Ensuring Transaction Atomicity in Multidatabase Systems*

Sharad Mehrotra†
Rajeev Rastogi†
Yuri Breitbart‡
Henry F. Korth§
Avi Silberschatz†

1 Introduction

This paper is concerned with the problem of ensuring the atomicity of transactions in a multidatabase system (MDBS) environment. An MDBS, built on top of pre-existing centralized local database management systems (DBMSS), is a facility that allows users to access and update data located at remote sites. An MDBS environment supports two types of transactions:

- **Local transactions**, those transactions that execute outside the control of the MDBS. Local transactions access data items managed by only a single DBMS and do not access data items managed by remote DBMSs.

- **Global transactions**, those transactions that execute under the MDBS control. Global transactions may access data managed by remote DBMSs. A global transaction consists of a number of subtransactions, each of which executes as an ordinary local transaction at one of the participating DBMSs.

Each local DBMS controls the concurrent execution of transactions to ensure both the atomicity and the isolation properties of transactions. A transaction is atomic if in case it commits, then all its effects appear in the database, or else if it aborts, then no effects appear in the database. This property is typically ensured through the use of write-ahead logging scheme and further by requiring that the schedules resulting from the concurrent execution of transactions be recoverable [BHG87]. A transaction is said to execute in isolation if it does not "see" the partial effects of other concurrently executing transactions. This property is typically ensured by requiring that schedules be serializable.

The local DBMSs are heterogeneous in the sense that they may follow different concurrency control protocols and recovery procedures to ensure correctness in presence of concurrent execution of transactions. The main challenge in building an MDBS environment is to design appropriate software on top of the existing heterogeneous local DBMSs such that both the atomicity and isolation properties of global transactions are ensured. Since the various local DBMSs may belong to autonomous organizations (e.g., competing airlines) and are pre-existing, the design of the MDBS software must satisfy the following two constraints:

- Existing software of the local DBMSs is not modified.
- Each local DBMS retains complete control over transactions that access data items in its database. For example, a local DBMS is free to abort a transaction at any time during its execution.

These two constraints are referred to as the autonomy constraints.

Ensuring the isolation property of global transactions in an MDBS environment has been widely studied in [BS88, DE89, GRS91, BGRS91, MRKS91]. The problem of ensuring the atomicity property has been previously addressed in [BST90, WV90, MR91, MRKS92]. Since each local DBMS ensures the atomicity of subtransactions of a global transaction, the task of ensuring the atomicity of global transactions reduces to ensuring that either all subtransactions of a global transac-
tion commit, or all subtransactions of a global transaction abort. One way to ensure this is to use an atomic commit protocol [BHG87]. A typical atomic commit protocol is the two-phase commit (2PC) protocol [LS76, Gra78]. An atomic commit protocol requires that each local DBMS support a prepared state for transactions. A transaction is in the prepared state if it can neither be unilaterally aborted nor committed by the local DBMS.

Using an atomic commit protocol, however, violates the autonomy constraints of an MDBS since:

- Pre-existing local DBMS may not support a prepared state for transactions thus requiring substantial changes to the existing software.
- Supporting a prepared state for transactions results in a local DBMS losing control over the transactions executing at its own site (since a transaction in a prepared state can neither be aborted nor committed by the local DBMS).

Due to the above reasons, it may not be possible to use a standard atomic commit protocol in an MDBS environment.

If an atomic commit protocol is not used to commit global transactions, then it is possible that certain subtransactions of a global transaction commit whereas others abort. There are two ways of dealing with this problem. The first is to use semantic recovery techniques as was suggested in [LKS91a, MRKS92, MR91]. An example of such a technique is recovery by compensation [GMS87, LKS91b]. However, recovery by compensation can only be used if it is possible to design a compensating transaction to semantically undo the effects of a committed transaction. Moreover, it is the responsibility of the programmer to design and code compensating transactions which is a very complex activity.

The other option is to redo the writes done by the aborted subtransactions as a new transaction. This technique was introduced in [BST90] and subsequently also in [WV90]. Redoing the writes of aborted subtransactions may, however, result in the loss of database consistency. In [BST90], the authors suggested sufficient restrictions on the data items accessed by global transactions and a new transaction management scheme such that database consistency is preserved in an environment where each site follows the strict two-phase locking protocol.

In this paper, we focus our attention on the redo technique by following the work reported in [BST90]. We develop a global commit protocol for MDBS environments that uses the redo technique to ensure global transaction atomicity. We identify minimal restrictions that need to be imposed on the data items accessed by the global subtransactions such that database consistency is not jeopardized. We then show that these restrictions need to be combined with some restrictions on the set of permitted executions of global transactions, and that the schedules produced by the participating local DBMSs are required to satisfy certain properties in order to preserve correctness. More precisely, we define a class of schedules, semi-rigorous schedules, such that if the schedules produced by the local DBMS are semi-rigorous, and the execution of global transactions is restricted to be rigorous [BGRS91], then the use of our commit protocol does not result in a loss of database consistency. Further, neither the requirement that local schedules be semi-rigorous, nor the restriction that the execution of global transactions be rigorous can be relaxed. Finally, since not all existing DBMSs produce semi-rigorous schedules, we propose further restrictions on the global transactions (the proposed restrictions are still weaker than the ones developed in [BST90] and [WV90]) such that the requirement of the local schedules to be semi-rigorous can be relaxed.

2 Preliminaries

An MDBS consists of a set of autonomous pre-existing local DBMSs located at sites s1, s2, ..., sm. The MDBS software that executes on top of the existing local DBMSs consists of a global transaction manager (GTM), and a set of servers, one associated with each local DBMS. Pre-existing local applications make calls directly to local DBMS interfaces. Global transactions are processed by the GTM. The GTM communicates with a local DBMS for the execution of the operations of a global transaction through the server which acts as the liaison between the GTM and the local DBMS. Operations belonging to a global subtransaction are submitted to the local DBMS by the server as a single transaction. We assume that each local DBMS acknowledges the execution of operations submitted to it. An operation belonging to a global transaction, if it conflicts with a previously submitted operation, is submitted to the local DBMS only after the acknowledgement for the execution of the previous operation has been received by the server.

A transaction Ti is a sequence of read (denoted by rj) and write (denoted by wj) operations followed by either a commit (denoted by cj) or an abort (denoted by aj) operation. Concurrent execution of transactions results in a schedule. Formally, a schedule S = (τ, <S) is a finite set τ of transactions with a partial order <S over the set of operations belonging to transactions in τ. The local schedule at site sj, denoted by Sj, is a total order over the operations belonging to local transactions and global subtransactions that access data items at sj. In
this paper, we assume that each local DBMS ensures serializability across the local schedules and the atomicity of the local transactions and global subtransactions executing at its site. The local DBMS may be following any algorithm for ensuring serializability of schedules and atomicity of transactions.

