Beyond Two-Phase Locking

G. N. BUCKLEY AND A. SILBERSCHATZ

University of Texas at Austin, Austin, Texas

Abstract. Many database systems maintain the consistency of the data by using a locking protocol to restrict access to data items. It has been previously shown that if no information is known about the method of accessing items in the database, then the two-phase protocol is optimal. However, the use of structural information about the database allows development of non-two-phase protocols, called graph protocols, that can potentially increase efficiency. Yannakakis developed a general class of protocols that included many of the graph protocols. Graph protocols either are only usable in certain types of databases or can incur the performance liability of cascading rollback. In this paper, it is demonstrated that if the system has a priori information as to which data items will be locked first by various transactions, a new graph protocol that is outside the previous classes of graph protocols and is applicable to arbitrarily structured databases can be constructed. This new protocol avoids cascading rollback and its accompanying performance degradation, and extends the class of serializable sequences allowed by non-two-phase protocols. This is the first protocol shown to be always as effective as the two-phase protocol, and it can be more effective for certain types of database systems.

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

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Concurrency, consistency, database systems, deadlocks, hypergraphs, locking protocols, serializability, transactions

1. Introduction

A database system is described by a set of database items \( D \) and the set of all user programs that may access \( D \), denoted by \( \mathcal{P} \). A transaction in the system is a single execution of one user program in the set \( \mathcal{P} \). For a database system that imposes consistency constraints on the items in the database, it is usual to require that a single transaction executed alone on a consistent database leaves the database in a consistent state. This implies that any serial execution of a set of transactions preserves consistency. If the set of transactions is allowed to execute concurrently, the system must ensure that the outcome of a concurrent execution of a set of transactions is the same as an outcome of some serial execution of the same set of transactions. A system that guarantees this property is said to ensure serializability [3, 10].

This research was partially supported by NSF Grants MCS 81-04017 and MCS 82-14613 and by ONR Contracts N00014-80-K-0987 and N00014-83-K-0490.

Authors’ present addresses: G. N. Buckley, Department of Computer Science, SUNY, Stony Brook, NY 11794; A. Silberschatz, Department of Computer Sciences, University of Texas at Austin, 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.

\( \odot 1985 \) ACM 0004-5411/85/0400-0314 $00.75

Almost all systems that ensure serializability divide the database into separate
data items and control access to each item by use of a concurrency control. The
most common model for such a system uses a locking protocol. A locking protocol
requires that each transaction in the system lock a data item before any access to
that item, and that it unlock the item only after all accesses to that item have been
completed. Thus, a locking protocol may be viewed as a restriction on when a
transaction may lock and unlock each of the entities in set $\mathcal{D}$.

For systems with no restrictions on the order in which data items can be locked,
Eswaran et al. [3] have shown that it is necessary and sufficient to require that each
transaction follow the two-phase locking protocol to ensure serializability. In such
a system each transaction must lock all data items to be accessed before unlocking
any item. However, if the data items in the set $\mathcal{D}$ are restricted to be locked in
some predefined partial order, there are several non-two-phase protocols that allow
a transaction to unlock some data items before all desired items are locked. The
first and simplest protocol of this type is the tree protocol [9], used in databases
organized (logically or physically) as rooted trees. This protocol allows a transaction
to lock any data item first. Any subsequent item can be locked if the transaction
currently holds a lock on the father, and an item can be unlocked at any time.

Yannakakis has developed a characterization for all previous non-two-phase
protocols using exclusive locks [11]. Most of these protocols were proposed for
database systems organized as directed acyclic graphs [5-7, 12]. Since these proto-
cols incur no more run-time overhead than the two-phase protocol, they can
provide more concurrency when all transactions lock exactly the same items in
both the two-phase and non-two-phase protocols. This follows from the fact that
any executing transactions removed owing to serializability constraints do not
require removal of additional transactions in the database system. Depending on
the acyclic graphs derived for $\mathcal{D}$ and the constraints of the locking protocol, the
non-two-phase protocol may require more items to be locked in order to gain
access to the desired items. This makes a comparison between the two-phase
protocol and any of these non-two-phase protocols difficult.

