

Locking Protocols: From Exclusive to Shared Locks

ZVI M. KEDEM

State University of New York at Stony Brook, Stony Brook, New York

AND

ABRAHAM SILBERSCHATZ

University of Texas, Austin, Texas

Abstract. This paper is concerned with the problem of developing a family of locking protocols which employ both SHARED and EXCLUSIVE locks and which ensure the consistency of database systems that are accessed concurrently by a number of asynchronously running transactions. First, a general result concerning extensions of all protocols that employ EXCLUSIVE locks only to also employ SHARED locks is presented. Then a family of protocols applicable to database systems that are modeled by directed acyclic graphs is presented.

Categories and Subject Descriptors: H.2.4 [Database Management] Systems—*transaction processing*

General Terms: Algorithms, Theory

Additional Key Words and Phrases. Consistency, locking, serializability

1. Introduction

A database system in the most general sense may be simply viewed as a pair $DBS = \langle V, \mathcal{T} \rangle$, where V is the set of the database entities and \mathcal{T} is the set of all the transactions that may access V . An important issue which arises in the design of a database system is the problem of ensuring the consistency of the database when it is accessed concurrently by a number of asynchronously running transactions. A common approach to 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 into a consistent state) and require that the outcome of processing a set of transactions concurrently will be the same as the one produced by running these transactions serially in some order. A system that ensures this property is said to be *serializable* [2, 8].

In order to ensure serializability, some form of supervision must be present to influence the manner in which the transactions executing in the database interact with each other. If no such supervision exists, consistency in general is not ensured.

This research was supported in part by the National Science Foundation under Grants MCS 80-25376, MCS 81-04882, MCS 81-04017, and MCS 81-10097 and in part by Office of Naval Research Contract N00014-80-K-0987.

Authors' addresses: Z. M. Kedem, Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794, A. Silberschatz, Department of Computer Sciences, University of Texas, Austin, TX 78712.

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.

© 1983 ACM 0004-5411/83/1000-0787 \$00.75

To illustrate this point, consider the following example [5]. Let program definitions P_1 and P_2 be

$$P_1: \text{ if } A = 0 \text{ then } B := B + 1$$

$$P_2: \text{ if } B = 0 \text{ then } A := A + 1$$

Let the consistency requirement be $(A = 0 \vee B = 0)$, with $A = B = 0$ the initial values. Consider the following sequence of execution:

<u>Instruction</u>	<u>Assertion</u>
$P_1: \text{ if } A = 0$	<i>true</i>
$P_2: \text{ if } B = 0$	<i>true</i>
$P_2: A := A + 1$	$A = 1$
$P_1: B := B + 1$	$B = 1$

In this case we have $\neg(A = 0 \vee B = 0)$ after the execution of both P_1 and P_2 , and thus the state is inconsistent.

One method for influencing the manner in which transactions interact with each other is the use of a *locking protocol*. It is required that a transaction locks an entity before accessing it. Thus a locking protocol may be viewed as a restriction on when a transaction may lock and unlock each of the entities in the set V .

In this paper we focus our attention on locking protocols that allow the database transactions to lock an entity with either a SHARED or an EXCLUSIVE lock. A SHARED or an EXCLUSIVE lock may be issued on an entity by a transaction that only reads that entity; in other cases an EXCLUSIVE lock must be issued. In any concurrent execution any number of transactions may simultaneously hold a SHARED lock on an entity; if any transaction is holding an EXCLUSIVE lock, no other transaction may be holding a lock on that entity.

Several different locking protocols for ensuring serializability have been proposed in the literature. These may be divided into two classes: *two-phase* protocols [2, 5, 8], and *non-two-phase* protocols [4, 6, 10]. It has been shown [2] that the two-phase protocol ensures serializability if both EXCLUSIVE and SHARED locks are permitted. However, practically all of the work concerning non-two-phase protocols has been confined to EXCLUSIVE locks. Although some researchers have believed that most results obtained for the model that uses only EXCLUSIVE locks can be simply generalized to models allowing SHARED locks, this is not the case. In this paper we present results on non-two-phase protocols for models which allow both EXCLUSIVE and SHARED locks.

2. System Model

Following the notation presented in [2], we consider a transaction T_i as a sequence

$$\langle T_i, a_1, e_1 \rangle; \langle T_i, a_2, e_2 \rangle; \dots; \langle T_i, a_n, e_n \rangle,$$

where a_j is the instruction executed at step j and e_j is the entity acted upon by that instruction. (To avoid cumbersome notation, we try to avoid double subscripts.)

Each entity in the database may be locked by either a SHARED or an EXCLUSIVE lock. If a transaction T_i requests a SHARED lock on an entity which is already locked by another transaction with an EXCLUSIVE lock, then T_i will be suspended; otherwise it succeeds in locking. If T_i requests an EXCLUSIVE lock on an entity which is already locked by another transaction (with either SHARED or EXCLUSIVE lock), then it will be suspended; otherwise it succeeds in locking. A request for locking or unlocking an entity is accomplished via the instruction LX (LOCK

EXCLUSIVE), LS (LOCK SHARED), or UN (UNLOCK). (Thus these are among the possible instructions that a transaction may issue.)

We will consider only partially interpreted transactions, namely transactions from which only the LX, LS, and UN were extracted. (Each transaction will of course satisfy certain obvious syntactic restrictions, such as: An item that is not locked cannot be unlocked, etc. For a more thorough discussion of this issue, see [2, 10].) We wish to examine traces of some concurrent executions between two quiescent states. (The discussion could easily be extended to an unbounded set of transactions for which the second quiescent state may not exist.) Such a trace will be referred to as *schedule* (history). In order to distinguish between the event of a transaction requesting a lock and of acquiring a lock, we introduce the pseudo-instructions \widehat{LX} , \widehat{LS} which indicate acquisition of a lock. To clarify this notation, we present a very simple example of a schedule consisting of three transactions accessing a database of two entities a, b :

$\langle T_0, LX, a \rangle; \langle T_0, \widehat{LX}, a \rangle; \langle T_0, LS, b \rangle; \langle T_0, \widehat{LS}, b \rangle; \langle T_1, LX, a \rangle; \langle T_2, LS, a \rangle;$
 $\langle T_0, UN, a \rangle; \langle T_2, \widehat{LS}, a \rangle; \langle T_0, UN, b \rangle; \langle T_2, UN, a \rangle; \langle T_1, \widehat{LX}, a \rangle; \langle T_1, UN, a \rangle.$

Note that T_0 acquires a lock on both a and b immediately after requesting the locks. In contrast, T_1 is delayed after requesting a lock on a until transaction T_2 requests, obtains, and releases a lock on a .

Generally, a schedule may not be “complete,” as the set of transactions participating in it may reach a quiescent state not only by completing the execution of the transactions, but also by entering a *deadlock* [1]. In this paper we will be only concerned with the problem of ensuring serializability. Thus our aim here is to impose restrictions on the various transactions so that each possible complete schedule will be equivalent to some serial schedule, namely, be *serializable*. We plan to deal with the issue of deadlocks in a subsequent paper.

Let the database consist of the entities in $V = \{v_0, v_1, \dots, v_N\}$. Consider a schedule of a finite set of transactions $T = \{T_0, T_1, \dots, T_M\}$. For each $v \in V$ we define a relation \rightarrow^v on T by writing $T_i \rightarrow^v T_j$ for $i \neq j$ if and only if the schedule is of the form

$\dots; \langle T_i, \widehat{L}_I, v \rangle; \dots; \langle T_j, \widehat{L}_{II}, v \rangle; \dots$

and

$\{\widehat{L}_I, \widehat{L}_{II}\} \subseteq \{\widehat{LX}, \widehat{LS}\}, \quad \{\widehat{L}_I, \widehat{L}_{II}\} \cap \{\widehat{LX}\} \neq \emptyset.$

Furthermore, define \rightarrow on T to be the union of the relations,

$\rightarrow^{v_0}, \rightarrow^{v_1}, \dots, \rightarrow^{v_N}.$

THEOREM 1. *A schedule is serializable if and only if the associated relation \rightarrow on T is acyclic.*

PROOF: See [9, Theorem 10.3]. Our relation is different from the one defined there, but as their transitive closures are identical, the proof carries through. \square

Recall that we informally defined a locking protocol P as a set of rules specifying when a transaction may lock a database entity. An alternative to this definition is to define P as a set of (partially interpreted) transactions. (Presumably, a transaction is a member of this set if it satisfies some “natural” conditions.) In the sequel we will switch between these two equivalent definitions of a protocol. Given these definitions, we shall say that a protocol *ensures serializability* if and only if every schedule of transactions following the protocol is serializable.