2.1 The Global Commit Protocol
To ensure global transaction atomicity, a global commit protocol is used in which the servers, rather than the local DBMSs, participate in the protocol. This implies that each server maintains a stable log to store the write operations of global subtransactions. The protocol is the usual 2PC protocol in which the GTM acts as a coordinator and the servers at the local DBMS act as the participants. The protocol works as follows.

The GTM, after the last operation of a global transaction has been processed, initiates the protocol by sending a VOTE-REQ message to the servers at the sites at which the transaction executed. Upon receipt of a VOTE-REQ message, a server responds by sending its vote: YES (commit) or a NO (abort). If the server votes to commit the transaction, it forces the appropriate log records corresponding to the writes done by the transaction onto stable storage before responding. If all the participating servers respond with a YES, then the GTM decides to commit the transaction and sends a COMMIT message to each participating server. Else, it sends an ABORT message to the servers. If a server receives a COMMIT message from the GTM, it submits the commit operation for the transaction to the local DBMS; if it receives an ABORT message from the GTM, it submits an abort operation for the transaction to the local DBMS.

Note that the local DBMSs do not participate in the global commit protocol above. We now discuss how recovery from global subtransaction aborts is done. A global subtransaction may be aborted by the local DBMS for a variety of reasons. A subtransaction may be aborted due to logical error, it may be chosen as a victim by the local deadlock detection algorithm, it may be aborted since it is involved in a cycle in the local serialization graph, or it may be aborted by the local DBMS due to a timeout constraint. If a global subtransaction is aborted by the local DBMS after the GTM has decided to commit the global transaction, the server at the site at which the subtransaction aborted uses its private log to construct a redo transaction. The redo transaction, consisting of all the writes performed by the subtransaction in question, is submitted to the local DBMS for execution by the server. In case of failure of the redo transaction, it is repeatedly resubmitted by the server until it commits. Note that since the redo transaction consists of only the write operations, it cannot logically fail.

2.2 A Motivating Example (Example 1)
Consider an MDBS consisting of two sites: site s1 with data items x, y, and site s2 with data item z. Let T1 be the following global transaction.

\[ T_1 : r_1(y) w_1(x) w_1(z) \]

Suppose that the GTM decides to commit T1 (using the global commit protocol). Further suppose that T1 successfully commits at s2, but the local DBMS at s1 aborts T1. In order to ensure the atomicity of T1, the server at s1 constructs the following redo transaction T2 from the log and submits it to the local DBMS.

\[ T_2 : w_2(x) \]

However, before T2 is executed, a local transaction:

\[ T_3 : r_3(x) w_3(y) \]

executes at s1. This results in the following local schedule at s1 (the write operation of T1 that aborted at s1 is included in square brackets for clarity).

\[ S_1 : r_1(y) [w_1(z)] a_1 r_3(x) w_3(y) c_3 w_2(x) c_2 \]

Since the local DBMS at s1 considers T1 and T2 to be separate transactions, schedule S1 is serializable in the local view (that is, in the view of the local DBMS). However, in the global view (that is, in the view of the MDBS), since read done by T1 and the write performed by T2 are part of one transaction, S1 is not serializable.

The example above illustrates that our scheme may result in the loss of database consistency. To preserve consistency, the execution at each local DBMS is required to be serializable in the global view. We refer to a local schedule as Multidatabase serializable (M-serializable) if it is serializable in the global view. The schedule, in the example above, is not M-serializable. We thus need to identify appropriate conditions under which a schedule is M-serializable.

3 Formalization of M-serializability
In order to formalize the notion of M-serializability, we need to develop some notation and present some basic definitions.

A local schedule \( S_j \) is said to be locally complete if for every transaction \( T_i \) in \( S_j \), a commit operation \( c_i \) or an abort operation \( a_i \) appears in \( S_j \). A locally complete schedule \( S_j \) may contain global subtransactions that are committed by the GTM but are aborted by the

\[ ^1 \text{Serializability, in this paper refers to conflict serializability [BHGS87].} \]
local DBMS at site \( sj \). If for each such subtransaction, in case the subtransaction was not a read-only transaction, the schedule \( S_j \) contains a redo transaction that has locally committed, then we refer to \( S_j \) as \textit{globally complete}. Thus, a globally complete schedule is also locally complete.

Let \( S_j \) be a globally complete schedule at site \( sj \). We denote by \( Gc \) the set of global subtransactions in \( S_j \) that are committed by both the GTM and the local DBMS. We denote by \( Ga \) the set of global subtransactions in \( S_j \) that are committed by the GTM but aborted by the local DBMS. Further, we partition the set \( Ga \) into two subsets: \( Gar \) and \( Gaw \). \( Gar \) contains all the transactions \( T_i \in Ga \) that are read-only transactions. \( Gaw \) contains all the transactions \( T_i \in Ga \) that are not read-only transactions. Corresponding to each transaction \( T_i \in Gaw \), there is a redo transaction (denoted by \( W_i \)) in \( S_j \) that is committed by the local DBMS at \( sj \). We denote the set of redo transactions and local transactions that are committed by the local DBMS by \( WC \) and \( Lc \) respectively. Finally, corresponding to each global subtransaction \( T_i \in Gaw \), we define \( Ri \) to be a transaction consisting of all operations of \( T_i \) except the write operations. Note that for each transaction \( T_i \in Gar \), \( Ri \) is the transaction \( T_i \) itself. The set of \( Ri \)'s is denoted by \( Ra \). To illustrate the above notation, in Example 1 the schedule \( S_1 \) is globally complete. Further, \( Ga = \{ T_1 \}, Gaw = \{ T_1 \}, Gar = \{ \}, Gc = \{ \}, WC = \{ T_2 \}, Lc = \{ T_3 \}, \) and \( Ra = \{ R_1 \} \), where \( R_1 : r_1(y) \ a_1 \).

We define two operators on transactions, \textit{oper} and \textit{dec}. A transaction \( T_i = \text{oper}(T_i) \circ \text{dec}(T_i) \) is either a commit, or an abort operation, \( \text{oper}(T_i) \) is the sequence of read and write operations performed by \( T_i \), and \( \circ \) is the concatenation of two sequences of operations.

A projection of a schedule \( S \) on a set of transactions \( \tau' \), denoted by \( S_{\tau'} \), is a schedule obtained from \( S \) by deleting all operations that do not belong to transactions in \( \tau' \). Formally, \( S_{\tau'} = (\tau', \prec_{S_{\tau'}}) \), where \( \prec_{S_{\tau'}} \subseteq \prec_S \) such that for all \( o_i, o_j \) belonging to transactions in \( \tau' \), \( o_i \prec_S o_j \) if and only if \( o_i \prec_{S_{\tau'}} o_j \).