The protocol developed by Yannakakis, the hypergraph protocol [11], executes
on an arbitrary database hypergraph [1], which is a generalization of the edge
structure of a simple directed graph. This protocol allows a transaction to lock
exactly the same number of items as the two-phase protocol, assuming that all
transactions entering the system are required to follow the precedence relations as
defined in the database hypergraph. Hence, the hypergraph protocol would be
directly comparable to the two-phase protocol if the execution time overhead were
the same. To ensure serializability, the hypergraph protocol may incur the addi-
tional overhead of cascading rollback. Cascading rollback occurs when the removal
of one unfinished transaction from the system forces the removal of other trans-
actions from the system. This is possible when the latter transactions have been
allowed to access data items updated by a previous transaction that cannot
complete. Cascading rollback can have serious performance consequences, since
transactions that have completed computation using the values of unlocked data
items will still need to be restarted. We do not discuss in this paper the effect of
transaction rollback due to user or environmental aborts, since these occurrences
would affect all non-two-phase protocols with the same frequency. We believe that
it is important to reduce or eliminate the need for transaction rollback and,
especially, cascading rollback as a method of ensuring serializability.

In this paper we introduce a new non-two-phase protocol, called the entry-point
protocol, which is free from cascading rollback in arbitrary hypergraphs and allows
a different set of serializable histories than all previous non-two-phase protocols. This new protocol uses two types of structural information regarding the data items. The first requires that $\mathcal{D}$ be organized as a directed hypergraph, and the second designates special status for each item that is the first data item locked by some transaction. These items, the entry points, must be known in advance. The entry-point protocol will be shown superior to the two-phase protocol in the sense that

(a) any transaction in the system can behave exactly as a two-phase transaction;
(b) there may be transactions that can unlock data items before all data items are locked, and yet lock the same number of data items as when run as a two-phase transaction. These transactions are guaranteed not to cause cascading rollback due to nonserializable sequences.

Thus, the entry-point protocol ensures that any database that requires a transaction to follow the precedence relations in $\mathcal{D}$ and that previously used a two-phase protocol will be at least as efficient using this new protocol. The use of the additional information (the entry points) extends the class of serializable histories allowed by non-two-phase protocols in hypergraphs that contain cycles.

In this paper, we also discuss the case of "spontaneous" two-phase transactions, where a transaction is allowed to dynamically create a new edge in the hypergraph. The simplest assumption to accommodate spontaneous transactions would be to require a complete graph, where there is a path between every two items in the database. This assumption reduces both the entry-point protocol and the hypergraph protocol to the two-phase protocol. Alternatively, it is possible to specify when a transaction can add new access paths to the hypergraph (either temporarily or on a permanent basis) during the operation of the system, so that a transaction can create a new or more direct path to data items yet to be accessed. We discuss this second option in Section 6.

The remainder of the paper is organized as follows. Section 2 presents conditions to prove that protocols operating in a database modeled as a directed hypergraph ensure serializability. Section 3 introduces the entry-point protocol and proves that the protocol ensures serializability and freedom from cascading rollback. Section 4 demonstrates how decomposing a graph into biconnected components can enhance the performance of the protocol, and Section 5 compares the efficiency of the new protocol with the two-phase protocol. Finally, in Section 6, we describe a scheme for dealing with spontaneous two-phase transactions that can create new access paths within the database hypergraph.

2. Basic Definitions

It is assumed that the set $\mathcal{D}$ is organized in the form of an arbitrary directed hypergraph consisting of data items and hyperedges. Each hyperedge has at least one data item in its tail and exactly one data item as its head. The use of hyperedges rather than simple edges allows easier specification of the access restrictions imposed on the database. A transaction that will access some data item $A$ for the first time must have previously accessed all data items in the tail of some hyperedge for which $A$ is the head. This restriction does not apply to the first data item accessed by a transaction. If each data item can be accessed independently, $\mathcal{D}$ is to be considered a complete graph.

**Definition 2.1.** A directed path (or path) from item $A_i$ to $A_j$ ($i \neq j$) exists if $A_i$ is the single item in the tail of a hyperedge with $A_j$ as its head, or a path exists from $A_i$ to every item in the tail of some hyperedge with $A_j$ as its head.
Definition 2.2. A set of items separate $A_i$ from $A_j$ (i ≠ j) if no path exists from $A_i$ to $A_j$ when all edges containing an item in the set are removed from the graph.

This definition of separate is weaker than the corresponding definition for undirected hypergraphs [11], but is useful in allowing earlier unlocking in graphs that contain cycles. This will be illustrated in the following section.

A transaction accesses data items according to the restrictions imposed by the hyperedges defined in $\mathcal{H}$, locking an item before the first access to that item, and releasing the lock after all accesses to the item are complete. If a transaction only reads an item's value, it may request a shared or exclusive lock (denoted S or X, respectively); otherwise, the transaction must request an exclusive lock. There may be several S locks on a data item, but a lock in X mode excludes other locks on the item. Hence, we say that a lock in X mode conflicts with other locks in X or S mode. We denote a shared (or exclusive) lock held on item $A$ as LS($A$) or LX($A$), respectively. The release of a lock on $A$ is denoted UN($A$).