We conclude this section with one more convenient notation. Let S be a schedule. For each T_i associated with S we define

$$\begin{aligned} LX(T_i) &\triangleq \{v \mid S \text{ is of the form } \dots \langle T_i, \widehat{LX}, v \rangle \dots\}, \\ LS(T_i) &\triangleq \{v \mid S \text{ is of the form } \dots \langle T_i, \widehat{LS}, v \rangle \dots\}, \\ L(T_i) &\triangleq LX(T_i) \cup LS(T_i). \end{aligned}$$

The definition does not require $LX(T_i) \cap LS(T_i) = \emptyset$, but as all the protocols presented in this paper will satisfy this condition, we assume this from now on.

3. General Result

As stated in the introduction, the non-two-phase locking protocols developed thus far cannot be simply generalized to allow the setting of SHARED locks. In this section we present a simple but powerful result concerning this subject.

Assume that each transaction T_i in a locking protocol can request either only EXCLUSIVE locks or only SHARED locks. We will derive a sufficient condition that ensures serializability under this assumption.

To be more formal, let P be a locking protocol that employs only EXCLUSIVE locks and which has been proved to ensure serializability. Suppose one defines a new protocol P' such that each transaction following the protocol

- (1) requests either only EXCLUSIVE locks or only SHARED locks (i.e., $LX(T_i) = \emptyset \vee LS(T_i) = \emptyset$), and
- (2) follows the same restrictions on locking and unlocking as required by P .

Under what conditions will P' ensure serializability? It is easy to see that in general such a new protocol P' does not ensure serializability.

Example 1. Suppose the protocol P is the tree protocol [6]. For the reader's convenience we briefly review the protocol. Assume that the database is organized as a directed rooted tree whose vertices are the database entities with arcs pointing away from the root. The tree protocol is defined by the following conditions on an individual transaction:

- (1) A transaction may lock any vertex first; to lock any other vertex it must be holding a lock on its father.
- (2) A transaction must not lock any vertex more than once.

We showed in [6] that this protocol ensures serializability and deadlock-freedom.

Consider the database system depicted in Figure 1. The following schedule consisting of four transactions, each of which follows the tree protocol and requests either only EXCLUSIVE or only SHARED locks, is not serializable (we omitted the \widehat{LX} and \widehat{LS} pseudo-instructions as each lock is obtained immediately following the request).

$\langle T_0, LS, a \rangle; \langle T_0, LS, b \rangle; \langle T_0, UN, a \rangle; \langle T_1, LX, a \rangle; \langle T_1, UN, a \rangle; \langle T_2, LS, a \rangle;$
 $\langle T_2, LS, b \rangle; \langle T_2, LS, c \rangle; \langle T_2, UN, a \rangle; \langle T_2, UN, b \rangle; \langle T_2, UN, c \rangle; \langle T_3, LX, c \rangle;$
 $\langle T_3, UN, c \rangle; \langle T_0, LS, c \rangle; \langle T_0, UN, b \rangle; \langle T_0, UN, c \rangle.$

Indeed, $T_0 \rightarrow^a T_1 \rightarrow^a T_2 \rightarrow^c T_3 \rightarrow^c T_0$. \square

In the following we present additional conditions which will ensure serializability.

Let P be a locking protocol. We denote by P^X (respectively, P^S) the subset of P consisting of all the transactions setting only EXCLUSIVE (respectively, SHARED) locks. If $P = P^X \cup P^S$, we say that P is *segregated*.

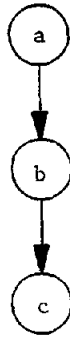


FIGURE 1

Consider some protocol P which requires the transactions to set EXCLUSIVE locks only (i.e., $P = P^X$). We say that a protocol Q is P^X -like if and only if for each transaction T_i in Q , the transaction T_i^X obtained from T_i by replacing each LS lock in T_i by an LX lock is in P^X .

THEOREM 2. *Let $P = P^X$ be a protocol that ensures serializability. Let Q be a P^X -like segregated protocol satisfying the condition*

$$\forall T_i \in Q^X \forall T_j \in Q^S \forall T_k \in Q^X [L(T_i) \cap L(T_j) \neq \emptyset \wedge L(T_j) \cap L(T_k) \neq \emptyset \Rightarrow L(T_i) \cap L(T_k) \neq \emptyset].$$

Then Q ensures serializability.

PROOF. Consider an arbitrary schedule of transactions in the protocol Q . We will show by induction on k that for this schedule there exist no minimal cycles in $\langle T, \rightarrow \rangle$ in which k transactions from Q^S participate. Let the cycle, without loss of generality, be

$$T_0 \rightarrow T_1 \rightarrow \dots \rightarrow T_{m-1} \rightarrow T_0. \tag{*}$$

Project from the schedule the instructions and the pseudo-instructions of the transactions T_0, T_1, \dots, T_{m-1} . We then obtain a schedule for which the graph $\langle T', \rightarrow \rangle$, for $T' = \{T_0, T_1, \dots, T_{m-1}\}$, is a cycle. Observe also that $m \geq 2k$, as there must be a transaction from Q^X "on each side" of a transaction from Q^S .

$k = 0$. In this case each T_i is in Q^X , and as $Q^X \subseteq P$ and P ensures serializability, the result follows.

$k = 1$. Let T_i be the unique transaction in Q^S . Replace T_i by T_i^X . The resulting schedule follows P and has the identical graph (cycle) under the relation \rightarrow . But again, as $\{T_0, \dots, T_{i-1}, T_i^X, T_{i+1}, \dots, T_{m-1}\} \subseteq P$, the result follows.

$k > 1$. Let $T_i \in Q^S$; then $\{T_{i-1}, T_{i+1}\} \subseteq Q^X$. As $m \geq 2k \geq 4$, $i - 1 \neq i + 1$ (modulo m , of course). Therefore, by the assumption of the theorem, $L(T_{i-1}) \cap L(T_{i+1}) \neq \emptyset$. Thus $T_{i-1} \rightarrow T_{i+1}$ or $T_{i+1} \rightarrow T_{i-1}$, and the cycle (*) was not minimal. \square

COROLLARY 1. *Let $P = P^X$ be a protocol that ensures serializability. Let Q be a P^X -like segregated protocol satisfying the condition*

$$\forall T_i \in Q^X \forall T_k \in Q^X [L(T_i) \cap L(T_k) \neq \emptyset].$$

Then Q ensures serializability.

We show now how the condition of Corollary 1 can be used in extending the tree

protocol, denoted now by P^X . Let $Q = Q^X \cup Q^S$ be defined by

$$Q^X = \{T_i | T_i \in P^X \text{ and } T_i \text{ locks the root first}\},$$

$$Q^S = \{T_i | T_i \text{ is obtainable from some } T_j \in P^X \text{ by replacing each LX by LS}\}.$$

Then by Corollary 1, Q ensures serializability.

This protocol is very useful when the database is queried (read) much more often than updated (written). For an interesting observation, assume that it is not known in advance which transactions will both query and update, and which will only query, but it is known that the vast majority of the transactions fall into the latter category. It may appear necessary to run each transaction as an update transaction, as the transaction may decide "on the fly" that it wishes to update. It is, however, much more productive to start each transaction as a querying transaction and, if it decides that it wishes to update (which will happen only in a small proportion of cases), to remove it from the system (it has not changed any values) and restart it as an updating transaction. (Of course, it may turn out that it no longer needs to update if another transaction has modified the database meanwhile.)