Let \( S = (\tau, \prec_S) \) be a schedule. Transactions \( T_i, T_j \in \tau, i \neq j \), are said to conflict in \( S \), denoted by \( T_i \rightleftharpoons T_j \), if there exist operations \( o_i \) in \( T_i \) and \( o_j \) in \( T_j \) such that both \( o_i \) and \( o_j \) access the same data item, \( o_i \prec_S o_j \), and at least one of \( o_i \) and \( o_j \) is a write operation. In our definition, the conflict relation \( \rightleftharpoons \) is defined over all transactions in \( S \) including those that aborted in \( S \). Further, \( \rightleftharpoons \) denotes the transitive closure of the \( \rightleftharpoons \) relation.

**Definition 1:** Let \( S_j \) be a globally complete schedule at site \( sj \). \( S_j \) is said to be \textit{M-serializable} if and only if for all transactions \( T_i \in S_j \), \( T_i \nleq T_j \), where

\[
\tau_0 = Lc \cup Gc \cup \{ \text{oper}(R_i) \circ W_i \mid T_i \in Gaw \} \cup \{ R_i \mid T_i \in Gar \}
\]

We next define a conflict relation over transactions, referred to as the \textit{M-conflict} relation, that formalizes when transactions conflict in the global view. To do so, we define a relation \textit{pair} such that for each global subtransaction \( T_i \in Gaw \), \( (R_i, W_i) \in \text{pair} \), and \( (W_i, R_i) \in \text{pair} \). In Example 1, since \( W_1 \) is the redo transaction \( T_2 \), \( \text{pair} = \{(R_1, T_2), (T_2, R_1)\} \).

**Definition 2:** Let \( S_j \) be a globally complete schedule at site \( sj \) and \( AS_j \) be the schedule \( S_j^p \), where \( \tau_1 = Lc \cup Gc \cup WC \cup Ra \). Let \( T_i, T_k \) be two distinct transactions in \( AS_j \). Transaction \( T_i \) is said to M-conflict with \( T_k \) in \( AS_j \) (denoted by \( T_i \prec M T_k \)) if either of the following holds:

1. \( T_i \prec T_k \) in \( AS_j \) and \( \prec \text{pair}(T_i, T_k) \).
2. \( T_i \prec T_k \) in \( AS_j \) and \( \prec \text{pair}(T_k, T_i) \).

By \( \prec \) we denote the transitive closure of the \( \prec \) relation. The definition of M-conflict is based on the observation that \( R_i \) and the corresponding \( W_i \) are part of the same transaction in the global view. Note that transactions that do not conflict may M-conflict. It can be shown that acyclicity of the transitive closure of the M-conflict relation is equivalent to M-serializability of \( S_j \). In Example 1 of Section 2.2, \( AS_1 \) corresponding to the local schedule \( S_1 \) is as follows.

\[
AS_1 : r_1(y) \ a_1 \ r_3(z) \ w_3(y) \ c_3 \ w_2(z) \ c_2
\]

Since \( R_1 \prec T_3 \) in \( AS_1 \), \( R_1 \prec M T_3 \). Further, since \( T_3 \prec W_1 \) in \( AS_1 \), \( T_3 \prec M R_1 \). Thus, we can derive \( R_1 \prec R_1 \) which means that we have a cycle in the transitive closure of the M-conflict relation and thus schedule \( S_1 \) in Example 1 is not M-serializable.

**Theorem 1:** Let \( S_j \) be a globally complete schedule at site \( sj \). \( S_j \) is M-serializable if and only if for all \( T_i \in AS_j \), \( T_i \nleq T_j \). \( \Box \)

4 Ensuring M-serializability

In this section, we identify conditions under which a local schedule is M-serializable. Let \( S_j \) be the schedule produced by the local DBMS at site \( sj \). Since \( S_j \) is serializable (an assumption in our model), the loss of
M-serializability can only occur due to the presence of certain M-conflicts as is stated in the following lemma.

**Lemma 1:** Let $S_j$ be a globally complete schedule at site $s_j$. If $S_j$ is serializable but is not M-serializable, then either one of the following conditions must hold:

- There exists a $R_k \in R_k$ such that $R_k \prec R_k$.
- There exists a $W_k \in W_k$ such that $W_k \prec W_k$.

Thus, M-serializability can be ensured by preventing all $R_k \prec R_k$ and all $W_k \prec W_k$ M-conflicts. To do so we need to do each of the following:

- Impose restrictions on the data items accessed by global transactions.
- Require local schedules to satisfy certain properties.
- Impose restrictions on the execution of global transactions.

In the remainder of this section, we derive appropriate restrictions such that M-serializability of a schedule $S_j$ is ensured.

### 4.1 Restrictions on Global Transactions

To identify restrictions on global transactions it suffices to consider the case where the local schedule $S_j$ contains only a single global subtransaction $T_1$. Using Lemma 1, if $S_j$ is not M-serializable, then either $R_1 \prec R_1$, or $W_1 \prec W_1$. As $S_j$ contains only one global subtransaction, it follows from the definition of M-conflict that the only way such M-conflicts can occur is if either one of the following holds in $AS_j$:

- $R_1 \prec R_1$
- $R_1 \prec W_1$
- $W_1 \prec R_1$

where $T_1 \prec T_2$ if $T_1 \prec T_2$ and $T_1 \neq T_2$.

To ensure M-serializability, the above mentioned conflicts have to be prevented. One trivial way of doing this is to require that global subtransactions do not access any data items that the local transactions either read or write. We, however, wish to impose as few restrictions as possible on the global transactions. To do so, we partition the set of data items at site $s_j$ into three disjoint sets as follows:

- **Local Data** ($LD(s_j)$): The set of data items that the local transactions can both read and write.
- **Global Data** ($GD(s_j)$): The set of data items that local transactions can only read.
- **Exclusive Data** ($ED(s_j)$): The set of data items that local transactions can neither read, nor write.

An example of global data items are the set of replicated data items (local transactions do not write on replicated data since that will violate the integrity constraints of the system). Exclusive data items are data items that are meant for only global transactions to access. For example, a list of participants in an MDBS may be part of exclusive data items.

In order to prevent $R_1 \prec W_1$ and $W_1 \prec R_1$ conflicts, restrictions need to be imposed on the set of data items that global subtransactions can read and write. The restrictions imposed should be such that either $R_1$ does not conflict with any local transaction, or $W_1$ does not conflict with any local transaction. We, therefore, impose the following restrictions on global transactions.