Definition 2.3. A history $H$ is the chronological sequence of the lock and unlock operations of an arbitrary set of transactions $\mathcal{F} = \{T_0, \ldots, T_{N-1}\}$. We define the before relation $\rightarrow$ on a history $H$ of a set $\mathcal{F}$ of transactions as follows:

$T_i \rightarrow T_j \iff$ There exists a data item $A$ locked in the history, first by $T_i$ and then by $T_j$, such that at least one lock was in X mode.

Through the remainder of this paper, we define transaction behavior in terms of X and S locks rather than individual read and write operations. All serializability and rollback constraints assume that a transaction holding a lock in X mode may have altered the value of the locked data item.

Lemma 2.1. A protocol ensures serializability if and only if all possible executions for a set of transactions produces an acyclic $\rightarrow$ relation. This is a standard database theorem [8], and we do not prove it here.

If transaction $T_i$ holds a lock on a data item and $T_j$ issues a lock request for that item, where at least one lock is in X mode, then $T_j$ must "wait for" $T_i$. Deadlock occurs when a set of transactions creates a wait-for cycle. For the locking protocols discussed throughout this paper, rollback occurs when some transaction in deadlock must be removed to enable further progress of the other transactions in the same deadlock.

Definition 2.4. Cascading rollback occurs when the removal of one transaction, which has written values, requires the removal of additional transactions, which have since read those values. This occurs when $T_i$ locked $A$ in X mode, unlocked $A$, and $T_j$ locked $A$ before it was determined $T_i$ must be removed.

3. The Entry-Point Protocol

In this section, we describe the entry-point protocol and prove that it ensures serializability. When a transaction $T_i$ enters the database, the first data item it locks is its entry point, and is denoted E($T_i$). The set of all entry points for all transactions is denoted ENTRY($\mathcal{H}$). We assume this set is static for a particular database system and is known in advance.

To guarantee the absence of cascading rollback, a transaction must be able to finish execution once it has unlocked any data item. A transaction that never locks any item after it has unlocked an item trivially guarantees this, and we term this as operating in a two-phase mode. Alternatively, a transaction $T_i$ can unlock earlier
if it uses the entry-point concept to ensure that all transactions preceding \( T_i \) in the history will complete before \( T_i \) unlocks its first item. Once \( T_i \) has unlocked a data item, it guarantees serializability by having separated all paths from any entry point to the items it has yet to lock. Hence, any transaction entering the database after \( T_i \) and conflict locking with \( T_i \) must come later in the history.

We now present the entry-point protocol. A transaction can lock any item in the set \( \text{ENTRY}(\mathcal{R}) \) as its first element. For \( T_i \) to lock item \( A \in \text{ENTRY}(\mathcal{R}) \), the following must be true:

1. All vertices in the tail of some hyperedge with \( A \) as its head are locked by \( T_i \).
2. If \( T_i \) has unlocked some data item, \( T_i \) can lock \( A \) if and only if
   (a) condition 1 holds;
   (b) \( A \notin \text{ENTRY}(\mathcal{R}) \);
   (c) the items locked in X mode by \( T_i \) separate all \( A_j \in \text{ENTRY}(\mathcal{R}) \) from \( A \).

A transaction is said to operate in two-phase (2P) mode if it always follows condition (1); otherwise it is operating in non-two-phase (N2P) mode. Notice that a transaction operates in 2P mode until it requests a lock on some data item after it has unlocked some data item. A transaction that enters N2P mode must lock a sufficient set of items in X mode to ensure completion before it unlocks any data item. If it is required that all transactions follow the precedence relations defined by the hyperedges, then any transaction using the entry-point protocol and operating in 2P mode follows the original two-phase protocol. This follows from the fact that the chronological access sequence of the transactions operating in 2P mode is the same access sequence as the original user program operating under the precedence constraints.

We now present two simple examples to illustrate the behavior of the entry-point protocol. The first is an example of the tree protocol, and the second shows how the use of entry points permits transaction histories not allowed by the hypergraph protocol.

**Example 3.1.** Consider the database hypergraph of Figure 1 with \( \text{ENTRY}(\mathcal{R}) = \{A_1, A_2\} \), and the following partial history consisting of transactions \( T_1, T_2, \) and \( T_3 \):

\[
\begin{array}{ccc}
T_1 & T_2 & T_3 \\
\text{LX}(A_2) & \text{LX}(A_3) & \text{UN}(A_2) \\
\text{LX}(A_1) & \text{LX}(A_2) & \text{UN}(A_1) \\
\text{LX}(A_1) & \text{LX}(A_3) & \text{UN}(A_3)
\end{array}
\]
Beyond Two-Phase Locking

This is an example of the original tree protocol that uses only X locks. Allowing $T_1$ to operate in N2P mode provides additional concurrency without the expense of extra locks or the possibility of cascading rollback in the system.