4. Extended Guard Protocol

We will now consider a different approach for ensuring serializability. It is natural to consider a database organized as a directed graph whose vertices correspond to the lockable entities and whose arcs correspond to some locking rights. A classical example would be the IMS system [3]; other examples can be found in [4] and [10]. We will thus consider databases modeled by a directed acyclic graph whose vertices are the database entities. Before describing our protocol we find it convenient to present some simple graph theoretical results.

A subgraph H of a graph $G = (V, E)$ spanned by some $W \subseteq V$ is a *partial block* if and only if $|W| \geq 2$ and

- (1) if $|W| = 2$, then the vertices of W are neighbors in G ;
- (2) if $|W| > 2$, then W is biconnected.

A *block* is a maximal partial block. In the sequel we shall refer to W as a block if it spans a block.

One can easily see that any two blocks share at most one vertex. A vertex shared by two (or more) blocks is a *post*. If v is a post and shared by two blocks B_1 and B_2 , we say that it is a post for B_1 and B_2 . As a vertex may be a post for more than one pair of blocks, we say that v is a post for $\{B_1, \dots, B_m\}$, $m \geq 2$, if and only if it is a post for every B_i and B_j for $i \neq j$.

Example 2. Consider the graph G of Figure 2. The blocks of G are

$$B_1 = \{1, 2, 3, 4\}, \quad B_2 = \{4, 5\},$$

$$B_3 = \{5, 6\}, \quad B_4 = \{6, 7, 8\},$$

$$B_5 = \{6, 9, 10\}, \quad B_6 = \{6, 11\}.$$

The posts of G are $\{4, 5, 6\}$: 4 is a post for $\{B_1, B_2\}$; 5 is a post for $\{B_2, B_3\}$; 6 is a post for $\{B_3, B_4, B_5, B_6\}$. \square

LEMMA 1. Let x, y be any two vertices in a connected graph G . Then $\exists m \geq 0$ posts b_1, b_2, \dots, b_m and $m + 1$ blocks B_0, B_1, \dots, B_m such that any chain¹ between x and y is of the form

$$(x = u_1^0, \dots, u_{n_0}^0 = b_1 = u_1^1, \dots, u_{n_1}^1 = b_2 = u_1^2, \dots, u_{n_{m-1}}^{m-1} = b_m = u_1^m, \dots, u_{n_m}^m = y)$$

¹ A chain is always undirected and, unless otherwise stated, simple.

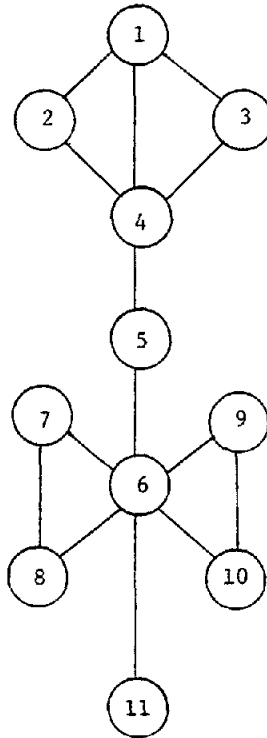


FIGURE 2

and satisfies the conditions

$$\forall i \forall j [n_i \geq 2 \text{ and } u'_i \in B_i].$$

(If $m = 0$ the lemma reduces to stating that any chain between x and y is entirely within a single block B_0 .)

LEMMA 2. Let G be a graph, B a block of G , and S_1, S_2, S_3 sets of vertices of G spanning connected subgraphs and satisfying $S_1 \cap S_2 \neq \emptyset, S_2 \cap S_3 \neq \emptyset, S_1 \cap B \neq \emptyset, S_2 \cap B = \emptyset, S_3 \cap B \neq \emptyset$. Then $S_1 \cap S_3 \cap B \neq \emptyset$.

PROOF. Let $u \in S_1 \cap B, w \in S_2, v \in S_3 \cap B$. Let $w = x_0, x_1, \dots, x_{p-1} = u$ and $w = y_0, y_1, \dots, y_{q-1} = v$ be chains such that

$$\begin{aligned} \{x_0, x_1, \dots, x_{p-1}\} &\subseteq S_1 \cup S_2, \\ \{y_0, y_1, \dots, y_{q-1}\} &\subseteq S_2 \cup S_3. \end{aligned}$$

Consider the not necessarily simple chain,

$$y_{q-1}, y_{q-2}, \dots, y_0 = x_0, x_1, \dots, x_{p-2}, x_{p-1}.$$

Choose a, b such that

$$y_{q-1}, y_{q-2}, \dots, y_a = x_b, x_{b+1}, \dots, x_{p-1}$$

is a simple chain. As y_{q-1} and x_{p-1} lie in a single block B , it follows that

$$\{y_{q-1}, \dots, y_a = x_b, \dots, x_{p-1}\} \subseteq B.$$

But as $S_2 \cap B = \emptyset$, we have $y_a \in S_3, x_b \in S_1$, and the vertex y_a lies in $S_1 \cap S_3 \cap B$. \square

We continue now with our discussion concerning the new protocol. In a previous paper [7] we considered a natural family of protocols with EXCLUSIVE locks only, which generalizes a number of previously proposed protocols. In the sequel we will extend the results to allow both EXCLUSIVE and SHARED locks. Our proof methodology could be used to prove more general results; however, we feel that the results we present are easier to understand intuitively.

Let V be the set of the (lockable) entities of the databases. We will construct by stages a *guarding graph* for V .

- (1) Enumerate V , namely, arrange the elements of V into a sequence v_1, v_2, \dots, v_n . When convenient, we will use V to refer to this sequence too.
- (2) Define a function **guard**: $V \rightarrow 2^{2^n \times 2^n}$ such that if

$$\text{guard}(v_k) = \{ \langle A_1^k, B_1^k \rangle, \langle A_2^k, B_2^k \rangle, \dots, \langle A_{n_k}^k, B_{n_k}^k \rangle \},$$

then

- (a) $\emptyset \neq B_i^k \subseteq A_i^k \subseteq V$, and
- (b) $v_q \in A_i^k \Rightarrow q < k$.

- (3) Define the graph $G = \langle V, E \rangle$ by $\langle v_q, v_k \rangle \in E \Leftrightarrow \exists i [v_q \in A_i^k]$.

This graph (which is of course acyclic) will be a *guarding graph* if the following two conditions are satisfied:

- (1) Any A_i^k lies within a single block of G .
- (2) Whenever $A_i^k \cap B_j^k = \emptyset$, then for every block B of G , either $A_i^k \cap B = \emptyset$ or $A_j^k \cap B = \emptyset$.

Formally we can define a *guarding graph* G for V by specifying the pair $\langle V, \text{guard}(V) \rangle$.

Example 3. Consider the graph of Figure 3 together with the guards defined in Table I. Examine, for instance $\text{guard}(v_8)$. One block contains $\{v_3, v_4, v_5, v_6, v_8\}$; another block contains $\{v_7, v_8, v_9, v_{10}\}$. Thus, as $A_1^8 = \{v_4, v_5\}$ and $B_2^8 = \{v_5, v_6\}$ lie in a single block, we must have $A_1^8 \cap B_2^8 \neq \emptyset$. On the other hand, as A_1^8 and $B_4^8 = \{v_7\}$ do not lie in a single block, $A_1^8 \cap B_4^8 = \emptyset$ is not prohibited. \square

To facilitate intuitive understanding, we first present the Guard Locking Protocol (GLP) for some guarding graph G , allowing EXCLUSIVE locks only. (It can be shown that GLP ensures serializability and deadlock-freedom.)

- (1) A transaction may lock any vertex first; to lock any other vertex v_k it must be holding a lock on the vertices in some B_i^k (thus $n_k > 0$) and must have locked (and possibly unlocked) the vertices in the corresponding $A_i^k - B_i^k$.
- (2) A transaction must not lock any vertex more than once.