*If a global subtransaction $T_i$ reads a local data item at site $s_j$, then it writes only exclusive data items at $s_j$.*

It is easy to see that relaxing the above restriction may result in either $R_1 \prec W_1$ or $W_1 \prec R_1$ conflicts and thus loss of M-serializability. To illustrate, consider again Example 1. In that example, since the local transaction $T_3$ wrote on data item $y$, it must be the case that $y \in LD(s_1)$. Further, since $T_3$ read $x$, it must be the case that data item $x \in GD(s_1) \cup LD(s_1)$. Since the subtransaction of global transaction $T_1$ at site $s_1$, read $y$ and wrote on $x$, this violated the restriction on global transactions above resulting in a loss of M-serializability. In the remainder of this section, transactions are assumed to be restricted as above.

### 4.2 Requirements of Local Schedules

Though the restrictions placed on the set of data items accessed by global transactions precludes the non M-serializable execution of Example 1, it is not sufficient to ensure M-serializability. To illustrate this, we continue with our scenario in which the local schedule $S_j$ contains only a single global subtransaction $T_1$. As mentioned previously, the restriction placed on global subtransactions prevents $R_1 \prec W_1$ and $W_1 \prec R_1$ conflicts, but it may not prevent a $R_1 \prec R_1$ conflict, as the following example illustrates.

**Example 2:** Consider an MDBS consisting of two sites $s_1$ and $s_2$. Let data items $x, y \in LD(s_1)$, and data item $z \in LD(s_2)$. Let $T_1$ be a global transaction, and let $T_2$ be a local transaction that executes at site $s_1$.

$$
T_1 : \ x \ (r_1) \ y \ (w_1) \\
T_2 : \ z \ (w_2) \ y
$$

168
Suppose that the GTM decides to commit $T_1$. Further suppose that $T_1$ successfully commits at $s_2$, but the local DBMS at $s_1$ aborts $T_1$ and schedules the operations of $T_2$. The above scenario results in the following local schedule at site $s_1$.

$$S_1 : w_2(x) \ r_1(x) \ r_1(y) \ a_1 \ w_2(y) \ c_2$$

In $AS_1$, since $R_1 \sim T_2$, $R_1 \not\sim T_2$. Also, since $T_2 \not\sim R_1$, $T_2 \not\sim R_1$. Thus, $R_1 \not\sim R_1$ and hence $S_1$ is not M-serializable. $\square$

To prevent $T_1 \not\sim T_1$ conflicts in $S_j$, the order in which transactions commit in $S_j$ is required to be of a restricted nature. To identify the requirements of $S_j$, we develop the following classification of schedules based on the conflicts between the various transactions in a schedule $S$.

- **ROW**: For all pairs of transactions $T_i, T_k$ in $S$, if $T_i$ reads a data item $x$ that is later written by $T_k$, then $T_k$ does not commit before $T_i$ either commits or aborts.

- **AROW**: For all pairs of transactions $T_i, T_k$ in $S$, if $T_i$ reads a data item $x$ that is later written by $T_k$, then $T_k$ does not commit before $T_i$ either commits or aborts.

- **WOR**: For all pairs of transactions $T_i, T_k$ in $S$, if $T_i$ writes a data item $x$ that is later read by $T_k$, then $T_k$ does not commit before $T_i$ either commits or aborts.

- **AWOR**: For all pairs of transactions $T_i, T_k$ in $S$, if $T_i$ writes a data item $x$ that is later read by $T_k$, then $T_k$ does not read $x$ before $T_i$ either commits or aborts.

- **WOW**: For all pairs of transactions $T_i, T_k$ in $S$, if $T_i$ writes a data item $x$ that is later written by $T_k$, then $T_k$ does not commit before $T_i$ either commits or aborts.

- **AWOW**: For all pairs of transactions $T_i, T_k$ in $S$, if $T_i$ writes a data item $x$ that is later written by $T_k$, then $T_k$ does not write on $x$ before $T_i$ either commits or aborts.

We will show in the sequel that to ensure M-serializability, the local schedules generated by the participating DBMSs are required to be restricted to certain combinations of these classes. Certain combinations of these class of schedules have been previously identified in the literature. For example, a schedule is rigorous [BGRS91] if it is $AROW + AWOR + AWOW$. A schedule is strongly recoverable [BGRS91] if it is $ROW + WOR + WOW$. Similarly, a schedule is strict [BHGS87] if it is $AWOR + AWOW$. A schedule avoids cascading aborts (ACA) [BHGS87] if it is $AWOW$. A schedule is recoverable [BHGS87] if it is $WOR$. We define a schedule to be semi-rigorous if it is $ROW + AWOR + WOW$. Figure 1 shows how the above properties relate to each other.

**Theorem 2**: Let $S_j$ be a globally complete schedule resulting from the execution of local transactions and a single global subtransaction at site $s_j$. If $S_j$ is semi-rigorous, then $S_j$ is M-serializable. $\square$

The requirement of $S_j$ to be semi-rigorous in Theorem 2 cannot be relaxed since if $S_j$ is not semi-rigorous, then it is possible to construct a system of local transactions and a global subtransaction such that $S_j$ is not M-serializable. The schedule in Example 2 illustrates that if $S_j$ is not AWOR (and thus not semi-rigorous), then loss of M-serializability can occur. The following

4 We use the symbol "$+$" to mean "and". For example, $AROW + AWOR$ means $AROW$ and $AWOR$. 

169
example further illustrates that if $S_j$ is not WOW, then M-serializability may not be ensured. An example that demonstrates the necessity of $S_j$ to be ROW can be similarly constructed.

**Example 3:** Consider an MDBS consisting of two sites $s_1$ and $s_2$. Let data items $x, y, z \in ED(s_1)$ and $u \in LD(s_2)$. Let $T_1$ be a global transaction and $T_2, T_3$ be local transactions that execute at $s_1$. Let $T_4$ be a global transaction and $T_5, T_6$ be local transactions that execute at $s_2$.

\[
\begin{align*}
T_1 & : r_1(x) \quad r_1(y) \quad w_1(u) \\
T_2 & : w_2(z) \quad w_2(y) \\
T_3 & : w_3(z) \quad w_3(x)
\end{align*}
\]

Suppose that the GTM decides to commit $T_1$. Further suppose that $T_1$ successfully commits at $s_2$, but the local DBMS at site $s_1$ aborts $T_1$. The above scenario results in the following local schedule at site $s_1$.

\[
S_1 : w_2(z) \quad w_3(z) \quad w_4(z) \quad c_3 \quad r_1(x) \quad r_1(y) \quad a_1 \quad w_2(y) \quad c_2
\]

In $AS_1$, since $R_1 \sim T_2$, $R_1 \sim T_2$. Similarly, we can derive $T_2 \sim T_3$ and $T_3 \sim R_1$. Thus, $R_1 \sim R_1$. Hence, $S_1$ is not M-serializable.