**Example 3.2.** Consider the database hypergraph of Figure 2, where $ENTRY(\mathcal{H}) = \{A_1, A_2\}$, and the following partial history of transactions $T_1$ and $T_2$:

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LX($A_2$)</td>
<td>LX($A_1$)</td>
</tr>
<tr>
<td>LX($A_3$)</td>
<td>LX($A_2$)</td>
</tr>
<tr>
<td>UN($A_2$)</td>
<td>UN($A_1$)</td>
</tr>
<tr>
<td></td>
<td>LX($A_4$)</td>
</tr>
</tbody>
</table>

$T_1$ could not lock $A_4$ under any of the previously known non-two-phase protocols, including the hypergraph protocol, since it is possible that a third transaction could lock $A_4, A_1$, and thus produce a nonserializable history. The use of entry points guarantees that all transactions start only at $A_1$ or $A_2$.

We now prove that the entry-point protocol ensures serializability and freedom from cascading rollback. We term a set of transactions related by $\rightarrow$ as a dependency chain, and abbreviate the representation of $T_i \rightarrow T_{i+1} \rightarrow \cdots \rightarrow T_j$ as $T_i \rightarrow T_j$.

**Definition 3.1.** The time at which $T_i$ unlocks its first data item is denoted $\tau_i$. $\tau_i$ may not exist if $T_i$ is in deadlock and must be removed from the system.

**Lemma 3.1.** If $T_i$ locked $A$ before $T_j$ issued a lock request on $A$ in conflicting mode (at least one X lock on $A$), and $\tau_i$ and $\tau_j$ exist, then $\tau_i < \tau_j$.

**Proof.** If $T_j$ issues a lock request on $A$ in 2P mode, then it will not unlock any item until it locks $A$, which is after $T_i$ unlocks $A$, and hence $\tau_i < \tau_j$. We now consider the case where $T_j$ locks $A$ in N2P mode, which implies $A \not\in E(T_i)$ and $A \not\in E(T_j)$. In order for $T_i$ to lock $A$, it must lock at least one path from $E(T_i)$ to $A$, from the rules of the protocol. Without loss of generality, select one path from $E(T_i)$ to $A$, termed $Q$, which $T_i$ locked in order to access $A$. The proof is by induction on $k$, the number of items in the path $Q$.

$k = 1$. $A = E(T_i)$ and, by the protocol, $T_j$ must lock $A$ in 2P mode.

$k > m$. The set of locks held in X mode by $T_j$ must separate every path from $E(T_i)$ to $A$, from the protocol. Since the set of locks that separate $E(T_i)$ from $A$ on path $Q$ were locked by $T_i$ before $T_i$ locked $A$ and are now locked by $T_j$ in X mode, $T_i$ and $T_j$ conflict on this set of locks. The number of items in the path $Q$ from $E(T_i)$ to any item in this set is at most $m$, and hence the inductive hypothesis holds.

**Theorem 3.1.** The entry-point protocol with shared and exclusive locks ensures serializability.

**Proof.** Assume, by contradiction, that there is some minimal cycle $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n \rightarrow T_1$ on data items $A_1$ to $A_n$. From the definition of the $\rightarrow$ relation,
If transaction \( T_i \) locked \( A_i \) before \( T_{i+1} \) issued a lock request on \( A_i \) in conflicting mode. This implies there is a cycle \( \tau_1 < \ldots < \tau_n < \tau_1 \), a logical contradiction. □

**Theorem 3.2.** The entry-point protocol does not ensure freedom from deadlock.

**Proof.** The two-phase locking protocol is a special case of the entry-point protocol that allows deadlocks. □

**Theorem 3.3.** The entry-point protocol ensures freedom from cascading rollbacks.

**Proof.** Assume, by contradiction, that some transaction \( T_k \) has unlocked some data items and then waits for a transaction \( T_j \), which is now in a deadlocked set of transactions. From the wait-for relation, each transaction in this set has issued a lock request for a data item presently locked in a conflicting mode. From Lemma 3.1, each transaction in the deadlocked set cannot unlock any item before the previous transaction unlocks some data item, and hence no transaction has unlocked any data item. Since \( T_j \) has not unlocked a data item, by Lemma 3.1, \( T_k \) has not unlocked a data item, which contradicts our assumption. □

Theorem 3.3 guarantees that any transactions that must be rolled back only affect the transaction in question and do not affect the rest of the transactions in the system. This is necessary to compare the protocol to the two-phase protocol, since that protocol also does not need to use cascading rollback to maintain serializability.