For a very simplified discussion consider G consisting of a single biconnected component and two transactions T_0, T_1 , attempting to lock some vertex v_k . Assume also that they both previously locked some vertex v_q . Clearly, we wish to enforce some priority in the order in which they lock v_k . (Indeed, if $T_0 \rightarrow^{v_q} T_1$, we must preclude $T_1 \rightarrow^{v_k} T_0$.) The protocol ensures that if T_0 is holding a lock on the vertices of some B_i^k , T_1 cannot during that period lock (and possibly "partially" unlock) vertices in any A_j^k . Thus if T_0 , after establishing priority of locking on v_q , maintains this priority at least through ancestors of v_k , it will also keep it on v_k . This is, of course, a very imprecise argument, to be formalized later.

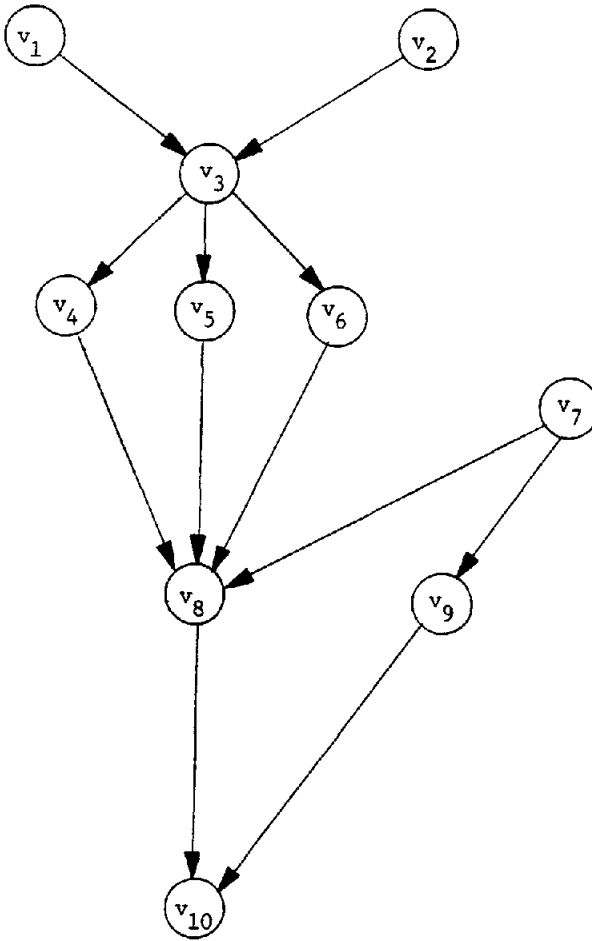


FIGURE 3

TABLE I

Vertex	Guard
v_1	\emptyset
v_2	\emptyset
v_3	$\{\langle v_1, \{v_1\} \rangle, \langle \{v_2, \{v_2\}\} \rangle\}$
v_4	$\{\langle \{v_3, \{v_3\}\} \rangle\}$
v_5	$\{\langle \{v_3, \{v_3\}\} \rangle\}$
v_6	$\{\langle \{v_3, \{v_3\}\} \rangle\}$
v_7	\emptyset
v_8	$\{\langle \{v_4, \{v_4\}\}, \langle \{v_5, \{v_5\}\} \rangle, \langle \{v_6, \{v_6\}\} \rangle, \langle \{v_7, \{v_7\}\} \rangle\}$
v_9	$\{\langle \{v_7, \{v_7\}\} \rangle\}$
v_{10}	$\{\langle \{v_8, \{v_8\}\}, \langle \{v_9, \{v_9\}\} \rangle\}$

Example 4. Consider now the graph of Figure 4 with the (presumed) guards defined in Table II. Here $\{v_2\} \cap \{v_3\} = \emptyset$, but v_2 and v_3 lie in a single block. Thus $\text{guard}(v_4)$ does not satisfy the conditions.

Indeed, the following schedule of two transactions following GLP is not serializable

FIGURE 4

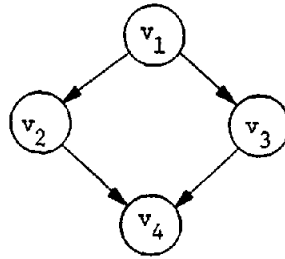


TABLE II

Vertex	Guard
v_1	\emptyset
v_2	$\{\{v_1\}, \{v_1\}\}$
v_3	$\{\{v_1\}, \{v_1\}\}$
v_4	$\{\{v_2\}, \{v_2\}\}, \{\{v_3\}, \{v_3\}\}$

(\widehat{LX} and \widehat{LS} are omitted):

- $\langle T_0, LX, v_1 \rangle; \langle T_0, LX, v_2 \rangle; \langle T_0, UN, v_1 \rangle;$
- $\langle T_1, LX, v_1 \rangle; \langle T_1, LX, v_3 \rangle; \langle T_1, LX, v_4 \rangle;$
- $\langle T_1, UN, v_1 \rangle; \langle T_1, UN, v_3 \rangle; \langle T_1, UN, v_4 \rangle;$
- $\langle T_0, LX, v_4 \rangle; \langle T_0, UN, v_2 \rangle; \langle T_0, UN, v_4 \rangle;$

We have in this case

$$T_0 \rightarrow^{v_1} T_1 \rightarrow^{v_4} T_0. \quad \square$$

The GLP is actually a class of locking protocols for a specific set V , one for each possible choice of enumeration and of guards. For example, the protocols on graphs described in [4, 10] are special cases of the GLP. The simplest one is the tree protocol on an unrooted directed tree. (A directed graph that is a tree is not rooted if it has more than one source.) Here the guarding graph is simply the original directed tree where $\text{guard}(v_j) = \{\{v_i\}, \{v_i\} \mid v_i \text{ is a predecessor (father) of } v_j\}$.

Allow now a transaction to acquire both EXCLUSIVE and SHARED locks. Given a transaction T_i executing in a database organized as a DAG G , the subsets $LX(T_i)$ and $LS(T_i)$ are defined as above. Consider the subgraph of G spanned by the set $LS(T_i)$. Generally it splits into a number of connected components, say C_1, C_2, \dots, C_k . A *pitfall* of T_i is defined as a set of the form $C_q \cup \{u \in LX(T_i) \mid \langle u, w \rangle \in E \text{ or } \langle w, u \rangle \in E \text{ for some } w \in C_q\}$. Note that the pitfalls are subsets of $L(T_i)$, and are not necessarily disjoint.

Example 5. Consider the graph in Figure 5. Transactions T_0 locked vertices such that $LS(T_0) = \{d, e, f, j, r\}$, $LX(T_0) = \{b, g, h, i, k, l, n, p, q\}$. $LS(T_0)$ splits into three components: $\{d, f, j\}$, $\{e\}$, $\{r\}$. The pitfalls are $\{b, d, f, i, j, n\}$, $\{b, e, g, h\}$, $\{p, r\}$. \square