The requirement of $S_j$ to be semi-rigorous in order to ensure M-serializability is necessary since the local DBMSs may be following any arbitrary concurrency control protocol for ensuring serializability of $S_j$. The non M-serializable executions in Example 2 and Example 3 could be generated if, for example, the local DBMS at site $s_1$ was following either a serialization graph testing (SGT) protocol [BH87], or a timestamp ordering (TO) [BH87] based protocol. If, however, the local DBMS was following a two-phase locking (2PL) protocol, then those executions could not be generated. In fact, if the local DBMS follows the 2PL protocol, then the requirements of $S_j$ can be significantly relaxed as is stated in the following theorem.

**Theorem 3:** Let $S_j$ be a globally complete schedule resulting from the execution of local transactions and a single global transaction at site $s_j$. If the local DBMS at $s_j$ follows the 2PL protocol, then $S_j$ is M-serializable. □

4.3 Restrictions On the Execution of Global Transactions

Unfortunately Theorems 2 and 3 do not hold if multiple global subtransactions are present in $S_j$ as the following example illustrates.

**Example 4:** Consider an MDBS consisting of two sites $s_1$ and $s_2$. Let data items $x \in ED(s_1)$, $y \in LD(s_1)$, and $z \in GD(s_1)$, and let data items $u \in LD(s_2)$ and $v \in GD(s_2)$. Let $T_1$ and $T_2$ be global transactions and $T_3$ be a local transaction that executes at site $s_1$.

\[
\begin{align*}
T_1 & : r_1(x) \quad w_1(z) \quad w_1(u) \\
T_2 & : r_2(y) \quad w_2(x) \quad w_2(v) \\
T_3 & : w_3(y) \quad r_3(z)
\end{align*}
\]

Suppose that the GTM decides to commit both $T_1$ and $T_2$. Further suppose that $T_1$ and $T_2$ commit at $s_2$, but the local DBMS at site $s_1$ aborts both $T_1$ and $T_2$ and schedules $T_3$ for execution. Since the GTM considers both $T_1$ and $T_2$ to be committed, the server at site $s_1$ executes the following redo transactions $T_4$ and $T_5$ for $T_1$ and $T_2$ respectively:

\[
\begin{align*}
T_4 & : w_4(z) \\
T_5 & : w_5(z)
\end{align*}
\]

The above scenario results in the following local schedule at site $s_1$.

\[
S_1 : r_1(x) \quad [w_1(z)] \quad a_1 \quad r_2(y) \quad [w_2(x)] \quad a_2 \quad w_3(y) \quad r_3(z) \quad c_3 \quad w_4(z) \quad c_4 \quad w_5(z) \quad c_5
\]

In $S_1$, $W_1$ and $W_2$ are the redo transactions $T_4$ and $T_5$ respectively. In $AS_1$, since $T_3 \sim T_4$ and $pair(T_4, R_1)$, $T_3 \sim R_1$. Since $R_2 \sim T_3$, $R_2 \sim T_3$. Thus, $R_2 \sim R_2$. Further, since $R_1 \sim T_3$ and $pair(T_3, R_2)$, $R_1 \sim R_2$ and hence $R_1 \sim R_1$, hence $S_1$ is not M-serializable.

Since the local schedule $S_1$ is serial, imposing further restrictions (than semi-rigorousness) on $S_1$ will not prevent non M-serializable executions as in Example 4. To prevent such executions we need to restrict the execution of global transactions.

To do so, we first define a projection of $S_j$, referred to as $GS_j$, over the operations belonging to the global subtransactions. $GS_j$ is the projection of $S_j$ over transactions in $\tau_j$; that is, $GS_j = S_j^{\tau_j}$, where

\[\tau_j = Gc \cup \{oper(T_i) \circ dec(W_i) \mid T_i \in Gar\} \cup \{R_i \mid T_i \in Gar\}\]

To ensure M-serializability, the schedule $GS_j$ is required to be rigorous. In Example 4, $GS_1$ corresponding to the local schedule $S_1$ is as follows:

\[
GS_1 : r_1(x) \quad w_1(z) \quad r_2(y) \quad w_2(x) \quad c_4 \quad c_5
\]

since $GS_1$ is not AROW, it is not rigorous and thus $S_1$ is not M-serializable. The following theorem states conditions under which $S_j$ is M-serializable.

170
Theorem 4: Let $S_j$ be a globally complete schedule resulting from the execution of local and global transactions at site $s_j$. If $GS_j$ is rigorous and $S_j$ is semi-rigorous, then $S_j$ is M-serializable. □

The requirement of $GS_j$ to be rigorous in Theorem 4 is necessary in that it is always possible to find a system of local transactions and global subtransactions such that if $GS_j$ is not rigorous, then M-serializability of local schedules is not ensured. Since global transactions execute under the control of the GTM, the GTM can ensure that the schedule $GS_j$ is rigorous by controlling the order of execution of the operations belonging to global transactions.

Similar to the case with a single global subtransaction, in the presence of multiple global subtransactions, the requirement of $S_j$ to be semi-rigorous is also necessary since the local DBMSS may be following any arbitrary concurrency control protocol. If in case the local DBMS follow the 2PL protocol, then as in Theorem 3, the requirement of $S_j$ to be semi-rigorous can be relaxed.

Theorem 5: Let $S_j$ be a globally complete schedule resulting from the execution of local and global transactions at site $s_j$. If $GS_j$ is rigorous and the local DBMS at $s_j$ follows the 2PL protocol, then as in Theorem 3, the requirement of $S_j$ to be semi-rigorous can be relaxed.

5 Relaxing Requirements of Local Schedules

In the previous section, we have shown that if each of the local DBMSS generate semi-rigorous schedules, then M-serializability can be ensured. Unfortunately, not all the concurrency control protocols followed by the pre-existing local DBMSS generate semi-rigorous schedules. Since the autonomy constraint prohibits us from making modifications to the existing local DBMSS software to ensure that they generate only semi-rigorous schedules, we need to impose further restrictions on the data items accessed by global transactions to ensure M-serializability. We, therefore, impose the following restriction.

If a global subtransaction $T_i$ reads a local data item at site $s_j$, then it does not write any data item at $s_j$.

We have thus disallowed a global subtransaction that reads local data item from writing exclusive data items. However, even with the above restriction on the global transactions, the semi-rigorousness requirement cannot be relaxed in a straightforward fashion. Note that in Example 2, even though the global subtransaction of transaction $T_1$ at site $s_1$ was a read-only transaction, loss of M-serializability resulted. The semi-rigorousness requirement of $S_j$ can, however, be relaxed by a slight modification to the global commit protocol and by forcing direct conflicts between subtransactions of global transactions.