4. *The Entry-Point Protocol with Biconnected Components*

A hypergraph can be separated into components with a single undirected path between components. If there is only one path to a data item, every transaction must follow that path, and therefore one can remove the restriction that a transaction must separate all entry points in the database with a path to some item yet to be locked before unlocking. This extends the protocol to allow additional concurrency. We separate the hypergraph into biconnected components, and require that a transaction operating in N2P mode must follow the rules of the N2P protocol only within a biconnected component.

An undirected hyperedge \( H \) can be defined by the set of data items in the edge, with no distinction between head and tail.

**Definition 4.1.** An undirected hyperpath between \( A_i \) and \( A_j \) is the sequence of hyperedges \( H_0 \ H_1 \cdots H_k \), where \( A_i \in H_0, A_j \in H_k, \) and \( H_x \cap H_{x+1} \neq \emptyset, \) \( 0 \leq x < k. \)

**Definition 4.2.** Items \( A_i \) and \( A_j \) are members of the same biconnected component if there exist at least two undirected hyperpaths from \( A_i \) to \( A_j \) with only items \( A_i \) and \( A_j \) in common.

Each transaction now has several entry points, one for each biconnected component that it traverses. We extend the definition of \( \text{ENTRY}(E) \) for each component \( E \), where the first item locked by some transaction in a biconnected component is an element of \( \text{ENTRY}(E) \) for that component. These are the items that have an arc from a separate biconnected component, together with the items for transactions entering the database system in this biconnected component. To lock the first item \( A \) in a biconnected component, a transaction must have a lock on the items in the
Beyond Two-Phase Locking

tail of the hyperedge to \( A \) from a separate biconnected component. To ensure serializability among all biconnected components, the protocol becomes the following:

1. \( T_i \) can lock item \( A \) in biconnected component \( \mathcal{G} \), where \( A \neq \) the first \( E(T_i) \) for the entire database, if and only if all vertices in the tail of some hyperedge with \( A \) as its head are locked by \( T_i \).
2. If \( T_i \) has unlocked any item in \( \mathcal{G} \), then it has separated all \( E(T_k, \mathcal{G}) \) with a path to \( A \).
3. If \( T_i \) has unlocked any item in \( \mathcal{G} \) where \( E(T_k, \mathcal{G}^3) \) has a path to some item in \( \mathcal{G}^2 \) not separated by locks in \( X \) mode held by \( T_i \), or if \( \mathcal{G}^2 \) is the same biconnected component as \( \mathcal{G}^3 \), then \( T_i \) has separated all \( E(T_k, \mathcal{G}) \) from any item not previously locked by \( T_i \) in \( \mathcal{G} \).
4. After \( T_i \) locks its first item in \( \mathcal{G} \), it cannot unlock any item in any biconnected component until all \( E(T_k, \mathcal{G}) \) are separated from items \( T_i \) will yet lock in \( \mathcal{G} \).

Rule 4 implies that \( T_i \) cannot unlock any item until it separates the first item it locked in the database from any items it has yet to lock.

We now demonstrate that the dependency relations created between transactions in a biconnected component still ensure serializability when combined with the dependency relations created in all other biconnected components.

**Theorem 4.1.** The entry-point protocol defined on biconnected components ensures serializability.

**Proof.** Assume, by contradiction, that there exists a smallest cycle \( T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_N \rightarrow T_1 \). If all transactions operate entirely in 2P mode over the section of the graph involved in the cycle, then deadlock occurs, which contradicts the assumption, and the theorem holds. If all transactions do not operate entirely in 2P mode over the section of the graph in the cycle, then we show that a smallest cycle cannot exist.