We will say that a transaction T_i (which locks each entity at most once) is *two-phase* on a set of entities $\{u_1, u_2, \dots, u_k\} \subseteq L(T_i)$ if and only if it is of the form

$$I_1, I_2, \dots, I_p, I_{p+1}, I_{p+2}, \dots, I_q,$$

such that the following two conditions hold:

$$\forall j \leq k \exists r \leq p \{I_r = \langle T_i, L, u_j \rangle\} \quad \text{where } L \in \{LX, LS\}$$

and

$$\forall r \leq p \forall j \leq k \{I_r \neq \langle T_i, UN, u_j \rangle\}.$$

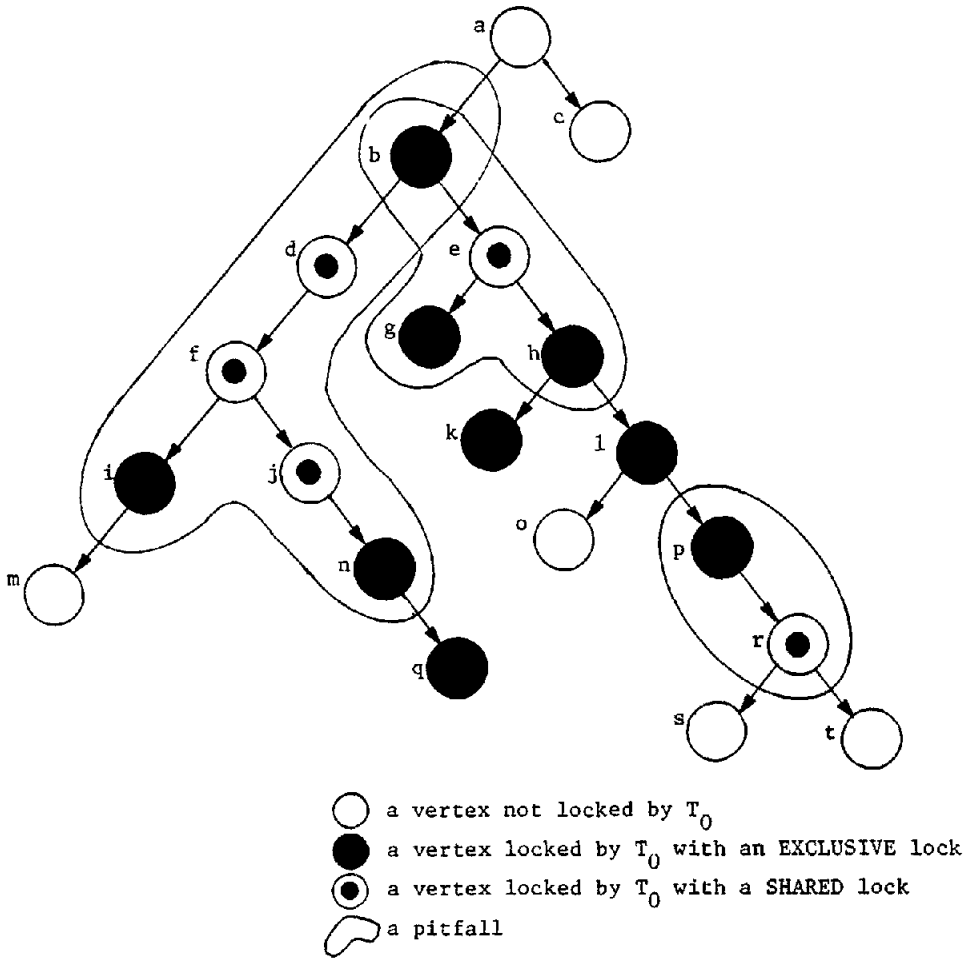


FIGURE 5

In other words, T_i is two-phase on some subset $W \subseteq L(T_i)$ if and only if it locks all the entities of W before unlocking any of them.

We define the *Extended Guard Locking Protocol* (EGLP) on a guarding graph by the following conditions on an individual transaction (in EGLP):

- (1) A transaction may lock any vertex first; to lock any other vertex u_k , it must be holding a lock on the vertices in some B_i^k (thus $n_k > 0$) and must have locked (and possibly unlocked) the vertices of the corresponding $A_i^k - B_i^k$.²
- (2) A transaction must not lock any vertex more than once.
- (3) A transaction must be two-phase on each of its pitfalls.³

Example 6. Example 4 shows that the GLP may fail if both EXCLUSIVE and SHARED locks are permitted. We present another example of a nonserializable schedule of transactions following the tree protocol of the graph in Figure 1 (\widehat{LX} and

² We shall say that the transaction uses $\langle A_i^k, B_i^k \rangle$ to lock u_k . Note that it may use more than one pair to lock a single vertex.

³ Note that even if two pitfalls share a vertex it does not necessarily follow that the transaction is two-phase on their union.

\widehat{LS} are omitted):

$\langle T_0, LX, a \rangle; \langle T_0, LS, b \rangle; \langle T_0, UN, a \rangle; \langle T_1, LX, a \rangle;$
 $\langle T_1, LS, b \rangle; \langle T_1, LX, c \rangle; \langle T_1, UN, a \rangle; \langle T_1, UN, b \rangle;$
 $\langle T_1, UN, c \rangle; \langle T_0, LX, c \rangle; \langle T_0, UN, b \rangle; \langle T_0, UN, c \rangle;$

we have $T_0 \rightarrow^a T_1 \rightarrow^c T_0$.

The cycle was created because T_0 and T_1 "switched priority" at b . T_0 locked it first, but T_1 used it first to lock c before T_0 did so. Requiring a transaction to be two-phase on a pitfall in effect lets it build a "fence" around an area in which it is vulnerable. (This, of course, is a gross oversimplification.) \square

We shall say that a transaction T_i *delays unlocking* of $M \subseteq L(T_i)$ if and only if every vertex of M is unlocked only after all the vertices of $L(T_i)$ have been previously locked. (There are no restrictions on unlocking of vertices in $L(T_i) - M$.)

Example 7. The transaction

$\langle T, LX, a \rangle; \langle T, LX, b \rangle; \langle T, UN, b \rangle; \langle T, LS, c \rangle;$
 $\langle T, LX, d \rangle; \langle T, UN, c \rangle; \langle T, UN, d \rangle; \langle T, UN, a \rangle$

delays unlocking of $\{c, d\}$ (and also of $\{a, c, d\}$, but not of $\{b, c\}$). \square

Let (w_1, w_2, \dots, w_k) be a chain contained entirely within $L(T)$ for some transaction T . We shall say that T is *piecewise two-phase* on (w_1, \dots, w_k) if and only if it is two-phase on every $\{w_i, w_{i+1}\}$ for $i = 1, \dots, k - 1$.

Example 8. Consider again the guarding graph of Example 3. The transaction

$\langle T, LX, v_3 \rangle; \langle T, LX, v_4 \rangle; \langle T, LX, v_5 \rangle; \langle T, LX, v_6 \rangle; \langle T, UN, v_3 \rangle; \langle T, UN, v_6 \rangle;$
 $\langle T, LX, v_8 \rangle; \langle T, UN, v_4 \rangle; \langle T, UN, v_5 \rangle; \langle T, UN, v_8 \rangle$

follows the GLP on the graph. It is piecewise two-phase on (v_3, v_4, v_6) , but it is not piecewise two-phase on (v_3, v_6, v_8) . \square

Let T be a transaction on some graph G , and let H be a subgraph of G . Then T^H , the *restriction* of T to H , is obtained from T by omitting from it the instructions referring to vertices not in H .

LEMMA 3. Let G be a guarding graph that is a (single) block, and let T_0 and T_1 be two transactions following the EGLP on G such that $L(T_0) \cap L(T_1) \neq \emptyset$. Then

- (i) There exists a transaction \mathcal{T} following the EGLP on G for which $L(\mathcal{T}) = L(T_0) \cup L(T_1)$.
- (ii) If x and y are two distinct vertices in $L(T_0) \cap L(T_1)$, then there exists a chain in G between x and y on which T_0 is piecewise two-phase and which lies entirely in $L(T_0) \cap L(T_1)$.

PROOF. V is the sequence v_1, v_2, \dots, v_n . We define $x < y$ if and only if $x = v_i$, $y = v_j$, and $i < j$. Thus, for any subset W of V , the minimum vertex in W is defined. Let $F(T_0), F(T_1)$ be the vertices locked first by T_0 and T_1 , respectively.

(i) We first show that $F(T_0) \in L(T_1)$ or $F(T_1) \in L(T_0)$. Assume otherwise. Let $q = \min(L(T_0) \cap L(T_1))$. Then $q \notin \{F(T_0), F(T_1)\}$, and it follows that T_0 and T_1 used some $\langle A_0, B_0 \rangle$ and $\langle A_1, B_1 \rangle$, respectively, to lock q . But, as G is a block, it follows that $B_0 \cap A_1 \neq \emptyset$. Thus, for any $p \in B_0 \cap A_1$, we have $p \in L(T_0) \cap L(T_1)$, contradicting the definition of q .

