," Proc. Seventh International Conference on Very Large Data Bases, September 1981.
3. Fussell, D., Kadem, Z., Silberschatz, A., "Deadlock Removal Using Partial Rollback in Database Systems," Proc. ACM/SIGMOD Int. Conf. on Management of Data, 1981.
4. Gray, J., "Notes on Data Base Operating Systems," Operating Systems, Lecture Notes in Computer Science: Volume 60, Springer-Verlag, 1978.
5. Kadem, Z., Silberschatz, A., "Controlling Concurrency Using Locking Protocols," Proc. Twentieth IEEE Symposium on Foundations of Computer Science, IEEE, October 1979.
6. Kadem, Z., Silberschatz, A., "Non-Two-Phase Locking Protocols with Shared and Exclusive Locks," Proc. Sixth International Conference on Very Large Data Bases, October 1980.
7. Kadem, Z., Silberschatz, A., Locking Protocols: From Exclusive to Shared Locks, University of Texas, Technical Report, 1980.
8. Korth, H., Locking Protocols: General Lock Classes and Deadlock Freedom, Ph.D. thesis, Princeton University, June 1981.
9. Mohan, C., Strategies for Enhancing Concurrency and Managing Deadlocks in Data Base Locking Protocols, Ph.D. thesis, University of Texas at Austin, December 1981.
10. Rosenkrantz, D.J., Stearns, R., Lewis, P.M., "System Level Concurrency Control for Distributed Database Systems," ACM Trans. on Data Base Systems 3, 2 (June 1978), 178-198.
11. Silberschatz, A., Kadem, Z., "Consistency in Hierarchical Database Systems," J. ACM 27, 1 (January 1980), 72-80.
12. Silberschatz, A., Kadem, Z., "A Family of Locking Protocols for Database Systems that are Modeled by Directed Graphs," IEEE Transactions on Software Engineering (to appear), .
13. Stearns, R., Lewis, P.M., Rosenkrantz, D.J., "Concurrency Control for Database Systems," Proc. Twelfth IEEE Symp. on Foundations of Computer Science, October 1976.
14. Yannakakis, M., Papadimitriou, C., Kung, H.T., "Locking Protocols: Safety and Freedom from Deadlocks," Proc. Twentieth IEEE Symp. on Foundations of Computer Science, October 1979.