**Case 1.** Consider a cycle of two transactions, where \( T_1 \rightarrow T_2 \) in one section of the graph and \( T_2 \rightarrow T_1 \) in another section of the graph. If both sections are in one biconnected component, this contradicts Theorem 3.1, and the theorem holds. If they are in separate biconnected components, then by Lemma 3.1, \( T_1 \) must unlock an item in \( \mathcal{G}_1 \) before \( T_2 \), and \( T_2 \) must unlock before \( T_1 \) in \( \mathcal{G}_2 \). Without loss of generality, let us assume the first transaction to unlock an item is \( T_1 \). Since \( T_2 \) has not yet unlocked an item, \( T_1 \) has not yet locked its item in \( \mathcal{G}_2 \) that \( T_2 \) conflict locked first, and hence \( T_1 \) must separate the path of \( E(T_2, \mathcal{G}_1) \) from any item in \( \mathcal{G}_2 \). If \( T_2 \) had already locked a path between \( \mathcal{G}_1 \) and \( \mathcal{G}_2 \), then it had to unlock some item to allow \( T_1 \) to separate the path, which contradicts the assumption that \( T_1 \) is the first to unlock an item, and the theorem holds. If \( T_2 \) had not unlocked an item and the path is from \( \mathcal{G}_1 \) to \( \mathcal{G}_2 \), then \( T_2 \) cannot lock a path to an item in \( \mathcal{G}_2 \) until \( T_1 \) unlocks in \( \mathcal{G}_2 \), which is after \( T_2 \) unlocks in \( \mathcal{G}_2 \), a logical contradiction. If the path is from \( \mathcal{G}_2 \) to \( \mathcal{G}_1 \), the \( T_1 \) had earlier locked a path from \( \mathcal{G}_2 \) to \( \mathcal{G}_1 \), in order to lock items in \( \mathcal{G}_1 \). If \( T_2 \) locks some item in conflicting mode with \( T_1 \) in \( \mathcal{G}_1 \) before it unlocks an item in \( \mathcal{G}_2 \), this contradicts the protocol since \( T_2 \) has a path to both an unlocked item of \( T_1 \) and has a path to a yet-to-be-locked item of \( T_1 \). If \( T_2 \) unlocks some item in \( \mathcal{G}_2 \) before locking in \( \mathcal{G}_1 \), then it has to separate the path of \( T_1 \), which implies \( T_1 \) had unlocked some item, and again \( T_2 \) has a path to an unlocked and yet-to-be-locked item of \( T_1 \), which contradicts the protocol. In both cases, the theorem holds.
Case 2. If the cycle contains at least three transactions, we show that there is always a smaller cycle. The graph on which the cycle is created is an undirected tree of biconnected components, where a transaction \( T_i \) enters the graph at some component and traverses down the branches of the tree-locking items until it obtains all conflict locks. If the graph is one biconnected component, this contradicts Theorem 3.1, and the theorem holds. If the graph is composed of several biconnected components, let us first consider the case where at least one transaction \( T_j \) in N2P mode separates component \( C_1 \) from \( C_2 \), where items in both \( C_1 \) and in \( C_2 \) are locked either by some transaction \( T_m \) not conflict locking with \( T_j \) in the cycle, or by both \( T_{j-1} \) and \( T_{j+1} \). This creates a smaller cycle in the following way. If transaction \( T_m \) had not previously conflict locked with \( T_j \), this creates one of two smaller cycles. The smaller cycle can be composed of transactions \( T_m, T_j, \) and the transactions between the two, or composed of all transactions except the transactions between \( T_j \) and \( T_m \). If \( T_{j-1} \) and \( T_{j+1} \) are the only transactions that conflict lock with \( T_j \), then one of three possibilities hold. First, \( T_m \) does separate exactly one transaction \( T_i \) in the cycle, and \( T_i \) must precede or follow \( T_m \) in this component, by Lemma 3.1. Since it follows the same path as \( T_m \), it either causes deadlocks, or conflict locks with the transaction that follows or precedes \( T_m \) in the cycle, creating a smaller cycle, and the theorem holds for cycles of length greater than 3. Third, if \( T_m \) separates two or more transactions, at least two of them must conflict lock with \( T_m \) to produce a cycle that includes both \( T_{j-1} \) and \( T_{j+1} \). If \( T_m \) is either \( T_{j-1} \) or \( T_{j+1} \), then this forms a cycle consisting of the three transactions, or the original cycle excluding \( T_j \). In either case, the theorem holds for transaction cycles of length greater than 3. In an original cycle of only three transactions, seven of the eight dependency combinations created on the items \( T_j \) uses to separate \( C_1 \) from \( C_2 \) form a cycle of length 2, which contradicts Case 1, and the eighth creates a cycle in a single biconnected component, which contradicts Theorem 3.1. In all cases, the theorem holds. If all N2P transactions only separate one other transaction with which they conflict lock, then either the additional dependency creates a cycle of length 2, which contradicts Case 1, or it is the same dependency already in the cycle. In this case, Lemma 3.1 states that each of these transactions must wait to unlock until the previous transaction unlocks an item in the biconnected component, and since any remaining transactions are 2P over the graph involved in the cycle, deadlock results, which contradicts the assumption, and the theorem holds.  