Without loss of generality, $F(T_1) \in L(T_0)$. For simplicity assume that both T_0 and T_1 are two-phase (on $L(T_0)$ and $L(T_1)$, respectively). There is no loss of generality in

this assumption, as for any transaction T_0 which follows the EGLP and is of the form,

$$I_1; \dots; I_{p-1}; I_p = \langle T_0, \text{UN}, x \rangle; I_{p+1}; I_{p+2}; \dots; I_m$$

there exists a transaction T'_0 of the form,

$$I'_1; \dots; I'_{p-1}; I'_{p+1}; I'_p = \langle T'_0, \text{UN}, x \rangle; I'_{p+2}; \dots; I'_m,$$

which follows the protocol, where I'_j is obtained from I_j by replacing T_0 by T'_0 .

Define a two-phase transaction T locking exactly $L(T_0) \cup L(T_1)$ by the sequence (we write L for either LX or LS as appropriate)

$$I_1; \dots; I_p = \langle T, L, F(T_1) \rangle; I_{p+1}; \dots; I_q; \dots; I_N; J_1; \dots; J_N,$$

where

$I_1; \dots; I_p$ is the initial subsequence of instructions of T_0 (up to and including the instruction locking $F(T_1)$).

$I_{p+1}; \dots; I_q$ consists of all the locking instructions issued by T_1 and referring to vertices not yet locked.

$I_{q+1}; \dots; I_N$ consists of the remaining locking instructions of T_0 , other than the instructions locking elements of $L(T_0) \cap L(T_1)$.

$J_1; \dots; J_N$ consists of the unlocking instructions.

It is clear that T follows the EGLP on G .

(ii) Assume the converse. We will say that x, y in $L(T_0) \cap L(T_1)$ satisfy the chain condition if they can be connected by a chain that satisfies the lemma. Let $x, y \in L(T_0) \cap L(T_1)$ be minimal that *do not* satisfy the chain condition. (This means that if $x_1 \leq x, y_1 \leq y, \{x_1, y_1\} \subseteq L(T_0) \cap L(T_1)$, and $\{x_1, y_1\} \neq \{x, y\}$, then x_1, y_1 satisfy the chain condition.)

We first show that $\{x, y\} \neq \{F(T_0), F(T_1)\}$. Indeed, otherwise as $\{x, y\} \subseteq L(T_0) \cap L(T_1)$, it follows that $F(T_0) \in L(T_1)$ and $F(T_1) \in L(T_0)$. From here, $F(T_1) \leq F(T_0)$ and $F(T_0) \leq F(T_1)$, and thus $F(T_0) = F(T_1)$, contradicting $x \neq y$.

Thus, without loss of generality, assume that $y \notin \{F(T_0), F(T_1)\}$; T_0 and T_1 used (A_0, B_0) and (A_1, B_1) , respectively, to lock y ; and $B_0 \cap A_1 \neq \emptyset$. Let $q \in B_0 \cap A_1$. Note that T_0 is two-phase on $\{q, y\}$.

If $x = q$, then the chain (x, y) satisfies the lemma. If $x \neq q$, then x, q are two vertices which, by the minimality assumption above, satisfy the chain condition. Let the appropriate chain be

$$(x = w_0, w_1, \dots, w_{k-1} = q).$$

Now, as this chain does not include y (otherwise x, y would satisfy the chain condition), the chain

$$(x = w_0, w_1, \dots, w_{k-1} = q, w_k = y)$$

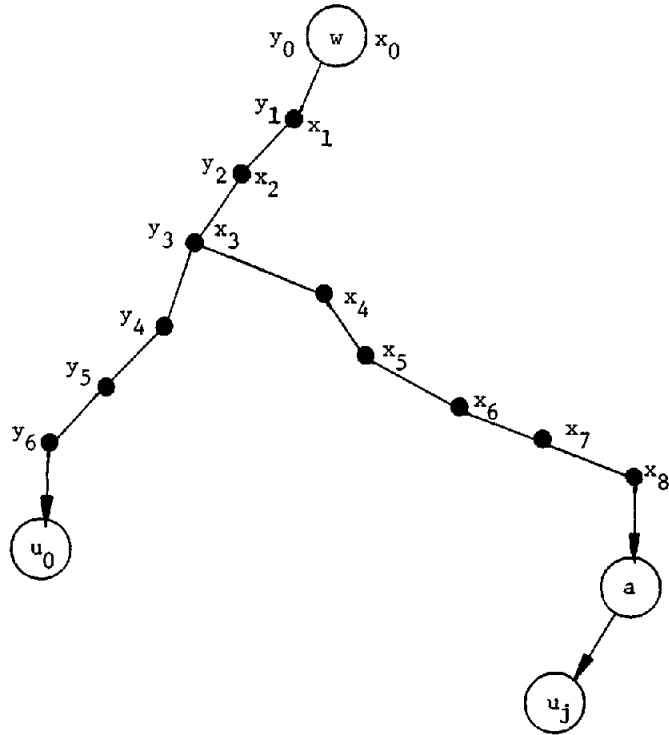
satisfies the chain condition—a contradiction. \square

LEMMA 4. Let $G = \langle V, E \rangle$ be a guarding graph, and let $H = \langle W, F \rangle$ be a block of G . Then H is a guarding graph with the choice of guards defined by

$$\text{guard } H(u) = \{ \langle A \cap W, B \cap W \rangle \mid B \cap W \neq \emptyset \wedge \langle A, B \rangle \in \text{guard}(u) \}$$

for every $u \in W$. (In effect, we define $\text{guard } H(u)$ by restriction.) Furthermore, if T follows the EGLP on G , then T^H follows the EGLP on H .

FIGURE 6



PROOF. If $|W| = 2$, the result follows immediately. Assume that $|W| \geq 3$. As for $\langle A, B \rangle \in \text{guard}(u)$ for any u , A lying in a single block of G , it easily follows that H is a guarding graph and we only show that T^H follows EGLP on H .

We wish, in effect, to show that W contains all the guards T^H needs. (One can imagine that removal of some locking instructions from a transaction causes it not to follow the locking protocol.) Let T^H lock in order the vertices u_0, u_1, \dots, u_{k-1} . We prove by induction on j that when T^H issued instructions locking u_j , it was permitted to do so under the rules of the EGLP.

$j = 0$. Any vertex can be locked first.

$j > 0$. T used some $\langle A, B \rangle$ to lock u_j . It will suffice to show that $A \subseteq W$. The proof is similar to the proof of Lemma 2. Assume by contradiction that $a \in A - W$. Let w be the first vertex locked by T . By the rules of the protocol there exists a path ("directed" chain) from w to a , say, $(w = x_0, x_1, x_2, \dots, x_{p-1} = a)$, consisting of some vertices locked and possibly unlocked by T . There exists also a path from w to u_0 , say, $(w = y_0, y_1, y_2, \dots, y_{q-1} = u_0)$, consisting of vertices locked and possibly unlocked by T (see Figure 6). Consider the sequence

$$z_0 = y_{q-1}, z_1 = y_{q-2}, \dots, z_{q-1} = y_0, z_q = x_1, z_{q+1} = x_2, \dots, z_{q+p-2} = a.$$

For every $f, 0 \leq f \leq q + p - 3$, either $\langle z_f, z_{f+1} \rangle \in E$ or $\langle z_{f+1}, z_f \rangle \in E$. Clearly, $z_0 \in W, z_{q+p-2} \notin W$. Let z_r be the last vertex in the sequence that is also in W . It is easily seen that it is possible to extract from $z_0, z_1, \dots, z_{q+p-1} = u_j$, a sequence $t_0 = z_r, t_1, \dots, t_{s-2} = a, t_{s-1} = u_j$, satisfying the conditions:

- (i) $s \geq 3$,
- (ii) $\forall e [0 \leq e \leq s - 2 \Rightarrow \langle t_e, t_{e+1} \rangle \in E \text{ or } \langle t_{e+1}, t_e \rangle \in E]$,
- (iii) $\forall e [0 < e < s - 1 \Rightarrow t_e \notin W]$,
- (iv) $\{t_0, t_{s-1}\} \subseteq W$.