5.1 Early Commit Protocol

In Example 2, the loss of M-serializability occurred since in $S_1$ the subtransaction of $T_1$ that read local data items was considered committed by the GTM but aborted by the local DBMS. We imposed restrictions on the order in which transactions commit in $S_j$ to prevent such executions. Another way of preventing the execution in Example 2 is by modifying the GTM commit protocol as follows.

The GTM, in the first phase of the global commit protocol, instead of sending a VOTE-REQ message, sends a COMMIT message to the servers at all the sites at which the subtransaction is a read-only transaction. We refer to such servers as r-servers. To the remaining servers, referred to as w-servers, the GTM sends a VOTE-REQ message. An r-server, on receipt of a COMMIT message from the GTM submits a commit operation for the transaction to the local DBMS. On receipt of an acknowledgement from the local DBMS, the server sends a COMMIT-ACK message to the GTM; if the subtransaction is aborted by the local DBMS the server sends a ABORT-ACK message to the GTM. A w-server behaves as in the previous case and sends its vote: YES (commit) or a NO (abort). If the GTM receives a YES from each w-server and a COMMIT-ACK from each r-server, it decides to commit the transaction and sends a COMMIT message to each of the w-servers. Else, if it receives a NO from some w-server or an ABORT-ACK from some r-server, it decides to abort the transaction and sends an ABORT message to each of the w-servers. A w-server on receipt of a message containing the decision of the GTM submits either a commit or an abort operation to the local DBMS (depending upon the decision).

We refer to the above protocol as the early commit (EC) protocol. If the GTM uses the EC protocol, the execution in Example 2 that was not M-serializable will not be permitted since the GTM will not consider transaction $T_1$ in the example as committed until after the local DBMS at site $s_1$ has committed $T_1$.

Theorem 6: Let $S_j$ be a globally complete schedule resulting from the execution of local transactions and a single global subtransaction at site $s_j$. If the GTM follows the EC protocol, then $S_j$ is M-serializable. □

In the presence of multiple global subtransactions, however, unless additional restrictions are placed both
Example 5: Consider an MDBS consisting of two sites $s_1$ and $s_2$. Let data items $x, y \in GD(s_1)$, $z \in LD(s_1)$ and $u, v \in GD(s_2)$. Let $T_1$ and $T_2$ be global transactions and $T_3$ and $T_4$ be local transactions that execute at site $s_1$.

\[
\begin{align*}
T_1 & : r_1(x) w_1(y) w_1(u) \\
T_2 & : w_2(x) w_2(v) \\
T_3 & : w_3(z) r_3(z) \\
T_4 & : r_4(y) r_4(x)
\end{align*}
\]

Suppose that the GTM decides to commit $T_1$. Further suppose that $T_1$ successfully commits at $s_2$, but the local DBMS at $s_1$ aborts $T_1$ and schedules the operations of $T_3$ and $T_4$. Since the GTM considers $T_1$ to be committed, the server at site $s_1$ executes the following redo transaction $T_5$ for $T_1$:

\[T_5 : w_5(y)\]

Finally, transaction $T_2$ is executed and it commits at both $s_1$ and $s_2$. The above scenario results in the following local schedule at site $s_1$.

\[
\begin{align*}
S_1 : r_1(x) [w_1(y)] a_1 w_3(z) r_4(y) r_4(z) c_4 \\
& w_5(y) c_5 w_2(x) c_2 r_3(x) c_3
\end{align*}
\]

In $S_1$, $W_1$ is the redo transaction $T_5$. In $AS_1$, since $R_1 \sim T_2$, $T_2 \sim T_3$ and $T_3 \sim T_4$, we can derive $R_1 \sim T_4$. Further, since $T_4 \sim T_5$ and $pair(T_5, R_1)$, $T_4 \sim R_1$. Thus, $R_1 \sim R_1$ and hence $S_1$ is not M-serializable.

In Example 5, schedule $GS_1$ is serial and thus no restriction on $GS_1$ will prevent the non M-serializable execution above. To prevent the loss of M-serializability, $S_j$ is required to be strongly recoverable. In Example 5, since $S_1$ is not WOR, it is not strongly recoverable resulting in a loss of M-serializability. Examples in which non M-serializable executions result if $S_j$ is either not ROW or not WOW can be similarly constructed.

Theorem 7: Let $S_j$ be a globally complete schedule resulting from the execution of local and global transactions at site $s_j$. If the GTM follows the EC protocol, $GS_j$ is ROW + AWOR, and $S_j$ is strongly recoverable, then $S_j$ is M-serializable.

5.2 Forcing Conflicts

If the GTM uses the EC protocol, then the requirement that $S_j$ be semi-rigorous can be relaxed to strong recoverability. Thus, the EC protocol by itself may not be useful if the local DBMS does not ensure that schedules satisfy the strong recoverability requirement. To further relax the requirements on $S_j$, the EC protocol needs to be augmented by the following GTM scheme that we refer to as the ticket scheme.

In the ticket scheme, site $s_j$ contains a special data item called $ticket_j$. Data item $ticket_j$ is maintained only for the purpose of the GTM concurrency control and is not accessed by local transactions. The GTM augments each global subtransaction $T_i$ executing at $s_j$ which is not a read-only subtransaction to write onto $ticket_j$ sometime during the execution of $T_i$. Thus, any two global subtransactions that are not read-only transactions directly conflict in the schedule $S_j$.

Theorem 8: Let $S_j$ be a globally complete schedule resulting from the execution of local and global transactions at site $s_j$. If the GTM follows the ticket scheme and uses the EC protocol, and $GS_j$ is strongly recoverable and strict, then $S_j$ is M-serializable.

Note that in Theorem 8 no requirements are placed on $S_j$ for the purpose of ensuring M-serializability and thus autonomy constraints of the MDBS are fully preserved. In the following example we demonstrate that relaxing the requirement of $GS_j$ to be AWOW may result in a loss of M-serializability. Similar examples that illustrate the need of $GS_j$ to be ROW and AWOR can be constructed.