**Theorem 4.2.** The entry-point protocol defined on biconnected components is free from cascading rollback.

**Proof.** At the time \( T_i \) separates its first data item locked in the database from all items yet to be locked, it has not unlocked any item, and therefore has a path locked to each of these items in X mode. Hence, no other transaction can have separated any of \( T_i \)'s paths, and a partial order exists among transactions operating in different biconnected components, since each must follow the unique path between biconnected components. This implies that there is no deadlock in different components among the set of transactions whose first items locked were locked in different biconnected components. Within a biconnected component,
Theorem 3.2 guarantees that \( T_i \) will not get into cascading rollback for that component. If \( T_i \) operates between biconnected components, \( T_i \) has not unlocked any item previously locked in X mode between the step in which \( T_i \) locked its entry point in that component until after the step where \( T_i \) separated all entry points with paths to items it will yet lock. \( T_i \) will not cause cascading rollback in any other component, and the theorem holds.

For a simple example of the utility of the extension for biconnected components, we contrast the original protocol and the extension just presented.

Example 4.1. Consider Figure 1, with \( \text{ENTRY}(\emptyset) = \{A_1, A_3\} \), and transactions \( T_1 \) and \( T_2 \) operating in N2P mode. If the hypergraph is divided into biconnected components, we get exactly the same behavior as the tree protocol referenced earlier. Using biconnected components, the entry points become \( \text{ENTRY}(\emptyset_1) = \{A_1\}, \text{ENTRY}(\emptyset_2) = \{A_2\} \), and \( \text{ENTRY}(\emptyset_3) = \{A_3\} \), and \( T_1 \) can unlock item \( A_1 \) any time after it locks item \( A_2 \) in X mode.

5. Comparison with the Two-Phase Protocol

In order to compare any two protocols, we must assume the database system has defined the precedence relations in the hypergraph of \( \mathcal{D} \) to be the same for all concurrency controls. Such precedence relations are often due to logical addressing of data items, as in IMS [4], or to the structure of the physical storage of items. Consequently, the access sequence of each user program in \( \mathcal{D} \) must satisfy the precedence relations in \( \mathcal{D} \) under any protocol.

Corollary 5.1. The entry protocol is at least as efficient as the two-phase protocol.

Proof. The graph is constructed in such a manner that each transaction locks only the data items required by a two-phase transaction. Those transactions whose items include one item that is the dominator of all data items to be locked may run in N2P mode, with possible increase in concurrency for the system. There is no additional run time costs for this classification.

We now discuss the conditions under which earlier unlocking can be guaranteed, and when such an increase should be expected. Let us first point out that transactions operating entirely in N2P mode on trees reduce to the tree protocol, since every data item in a tree is a biconnected component.

We begin by considering the hypergraph of Figure 1. Let transactions \( T_1, T_2, T_3 \) be such that \( E(T_1) = E(T_2) = A_1 \), and \( E(T_3) = A_3 \). \( T_1 \) and \( T_2 \) each consecutively access \( A_1, A_2, \) and \( A_3 \) in X mode for 10 time units apiece, whereas \( T_3 \) accesses \( A_3 \) in X mode for 30 time units. The minimum execution time if the transactions run in N2P mode is 50 units, whereas the minimum execution time if the transactions run entirely in 2P mode is 60 units.

We now discuss which classes of transactions can be guaranteed an increase in concurrency for an arbitrary hypergraph. It is a simple generalization from the preceding example, where two transactions \( T_j \) and \( T_k \) wait on transaction \( T_i \) to release a lock. Consider a graph that contains data items \( A_m \) and \( A_n \), with a path from \( A_m \) to \( A_n \). The entry-point protocol can guarantee an increase in concurrency over the two-phase protocol if the following conditions hold:

- \( T_j \) will access item \( A_n \) only after completing access to data item \( A_m \). The items locked in X mode by \( T_j \) before accessing \( A_n \) separate \( A_m \) and all entry points from \( A_n \).
There is a positive probability that $T_j$ is waiting to access item $A_n$ while $T_i$ continues to access $A_n$.

There is a positive probability that $T_k$ is waiting to access $A_m$ while $T_j$ is waiting on $A_n$.

In this case, the entry-point protocol would allow $T_j$ to release its lock on $A_m$ and decrease the waiting time for $T_k$.

