On the Heterogeneous Guard Locking Protocol

The heterogeneous guard locking protocol is a general non-two-phase locking protocol that is applicable to database systems organized as rooted directed acyclic graphs. The protocol is one of the few serializable protocols that employs both shared and exclusive locks and yet remains deadlock free. In this note we present the protocol and prove that it ensures freedom from deadlock. This, in turn implies that consistency of the database can be maintained without resorting to transaction rollback.

1. Introduction

Many database systems ensure serializability by dividing the database into entities, and restricting access to an entity by use of a concurrency control scheme. The most common model for such a system involves the notion of a locking protocol. Each transaction which executes in the system must lock an entity before it wishes to access that entity, and unlock the entity after all accesses are complete. A locking protocol may thus be viewed as a set of rules defining the allowable sequences of lock and unlock instructions which may appear in a transaction. A transaction may hold either an exclusive (X) or a shared (S) lock on a data item. Several S mode locks may be held simultaneously on a single data item, but a lock in X mode prohibits any other lock on that item. An X mode lock on a data item permits the transaction holding that lock to read and modify the item, whereas an S mode permits only the reading of the item.

The first useful locking protocol developed was the two-phase locking protocol, which states that a transaction is not allowed to lock a data item after it has unlocked any item. Yannakakis has shown that for systems that allow data items to be accessed in any order, two phase locking is the best possible locking policy. Silberschatz and Kedem have shown that if one has a priori knowledge as to how the entities of the database are organized (logically or physically), one may design non-two-phase locking protocols which assure serializability and deadlock freedom. Since then a number of new non-two-phase locking protocols have been developed which potentially allow more concurrency than two-phase protocols.

One of the more general non-two-phase locking protocols is the guard protocol, used for databases modelled as directed acyclic graphs. By proper choice of sets of vertices in the guards, most previously proposed non-two-phase protocols (e.g. tree, majority and DAG) can be obtained as special cases of that protocol. The basic protocol restricts a transaction to employ only X mode locks. It was shown that this protocol assures deadlock freedom. In Ref. 11, a general result concerning extension of all protocols that employ X locks only to also employ S locks was presented. In general, this extension does not preserve deadlock freedom, even though the original protocol may have been deadlock free. In this note we show how this extension can be applied to the guard protocol to produce a new protocol, called the heterogeneous guard protocol. We show that the heterogeneous protocol ensures serializability and maintains deadlock freedom, implying that no rollbacks are necessary in order to maintain the consistency of the database.

2. The heterogeneous guard protocol

The database is modelled by a rooted acyclic directed graph, where the nodes of the graph correspond to the data items and the edges correspond to access paths between data items. For each vertex \( v \in V \) (except the root) we associate a non-empty set of pairs (subguards):

\[ \text{guard}(v) = \{ (A'_i, B'_i), \ldots, (A'_{m}, B'_{m}) \} \]

satisfying the conditions:

1. \( \emptyset \neq B'_i \subseteq A'_i \subseteq V \).
2. If \( v \in A'_i \), then \( u \) is a father of \( v \).
3. \( A'_i \cap B'_i \neq \emptyset \) for every \( i \) and \( j \).

The heterogeneous guard protocol supports two types of transactions: read-only transactions and update transactions, denoted \( R_i \) and \( U_i \), respectively. An update transaction may only issue X locks, whereas a read-only transaction may only issue S locks. These transactions operate on a guarded graph \( G \) by the following rules:

1. Update transactions must lock the root of the DAG first. Read-only transactions may lock any vertex first.
2. A transaction \( T \) may lock any subsequent vertex \( v \) only if \( T \) has not previously locked \( v \), and there exists a subguard \( (A'_i, B'_i) \) such that \( T \) currently holds a lock on all vertices in \( B'_i \), and it has locked (and possibly unlocked) all vertices in \( A'_i - B'_i \).
3. A vertex may be unlocked at any point in time.

A simple example of this protocol is given using Fig. 1, where the sets of guards for an item are written beside that item. Consider two transactions \( T_1 \) and \( T_2 \), where \( T_1 \) will access \( b \) and \( d \), and \( T_2 \) will access \( a \) and \( c \). The guard assignment given allows \( T_1 \) to enter the database at \( b \) and lock only items \( b \) and \( d \), whereas \( T_2 \) enters at item \( a \) and locks both \( b \) and \( c \) before locking \( d \). For this example and this guard assignment, the transactions lock the minimum number of data items and still follow the restrictions of the guard protocol.

Figure 1

3. Properties of the protocol

We now prove that the heterogeneous guard protocol ensures the consistency of the database system.

Theorem 1

The heterogeneous guard protocol ensures serializability.

Proof. The serializability proof follows from the fact that the guard protocol with \( X \) locks only ensures serializability and by applying the general result given in Ref. 11 to that protocol.