Example 6: Consider an MDBS consisting of two sites $s_1$ and $s_2$. Let data items $x \in GD(s_1)$ and $y \in GD(s_2)$. Let $T_1$ and $T_2$ be global transactions that execute at sites $s_1$ and $s_2$.

\[
\begin{align*}
T_1 & : r_1(x) w_1(ticket_1) w_1(y) \\
T_2 & : w_2(ticket_1) w_2(x) w_2(y)
\end{align*}
\]

Suppose that the GTM decides to commit $T_1$. Further suppose that $T_1$ successfully commits at $s_2$, but the local DBMS at $s_1$ aborts $T_1$ and schedules the operations of $T_2$. Since the GTM considers transaction $T_1$ to be committed, the server at site $s_1$ executes the following redo transaction $T_3$ for $T_1$:

\[T_3 : w_3(ticket_1)\]

The above scenario results in the following local schedule at site $s_1$.

\[
\begin{align*}
S_1 : r_1(x) [w_1(ticket_1)] a_1 w_2(ticket_1) w_3(ticket_1) c_3 w_2(x) c_2
\end{align*}
\]

In $S_1$, $W_1$ is the redo transaction $T_3$. In $AS_1$, since $R_1 \sim T_2$, $R_1 \sim T_3$, $T_2 \sim T_3$, $T_2 \sim R_1$. Further, since $T_2 \sim T_3$, $T_2 \sim R_1$. Thus, $R_1 \sim R_1$ and hence $S_1$ is not M-serializable.
6 Conclusions

We studied the problem of ensuring atomicity of global transactions in an MDBS environment. As shown, the atomicity requirement of global transactions and the autonomy constraints on the design of the MDBS software are mutually conflicting goals. Traditional mechanism of ensuring atomicity of transactions by utilizing an atomic commit protocol is unsuitable for MDBS environments. We, therefore, developed a global commit protocol that ensures atomicity without violating autonomy of the local DBMSs. The use of our protocol, however, may result in local schedules that are not serializable in the view of the MDBS. We identified restrictions that need to be placed on (a) the data items accessed by global transactions, (b) the execution of global transactions and the requirements of local schedules such that local schedules are serializable in the MDBS view if the global commit protocol is used to ensure atomicity of global transactions. The restrictions identified are minimal in the sense that relaxing them may result in the loss of database consistency.

For brevity, we have omitted the proofs of the various theorems. However, to illustrate the nature of the proofs, we present the proof of Theorem 7 in the Appendix. The proofs of the remaining theorems can be found in [MRB*92b].

We further need to examine how our commit protocol can be used in conjunction with the existing schemes for ensuring global serializability (e.g., the commit graph approach [BST90] and various other schemes developed in [MRB*92a]) to provide a unified solution to the transaction management problem in MDBS environments.

References


Appendix

In this appendix, we present the proof of Theorem 7. To do so, we will need the following notation. For each transaction $T_i \in AS_j$ (that is, $T_i \in Gc \cup Wc \cup Lc \cup Ra$) we define $\overline{T_i}$ as follows:

$$\overline{T_i} = \begin{cases} T_k & \text{if } T_i \in Ra \text{ and } pair(T_i, T_k) \\ T_i & \text{otherwise} \end{cases}$$

Note that if $T_i$ is $R_1$ for some global subtransaction $T_i \in Ga$, and if $T_i$ is a read-only transaction, then according to the definition above, since for no $T_k$, $pair(T_i, T_k)$, $T_i$ is the transaction $T_i$ itself.

For each transaction $T_i \in Wc$ by $\overline{T_i}$ we denote the transaction $oper(T_k) \circ dec(T_i)$, where $T_k \in Ga$, and $T_i$ is the redo transaction corresponding to $T_k$ (that is, $T_i$ is $W_k$).

In order to prove Theorem 7, we will need the following lemmas.

**Lemma 2:** Let the GTM follow the EC protocol and global transactions be restricted as in Section 5. If $T_i \sim T_k$ in $AS_j$, then

1. If $T_i \in Ra$, then $T_k \in Gc \cup Wc$.
2. If $T_k \in Ra$, then $T_i \in Gc \cup Wc$.

**Proof:** We only prove (1). The proof of (2) is similar and thus omitted. Let $T_i$ be $R_1$ for some global transaction $T_i \in Ga$. Since the GTM uses an EC protocol, $T_i$ contains a write operation (else, $T_i$ would not be in $Ra$). Due to the restrictions on global transactions, since $T_i$ is not a read-only transaction, $T_i$ could not have read any local data item. Since $T_i \sim T_k$, there exists conflicting operations $o_1, o_2$ such that $o_1 = r(z), o_2 = w(x)$, and $T_i \prec AS, o_2$, where $z$ is not a local data item. Thus, since $T_k$ writes $z$, $T_k$ is not a local transaction. Hence, $T_k \in Gc \cup Wc$.

**Lemma 3:** Let the GTM follow the EC protocol, $GS_j$ be ROW + AWOR, and $S_j$ be strongly recoverable. If $T_1 \prec T_2$ in $AS_j$, then $dec(T_1) \prec AS, dec(T_2)$.

**Proof:** To prove the lemma there are following three cases to consider.

1. $(T_1, T_2 \notin Ra)$: Note that $\overline{T_1} = T_1$, $\overline{T_2} = T_2$, $\overline{T_1} = c_1$ and $\overline{T_2} = c_2$. Since $T_1 \sim T_2$, there exists conflicting operations $o_1, o_2$ such that $o_1 = T_1(z), o_2 = T_2(x)$, and $o_1 \prec AS, o_2$, $o_1 \prec S, o_2$. Depending upon $o_1$ and $o_2$, there are three cases to consider:

   - $(o_1 = r_1(z)$ and $o_2 = w_2(x))$: Since $S_j$ is ROW and $dec(T_2) = c_2, dec(T_1) \prec S, dec(T_2)$. Hence, $dec(T_1) \prec AS, dec(T_2)$.

   - $(o_1 = w_1(z)$ and $o_2 = r_2(x))$: Since $S_j$ is WOR and $dec(T_2) = c_2, dec(T_1) \prec S, dec(T_2)$. Hence, $dec(T_1) \prec AS, dec(T_2)$.

   - $(o_1 = w_1(z)$ and $o_2 = w_2(x))$: Since $S_j$ is WOW and $dec(T_2) = c_2, dec(T_1) \prec S, dec(T_2)$. Hence, $dec(T_1) \prec AS, dec(T_2)$.

   Thus, in case $T_1, T_2 \notin Ra$, $dec(T_1) \prec AS, dec(T_2)$.

2. $(T_1 \notin Ra$ and $T_2 \in Ra)$: Note that $\overline{T_1} = T_1$ and $\overline{T_2} = c_1$. Since $T_1 \sim T_2$, there exists
conflicting operations \( o_1, o_2 \), such that \( o_1 \in T_1 \) and \( o_2 \in T_2 \), \( o_1 \prec_{AS} o_2 \). Let \( \tilde{T}_2 = T_2 \), \( o_1 = w_1(x) \), and \( o_2 = w_2(x) \). By Lemma 2, \( T_1 \in Wc \cup Gc \). There are two cases to consider:

- \((T_1 \in Gc)\): In this case, \( o_1, o_2 \) are operations in \( GS_j \). Since \( o_1 \prec_{AS} o_2 \), \( o_1 \prec_{GS} o_2 \). Since \( GS_j \) is AWOR, \( dec(T_1) \prec_{GS} o_2 \). Further since \( o_2 \prec_{GS} dec(T_3) \), \( dec(T_1) \prec_{GS} dec(T_3) \). Hence, \( dec(T_1) \prec_{AS} dec(T_3) \).