Concurrency is also increased even when data items that are not referenced must be locked in order to separate $A_m$ from $A_n$. If no transactions presently executing in the system wait on these additional data items, then concurrency will still be increased as long as the overhead for the additional locks is not significant. Once these additional data items to be locked begin delaying the progress of other transactions in the system, the time needed to maintain locks on those additional data items decreases the additional concurrency gained by unlocking $A_m$ earlier than in 2P mode. This comparison is not simple, since unlocking $A_m$ may open a path to a large portion of the database, whereas the locks on the additional data items may simply prohibit access to a few items. This trade-off will depend on both the frequency and behavior of the transactions, as well as on the design of the hypergraph.

6. Spontaneous Transactions

In the previous section we assumed that the hypergraph was static, so that any transaction entering the database must follow predefined paths from its entry point to any other data item it will access. In this section we relax this requirement and allow a transaction to temporarily add new access paths in the hypergraph. In this way, the database performance can be improved by allowing a transaction to create a shorter path between data items and hence lock fewer items. Also, it allows for the introduction of “spontaneous” two-phase transactions, which access a set of data items that may not be reachable from its entry point in the static hypergraph.

The restriction on entry points ensures that all transactions preceding a transaction in N2P mode will not be required to roll back as a result of the serializability constraints. A spontaneous transaction $T_i$, which will add a hyperedge from item $B$ to item $A$, must determine whether the transactions currently operating in N2P mode will be affected by the addition of this new edge. A transaction $T_j$ operating in N2P mode may be affected in one of three ways:

1. The edge to be added merges two biconnected components into one component, where one original component contained $A$ and the other contained items already accessed by $T_j$.
2. The added edge creates a path from $E(T_j)$ to a data item previously unreachable from $E(T_j)$.
3. The added edge no longer allows $T_j$ to acquire locks on all items in the tail of every hyperedge to a data item.

To avoid these problems, the database system must ensure that no such $T_j$ is presently executing. This can be accomplished by maintaining two lists:

1. The N2P list includes all transactions operating in N2P mode and their entry points.
2. The request list maintains two types of requests, those to add access paths and those to enter N2P mode. This list is kept in chronological order.
These lists need be updated only twice during the execution of a transaction that operates in N2P mode. This updating must take place just before the transaction unlocks its first lock and enters N2P mode, and when the transaction in N2P mode completes.

A spontaneous two-phase transaction $T_i$ may request that a new hyperedge be added from item $B$ to item $A$ during its execution. This request is inserted in the request list and is delayed until there is no path from $E(T_j)$ to $B$ or $A$, where $T_j$ is any transaction in either the N2P list or in the request list with a prior request for entering N2P mode. When all such $T_j$ have completed, the system adds the new hyperedge from $B$ to $A$; this hyperedge, however, can only be used by transaction $T_i$. This new hyperedge does not affect any N2P transaction operating in the system, since every transaction $T_j$ in the N2P list has no path from $E(T_j)$ to $B$ or $A$, and hence $T_j$ cannot wait on a transaction with access to either of these data items.

Any other transaction $T_k$ operating in the database can still access all data items that were originally accessible from $E(T_k)$, since no access paths have been deleted. If $T_k$ now requests entry into N2P mode after some new access path from $B$ to $A$ has been added (or is an earlier hyperedge request yet to be added), $T_k$ can still enter N2P mode immediately if there is no path from $E(T_k)$ to $B$ or $A$ (this follows from the discussion in the previous paragraph). If there is a path from $E(T_k)$ to $B$ or to $A$, it is sufficient to let $T_k$ continue in 2P mode until transaction using the temporary hyperedge has completed or been rolled back. When this occurs, the database system will remove the temporary hyperedge and allow $T_k$ to enter N2P mode. If the hyperedge from $B$ to $A$ is to be permanent, the database hypergraph must be updated to determine which entry points have paths to data items previously unreachable, and also to redetermine which items can be locked in N2P mode from a particular entry point. This may prohibit other transactions in the N2P list from entering N2P mode, and can be quite expensive at run time depending on the structure of the graph.

7. Conclusion

We have shown how additional structural information imposed on a database hypergraph can be used to derive the first general protocol that is always as efficient as the two-phase locking protocol. This follows from the fact that

(1) any transaction that must be removed owing to deadlock will not affect other transactions, and

(2) all transactions can be run in 2P mode unless the set of data items locked in N2P mode is identical to the set of items locked in 2P mode.

The transactions that can run in N2P mode and yet lock the same number of data items can take advantage of the organization of the database to enhance concurrency. If the database system has sufficient information about the set of transactions to determine the entry points, the system can use this information to enhance concurrency by using our newly proposed protocol. This concept of preanalyzing the behavior of transactions has been used in previous systems, including the distributed database system of SDD-1 [2].

ACKNOWLEDGMENT. The authors would like to thank the referees of this paper for their helpful suggestions.
REFERENCES