We will now show that consistency is maintained without resorting to transaction rollback. This will be done by proving that our protocol ensures deadlock freedom. To do so we must first establish several definitions and lemmas.

Definition 1

We say vertex \( v_i \) is above \( v_j \) if there exists a path from \( v_i \) to \( v_j \) in the graph. An acyclic graph admits a partial ordering of vertices, and thus the above relation is reflexive and antisymmetric.

Definition 2

Let \( L(T_i) \) be the set of data items locked by transaction \( T_i \).

Definition 3

Let \( T_i \) be an arbitrary transaction. We will denote the first vertex locked by \( T_i \) by \( F(T_i) \).

Lemma 1

Let \( T_i \) and \( T_j \) be two transactions such that \( L(T_i) \cap L(T_j) \neq \emptyset \). If \( F(T_i) \) is above \( F(T_j) \), then \( F(T_j) \) is the first vertex locked in \( L(T_i) \cap L(T_j) \) (by either transaction).

Proof. Assume by contradiction that some other vertex \( v \neq F(T_i) \) is the first vertex locked in \( L(T_i) \cap L(T_j) \). If \( v = F(T_j) \), then, from the assumption, \( F(T_j) \neq F(T_i) \), and there must be a path from \( F(T_j) \) to \( F(T_i) \) (implied by the guard protocol). This contradicts the assumption that \( F(T_i) \) is above \( F(T_j) \). If \( v \neq F(T_i) \), then \( F(T_j) \) exists an \( A'_i \) and \( B'_i \) locked by \( T_j \) and \( T_i \), respectively. But since the guard protocol stimulates \( A'_i \cap B'_i \neq \emptyset \), this contradicts the assumption that \( v \) was the first vertex locked in \( L(T_i) \cap L(T_j) \).

Lemma 2

Let \( T_i \) and \( T_j \) be two transactions such that \( L(T_i) \cap L(T_j) \neq \emptyset \). If \( F(T_i) \) is above \( F(T_j) \), and at least one transaction is an update transaction. The transaction which locks some \( v \in L(T_i) \cap L(T_j) \) first locks \( F(T_i) \) first.

Proof. By induction on the longest path length \( p \) from \( F(T_i) \) to a vertex \( v \).

1. \( p = 0 \). Trivial.
2. \( p > 0 \). Without loss of generality, assume \( T_j \) locks \( v \) first. At that time \( T_i \) had a lock on some \( B'_i \), from the heterogeneous guard protocol. Since \( X \) and \( S \) locks are mutually exclusive, the protocol requires that \( A'_i \cap B'_i \neq \emptyset \), and \( T_j \) can only lock a vertex once, \( T_i \) locked the vertices in \( A'_i \cap B'_i \) first. Each of these vertices is a father of \( v \), and any
path from $F(j)$ to a father of $e$ is at least one arc shorter than the longest path from $F(j)$ to $e$. From the inductive assumption, $T_e$ must lock $F(j)$ first.

Theorem 2

The heterogeneous guard protocol ensures freedom from deadlocks.

Proof. Assume by contradiction that a deadlock exists and there is a cycle $T_1, T_2, \ldots, T_n$, where $T_i + 1$ waits for $T_i$ to release a data item. We will show that this reduces to the case where the cycle consists of update transactions only, each of which claims to lock the root before the next transaction in the cycle. The cycle consists of sequences of the form $U_i, U_{i+1}, U_{i+2}, \ldots, U_n, U_1, U_2, \ldots, U_{i-1}$, since ready-only transactions need not wait for each other. For the first sequence, Lemmas 2 implies that $U_i$ locks the root of $G$ before $U_{i-1}$. From Lemma 2 and from the fact that the root of $G$ is above all vertices in $G$, the second sequence maintains that $U_i$ locks $F(i)$ before $R_i$ and $R_i$ locks $F(i)$ before $U_{i-1}$. Since transactions can only lock a root once, $U_{i-1}$ locks $F(i)$ before $U_i$. Hence $U_{i-1}$ locks the root of $G$ before $U_i$, from Lemma 2. This results in a cycle consisting of update transactions each claiming to lock the root before the subsequent update transaction, which contradicts the restriction that each transaction can only lock a vertex once.

4. Conclusion

We have presented the heterogeneous guard protocol and have shown that it ensures serializability and deadlock-freedom. We have thus constructed a general non-two-phase protocol using both shared and exclusive locks that is applicable to the database systems organized as directed acyclic graphs. The deadlock-freedom ensures that the consistency of the database can be maintained without transaction rollback.

Acknowledgements

This research was supported in part by the US Office of Naval Research under Contract N00014-80-K-0987, and by the US National Science Foundation under Grant MCS 81-4017.

GAEL N. BUCKLEY
ABRAHAM SILBERSCHATZ
Department of Computer Sciences
The University of Texas
Austin, Texas 78712, USA

References


Received March 1983