- \((T_1 \in Wc)\): Let \( T_4 = \tilde{T}_1 \). In this case, there exists \( o_3 \in T_4 \) such that \( o_3 = w_4(x) \) and \( o_3 \prec_{S_j} o_1 \). Since \( o_1 \prec_{AS} o_2 \), \( o_1 \prec_{S_j} o_2 \). Thus, \( o_3 \prec_{S_j} o_2 \). Since both \( o_3 \) and \( o_2 \) are operations in \( GS_j \), \( o_3 \prec_{GS} o_2 \). Since \( GS_j \) is AWOR, \( dec(T_4) = dec(T_1) \prec_{GS} o_2 \). Further since \( o_2 \prec_{GS} dec(T_3) \), \( dec(T_1) \prec_{GS} dec(T_3) \). Hence, \( dec(T_1) \prec_{AS} dec(T_3) \).

Thus, in case \( T_1 \notin Ra \) and \( T_2 \in Ra \), \( dec(T_1) \prec_{AS} dec(T_3) \), where \( T_3 = \tilde{T}_2 \). Since \( \tilde{T}_1 = T_1 \), \( dec(\tilde{T}_1) \prec_{AS} dec(\tilde{T}_2) \).

3. \((T_1 \in Ra \) and \( T_2 \notin Ra)\): Note \( \tilde{T}_2 = T_2 \) and \( dec(T_2) = o_2 \). Since \( T_1 \prec T_2 \), there exists conflicting operations \( o_1, o_2 \), such that \( o_1 \in T_1 \) and \( o_2 \in T_2 \), \( o_1 \prec_{AS} o_2 \). Let \( \tilde{T}_1 = T_3 \), \( o_1 = r_1(x) \) and \( o_2 = w_2(x) \). By Lemma 2, \( T_2 \in Gc \cup Wc \). There are two cases to consider:

- \((T_2 \in Gc)\): In this case, \( o_1, o_2 \) are operations in \( GS_j \). Since \( o_1 \prec_{AS} o_2 \), \( o_1 \prec_{GS} o_2 \). Since \( GS_j \) is ROW, \( dec(T_3) \prec_{GS} dec(T_3) \). Hence, \( dec(T_3) \prec_{AS} dec(T_3) \).

- \((T_2 \in Wc)\): Let \( T_4 = \tilde{T}_2 \). In this case, there exists \( o_3 \in T_4 \) such that \( o_3 = w_4(x) \) and \( o_3 \prec_{S_j} o_2 \). We first show that \( o_1 \prec_{S_j} o_3 \). Suppose on the contrary \( o_3 \prec_{S_j} o_1 \). Since both \( o_1, o_3 \) are operations in \( GS_j \), \( o_3 \prec_{GS} o_1 \). Since \( GS_j \) is AWOR, \( dec(T_4) = dec(T_2) \prec_{GS} o_1 \). Hence, \( dec(T_2) \prec_{S_j} o_1 \). Since \( o_2 \prec_{S_j} dec(T_2) \), this would imply that \( o_2 \prec_{S_j} o_1 \). Thus, \( o_2 \prec_{AS} o_1 \) which is contrary to our assumption that \( o_1 \prec_{AS} o_2 \). Hence, it must be the case that \( o_1 \prec_{S_j} o_3 \).

Since both \( o_1 \) and \( o_3 \) are operations in \( GS_j \) and \( o_1 \prec_{S_j} o_3 \), \( o_1 \prec_{GS} o_3 \). Since \( GS_j \) is ROW, \( dec(T_3) \prec_{GS} dec(T_4) = dec(T_2) \). Hence, \( dec(T_3) \prec_{AS} dec(T_2) \).

Thus, in case \( T_1 \in Ra \) and \( T_2 \notin Ra \), \( dec(T_3) \prec_{AS} dec(T_3) \), where \( T_3 = \tilde{T}_1 \). Since \( \tilde{T}_2 = T_2 \), \( dec(\tilde{T}_1) \prec_{AS} dec(\tilde{T}_2) \).

**Lemma 4:** Let the GTM follow the EC protocol, \( GS_j \) be ROW + AWOR, and \( S_j \) be strongly recoverable. If \( T_1 \prec T_2 \) in \( AS_j \), then \( dec(\tilde{T}_1) \prec_{AS} dec(\tilde{T}_2) \).

**Proof:** The proof is by induction on the length \( l \) of the derivation of \( T_1 \prec T_2 \).

**Basis (I=1):** Thus, \( T_1 \prec T_2 \). By the definition of \( \prec \), there are the following two cases to consider.

1. \((T_1 \prec T_2)\): Trivial by Lemma 3.

2. \((T_1 \prec T_3 \) and pair\((T_3, T_2)\)): By Lemma 3, \( dec(\tilde{T}_1) \prec_{AS} dec(\tilde{T}_3) \). Since pair\((T_2, T_3), \tilde{T}_3 \prec \tilde{T}_2 \). Hence, \( dec(\tilde{T}_2) = dec(\tilde{T}_3) \). Thus, \( dec(\tilde{T}_1) \prec_{AS} dec(\tilde{T}_2) \).

**Induction:** Let the lemma hold for all \( l \leq L \). We show that it holds for \( l = L + 1 \). Let \( T_1 \prec T_2 \) be any arbitrary M-conflict of length \( L + 1 \). Thus, there exists M-conflicts \( T_1 \prec T_3 \) of length \( L_1 \) and \( T_3 \prec T_2 \) of length \( L_2 \), for some \( L_1, L_2 \leq L \). By IH, \( dec(\tilde{T}_1) \prec_{AS} dec(\tilde{T}_3), \) \( dec(\tilde{T}_2) \). Thus, \( dec(\tilde{T}_1) \prec_{AS} dec(\tilde{T}_2) \). Hence proved. \( \square \)

**Proof of Theorem 7:** If \( S_j \) is not M-serializable, then by Theorem 1, there exists a \( T_1 \) such that \( T_1 \prec T_1 \). In that case, by Lemma 4, \( dec(\tilde{T}_1) \prec_{AS} dec(\tilde{T}_1) \) which is an absurdity. Thus, \( S_j \) is M-serializable. \( \square \)