But then it follows that $W \cup \{t_1, t_2, \dots, t_{s-2}\}$ is biconnected in G , contradicting the definition of H . \square

LEMMA 5. *Let $\{T_0, T_1, \dots, T_{n-1}\}$ be a set of transactions following the EGLP on a guarding graph G such that $L(T_i) \cap L(T_{i+1}) \neq \emptyset$ for $i = 0, 1, \dots, n - 1$ (subscripts are modulo n). Then for any $x \in L(T_0) \cap L(T_1)$ and $y \in L(T_{n-1}) \cap L(T_0)$ there exists a chain between x and y on which T_0 is piecewise two-phase, and which lies entirely within $L(T_0) \cap \bigcup_{i \neq 0} L(T_i)$.*

PROOF. As both $L(T_0)$ and $\bigcup_{i \neq 0} L(T_i)$ are connected and contain x, y , it follows that there exist two (not necessarily distinct) chains between x and y , one containing elements of $L(T_0)$, the other containing elements of $\bigcup_{i \neq 0} L(T_i)$. By Lemma 1 these chains can be written as

$$(x = u_1^0, \dots, u_{n_0}^0 = b_1 = u_1^1, \dots, u_{n_{m-1}}^{m-1} = b_m = u_1^m, \dots, u_{n_m}^m = y)$$

and

$$(x = \tilde{u}_1^0, \dots, \tilde{u}_{\tilde{n}_0}^0 = b_1 = \tilde{u}_1^1, \dots, \tilde{u}_{\tilde{n}_{m-1}}^{m-1} = b_m = \tilde{u}_1^m, \dots, \tilde{u}_{\tilde{n}_m}^m = y),$$

where $u_j^i, \tilde{u}_j^i \in B_i$.

Consider some $i \in \{0, 1, \dots, m\}$. By Lemma 4, B_i is a guarding graph under the restriction of guards of G . Also by Lemma 4, the transactions $T_1^{B_i}, \dots, T_{n-1}^{B_i}$ all follow the EGLP on B_i (some of these transactions may be empty).

Let $T_{p_1}, T_{p_2}, \dots, T_{p_q}$ be the subsequence of T_1, T_2, \dots, T_n consisting of all the transactions T_j for which $L(T_j) \cap B_i \neq \emptyset$. Clearly,

$$\{\tilde{u}_1^i, \tilde{u}_{\tilde{n}_i}^i\} \subseteq \bigcup_{j=1}^q L(T_{p_j}^{B_i}),$$

and we claim that

$$L(T_{p_j}^{B_i}) \cap L(T_{p_{j+1}}^{B_i}) \neq \emptyset \quad \text{for } j = 1, \dots, q - 1.$$

Let then $j \in \{1, \dots, q - 1\}$, and assume by contradiction that $L(T_{p_j}) \cap L(T_{p_{j+1}}) \cap B_i$ (which is $L(T_{p_j}^{B_i}) \cap L(T_{p_{j+1}}^{B_i})$) is empty. Consider two cases:

- (1) $L(T_{p_j}) \cap L(T_{p_{j+1}}) \neq \emptyset$. Let $S_1 = L(T_{p_j})$, $S_3 = L(T_{p_{j+1}})$, and S_2 be a set of cardinality one containing any vertex in $L(T_{p_j}) \cap L(T_{p_{j+1}})$. Then by Lemma 3 the claim immediately follows.
- (2) $L(T_{p_j}) \cap L(T_{p_{j+1}}) = \emptyset$. Then $p_{j+1} > p_j + 1$. Let $S_1 = L(T_{p_j})$, $S_3 = L(T_{p_{j+1}})$, and $S_2 = \bigcup_{k=p_j+1}^{p_{j+1}-1} L(T_k)$. Then again by Lemma 2 the claim immediately follows.

By Lemma 3 there exists a transaction T^* following the EGLP on B_i such that

$$L(T^*) = \bigcup_{j=1}^q L(T_{p_j}^{B_i}),$$

and also by Lemma 2 there exists a chain between \tilde{u}_1 and $\tilde{u}_{\tilde{n}_i}$ entirely within

$$L(T_0^{B_i}) \cap \bigcup_{j=1}^q L(T_{p_j}^{B_i}),$$

and on which $T_0^{B_i}$ is piecewise two-phase. Let this chain be

$$(\hat{u}_1^i, \hat{u}_2^i, \dots, \hat{u}_{\hat{n}_i}^i),$$

where of course

$$\hat{u}_1^i = \tilde{u}_1^i = u_1^i, \quad \hat{u}_{\hat{n}_i}^i = \tilde{u}_{\tilde{n}_i}^i = u_{n_i}^i.$$

Repeating such construction for each i , we define a chain

$$(x = \hat{u}_1^0, \dots, \hat{u}_{\hat{\alpha}_0}^0 = b_1 = \hat{u}_1^1, \dots, \hat{u}_{\hat{\alpha}_{m-1}}^{m-1} = b_m = \hat{u}_1^m, \dots, \hat{u}_{\hat{\alpha}_m}^m = y)$$

that satisfies the lemma. \square

THEOREM 3. *The EGLP ensures serializability.*

PROOF. Assume by contradiction that the EGLP does not ensure serializability. Then without loss of generality let $\{T_0, T_1, \dots, T_{n-1}\}$ be a set of transactions following the EGLP on G such that

- (i) $T_0 \rightarrow T_1 \rightarrow \dots \rightarrow T_{n-1} \rightarrow T_0$ and there are no smaller cycles.
- (ii) $\sum_{i=0}^{n-1} |L(T_i)|$ is minimum for all sets of transactions satisfying condition (i).
- (iii) If M_i is the set of all vertices that T_i delays unlocking, then $\sum_{i=0}^{n-1} |M_i|$ is maximum for all sets of transactions satisfying conditions (i) and (ii).

These assumptions are hypotheses for Lemmas 6 and 7.

LEMMA 6. *Let $C_i = \{v | T_i \rightarrow^v T_{i+1}\}$, for $i = 0, \dots, n-1$. Then for every i , $|C_{i-1}| = 1$ or $|C_i| = 1$ (subscripts modulo n).*

PROOF. Let T_i be the sequence of instructions

$$I_1; I_2; \dots, I_i.$$

Assume now that the lemma is false. We shall write L for either LX or LS, when the distinction is not important. Let

$$\begin{aligned} j_1 &\triangleq \min k[(I_k = \langle T_i, L, v \rangle) | T_{i-1} \rightarrow^v T_i \text{ for some } v], \\ j_2 &\triangleq \min k[(I_k = \langle T_i, L, v \rangle) | T_i \rightarrow^v T_{i+1} \text{ for some } v], \\ j &\triangleq \max\{j_1, j_2\}. \end{aligned}$$

Delete from T_i every I_k of the form $\langle T_i, L, v \rangle$ for $k > j$ and its associated $\langle T_i, \text{UN}, v \rangle$. As the result, a new transaction T'_i is obtained. This transaction follows the EGLP on G , and

$$T_0 \rightarrow \dots \rightarrow T_{i-1} \rightarrow T'_i \rightarrow T_{i+1} \rightarrow \dots \rightarrow T_{n-1} \rightarrow T_0,$$

but $L(T'_i) \subset L(T_i)$ (proper subset), contradicting assumption (ii). \square

We note that by the construction in Lemma 6 it is clear that for the last vertex locked by T_i , say w , $w \in C_{i-1}$ or $w \in C_i$. Furthermore, in the previous case $C_{i-1} = \{w\}$, and in the latter case $C_i = \{w\}$. Informally, the last vertex locked by T_i "formed" one of the two sets, C_{i-1} or C_i .

LEMMA 7. *$L(T_i) - C_i \subseteq M_i$. Thus if C_i is a single vertex, T_i is two-phase.*

PROOF. Indeed, consider the case where some $w \in L(T_i) - \{v | T_i \rightarrow^v T_{i+1}\}$ is unlocked by T_i before some vertex of $L(T_i)$ is locked by T_i . Observe first the delaying of unlocking until the last locking instruction is executed is permissible under the EGLP. How can such delay of unlocking of w change the original schedule? As for each j , $T_i \not\rightarrow^w T_j$, it follows that delaying unlocking of w will not change the order of other locking and unlocking instructions. Then it immediately follows that the unlocking of w was delayed until after the last locking instruction was executed (assumption (iii)), and the lemma has been proved. \square

PROOF OF THEOREM 3 (CONTINUED). Obviously the following holds:

$$\emptyset \subset C_i \subseteq L(T_i) \cap L(T_{i+1}).$$

We will pick $u_i \in C_i, i = 0, \dots, n - 1$, such that

$$T_0 \rightarrow^{u_0} T_1 \rightarrow^{u_1} \dots \rightarrow^{u_{n-2}} T_{n-1} \rightarrow^{u_{n-1}} T_0.$$

If $|C_i| = 1$, then $C_i = \{u_i\}$; in this case we say that u_i was *uniquely chosen*. Observe that if $|C_i| > 1$, then by Lemma 6, $|C_{i-1}| = 1$ and $|C_{i+1}| = 1$.

Assume that $|C_i| > 1$. By Lemma 5, for any $u \in C_i$ there exists a chain between the uniquely chosen u_{i-1} and u , entirely within $L(T_i) \cap \bigcup_{j \neq i} L(T_j)$ and on which T_i is piecewise two-phase. We shall call such a chain *satisfactory* for $\{u_{i-1}, u\}$. Pick u_i such that the chain

$$(u_{i-1} = w_1, w_2, \dots, w_k = u_i)$$

is both satisfactory and of minimum length for $\{u_{i-1}, u_i\}$. In this case we shall say that u_i was *not uniquely chosen*. (A careful reader will notice in the sequel that if we consider chains between u and the uniquely chosen u_{i+1} , instead of u_{i-1} , the rest of the proof does not follow.)

For each i pick L'_i to be the set of the vertices of some minimum length chain satisfactory for $\{u_{i-1}, u_i\}$. Its existence is again guaranteed by Lemma 5.

Consider now transactions $T'_0, T'_1, \dots, T'_{n-1}$, where $T'_i = T_i^{L'_i}$. (Note that T'_i does not necessarily follow the EGLP on G .) We also obtain a new schedule by dropping from the original one every instruction of T_i referring to vertices in $V - L'_i$. In this schedule,

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

We will now show that each T'_i is two-phase (on $L(T'_i) = L'_i$).

If $|L'_i| = 2$, then the claim follows from the fact that T_i was piecewise two-phase on the chain (u_{i-1}, u_i) . For the case when $|L'_i| \geq 3$, we remind the reader that $L(T'_i) \subseteq \bigcup_{j \neq i} L(T_j)$. Consider two cases:

- (1) u_i was uniquely chosen. By Lemma 7, $M_i \supseteq L(T_i) - C_i = L(T_i) - \{u_i\}$. As in this case, C_i is a single vertex, T_i was two-phase on $L(T_i)$ and thus also two-phase on $L(T'_i)$. Consequently, T'_i was two-phase on its domain.
- (2) u_i was not uniquely chosen. In this case u_{i-1} was uniquely chosen (and in a certain sense " u_i was as close as possible to it"). Thus by the choice of u_i , for every $w, T_i \not\rightarrow^w T_j$ and $T_j \not\rightarrow^w T_i$ for $j \neq i$. As for such $w, w \in \bigcup_{j \neq i} L(T_j)$, we deduce that $w \in LS(T_i)$. It follows that $L(T'_i)$ is a subset of a single pitfall of T_i . Thus T_i and T'_i are two-phase on $L(T'_i)$.

We showed that if our extended guard protocol does not ensure serializability, then there exist a nonserializable schedule of two-phase transactions. These transactions do not necessarily follow the extended guard protocol, but the existence of such a nonserializable schedule would contradict the fundamental result of Eswaran et al. [2]. \square

5. Conclusion

We have presented new results concerning non-two-phase locking protocols which allow transactions to request both SHARED and EXCLUSIVE locks. Some results are applicable to general database systems and some to systems that are modeled by directed acyclic graphs.

We have presented two different approaches for handling SHARED locks. The first approach allows one to use a "correct" protocol which employs EXCLUSIVE locks only, with no restructuring of the protocol. This is done by restricting each

transaction to employ either only EXCLUSIVE locks or only SHARED locks, and by requiring that the sets of entities the transactions lock interact with each other in a specific manner. The second approach requires that each transaction follow the guard protocol with the restriction that the transaction be two-phase on its pitfalls.

It is interesting to compare our second approach with the other approaches previously published. All previous work on locking protocols which employ both SHARED and EXCLUSIVE locks was confined to variations of the two-phase concept. The one directly applicable to systems modeled by trees and DAGs is the intention mode locking protocol [3] (referred to in [9] as the warning protocol). Let us compare the warning protocol with ours. Both protocols allow the inclusion of SHARED and EXCLUSIVE locks. The warning protocol requires that the transaction be completely two-phase, while our protocol requires that a transaction be two-phase only on its pitfalls. The warning protocol requires that a transaction starts locking at the root, while our protocol allows the transaction to first lock any entity in the graph. In the warning protocol locking is done on a subgraph basis, while in our case each entity must be individually locked. Thus with the warning protocol for certain graphs the number of locks that need to be set may be fewer than the number of entities accessed by the locking transaction. However, it may also result in a substantial loss of potential concurrency.

ACKNOWLEDGMENTS. We wish to thank Al Crocker, Don Fussell, and especially Hank Korth and Dan Rosenkrantz for their helpful comments.

REFERENCES

1. COFFMAN, E., ELPHICK, M., AND SHOSHANI, A. System deadlocks *Comput. Surv.* 3, 2 (June 1971), 67-78.
2. ESWARAN, K.P., GRAY, J.N., LORIE, R.A., AND TRAIGER, I.L. The notions of consistency and predicate locks in a database system. *Commun. ACM* 10, 11 (Nov. 1976), 624-633.
3. GRAY, J.N. Notes on database operating systems. Res. Rep. RJ2188, IBM Research Laboratory, San Jose, Calif., Feb. 1978.
4. KEDEM, Z., AND SILBERSCHATZ, A. Controlling concurrency using locking protocols. In *Proc. 20th IEEE Symp. on Foundation of Computer Science* (San Juan, Puerto Rico, Oct. 1979), IEEE, New York, pp. 274-285.
5. ROSENKRANTZ, D.J., STEARNS, R.E., AND LEWIS, P.M., II System level concurrency control for distributed database systems. *ACM Trans. Database Syst.* 3, 2 (June 1978), 178-198.
6. SILBERSCHATZ, A., AND KEDEM, Z. Consistency in hierarchical database systems *J. ACM* 27, 1 (Jan 1980), 72-80.
7. SILBERSCHATZ, A., AND KEDEM, Z. A family of locking protocols for database systems that are modeled by directed graphs. *IEEE Trans. Softw. Eng.* 8, 6 (Nov. 1982), 558-562.
8. STEARNS, R.E., LEWIS, P.M., II, AND ROSENKRANTZ, D.J. Concurrency control for database systems. In *Proc. 17th IEEE Symp. on Foundations of Computer Science* (Houston, Texas, Oct. 1976), IEEE, New York, pp. 19-32.
9. ULLMAN, J.D. *Principles of Database Systems*. Computer Science Press, Potomac, Md., 1980.
10. YANNAKAKIS, M., PAPADIMITRIOU, C.H., AND KUNG, H.T. Locking policy: Safety and freedom from deadlock. In *Proc. 20th IEEE Symp. on Foundations of Computer Science* (San Juan, Puerto Rico, Oct. 1979), IEEE, New York, pp. 286-297.

RECEIVED MAY 1980, REVISED DECEMBER 1982, ACCEPTED JANUARY 1983