Overcoming Heterogeneity and Autonomy in Multidatabase Systems

Sharad Mehrotra
Rajeev Rastogi
Yuri Breitbart
Henry F. Korth
Abraham Silberschatz

1 Dept. of Information and Computer Science
424 Computer Science
University of California, Irvine
Irvine, CA 92697
sharad@ics.uci.edu

2 AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

A multidatabase system (MDBS) is a software system for integration of pre-existing and independent local database management systems (DBMSs). The transaction management problem in MDBSs consists of designing appropriate software, on top of local DBMSs, such that users can execute transactions that span multiple local DBMSs without jeopardizing database consistency. The difficulty in transaction management in MDBSs arises due to the heterogeneity of the transaction management algorithms used by the local DBMSs, and the desire to preserve their local autonomy. In this paper, we develop a framework for designing fault-tolerant transaction management algorithms for MDBS environments that effectively overcomes the heterogeneity and autonomy induced problems. The developed framework builds on our previous work. It uses the approach described in [Mehrotra et al. 1992a] to overcome the problems in ensuring serializability that arise due to heterogeneity of the local concurrency control protocols. Furthermore, it uses a redo approach to recovery for ensuring transaction atomicity [Breitbart et al. 1990; Mehrotra et al. 1992b; Wolski and Veijalainen 1990] which strives to ensure atomicity of transactions without the usage of the 2PC protocol. We reduce the task of ensuring serializability in MDBSs in the presence of failures to solving three independent subproblems, solutions to which together constitute a complete strategy for failure-resilient transaction management in MDBS environments. We develop mechanisms using which each of the three subproblems can be solved without requiring any changes be made to the pre-existing software of the local DBMSs and without compromising their autonomy.

1 Much of this work was done at the University of Texas at Austin, with support from Texas Advanced Research Program under Grant ARP-209, the National Science Foundation under Grants IRI-9221301 and IRI-9312063, and grants from the IBM and Hewlett-Packard corporations.
1. INTRODUCTION

Recent progress in networking and database technologies have made it possible to integrate a number of pre-existing database management systems that may belong to different autonomous business organizations and may be dispersed over a number of geographically distributed sites. The resulting integrated system, called a multidatabase system, provides the users facilities to access and update data located at other remote databases without requiring them to know either the location or the characteristics of different databases and their corresponding database management systems. The problem of transaction management in multidatabase consists of developing a software module, on top of local database management systems, that allows users to execute transactions that span multiple databases without jeopardizing the consistency of the data. The difficulty arises due to the following two characteristics of the MDBS environments:

—**Heterogeneity:** Each local DBMS may follow different concurrency control and recovery algorithms.

—**Autonomy:** The participation of a local DBMS in the MDBS must not result in a loss of control by the local DBMS over its data and its local transactions.

In a conventional distributed database system, serializability is ensured using the *distributed two-phase locking* (2PL) protocol [Bernstein et al. 1987] and atomicity of transactions is ensured using the *two-phase commit* (2PC) protocol [Bernstein et al. 1987]. In an MDBS environment, if each local system follows the 2PL protocol, is capable of participating in a 2PC protocol, and conforms to the X/Open DTP standard [Gray and Reuter 1993], then, from the perspective of transaction management, the local DBMSs could be integrated using existing TP systems [Gray and Reuter 1993]. There are three major reasons why this approach is unacceptable. These reasons collectively motivate research on transaction management in MDBSs.

First, the local DBMSs may be pre-existing legacy systems that may have been developed independently without any regard that these systems will be integrated into an MDBS on a later date. Legacy DBMSs may not adhere to standards and may not even support an interface for the execution of the 2PC protocol. Historically, many commercial transaction processing monitors have been developed as *closed* systems which cannot participate in a 2PC protocol. Examples of such closed systems include several popular commercial systems like IBM’s IMS and Tandem’s TMF. Requiring that either modifications be made to the existing software to support the standard protocols, or that the data from these pre-existing systems be migrated to a new system that is capable of interoperation may not be a feasible cost-effective solution to integration.

Second, it is possible that the local DBMSs are highly specialized data management systems (as contrasted to general-purpose systems) which have been developed for a specific application domain and they use special-purpose concurrency control and recovery algorithms. For example, a local DBMS may be a full-text database system used within an organization for storage and retrieval of office documents. Such a system may use a special-purpose transaction processing scheme to preserve consistency of the document index. It may not be possible
to integrate such specialized home-brewed local DBMSs into an MDBS using existing transaction processing monitors.

Another compelling reason why existing transaction processing software does not suffice for the task of MDBS integration is that the usage of standard transaction management protocols (viz., the 2PC protocol) may be unacceptable in environments where preservation of local autonomy is of utmost importance [Wolski and Veijalainen 1990; Breitbart et al. 1992]. This is due to the fact that the 2PC protocol requires the local DBMSs to support a prepared state for transactions in which the local DBMSs can neither unilaterally abort nor commit the transaction\(^2\). A transaction while it is in a prepared state at a local DBMS, holds onto the resources (e.g., locks on data it accessed) thereby blocking local applications from accessing the data for possibly unbounded durations. Thus, participating in the 2PC protocol results in the local DBMS losing control over its own data and its applications which is a violation of the autonomy of the local DBMSs. As a result, usage of the 2PC protocol is unacceptable in environments in which preservation of autonomy is important. For example, in an MDBS that integrates autonomous divisions of a multi-national organization, it is unlikely that one division will permit other divisions to hold onto its resources for unbounded periods of time thus blocking local applications since that could result in a significant performance degradation in processing its local applications.

Due to the above reasons, we assume that MDBS considers local DBMSs as “black boxes” that cannot be modified for integration. Local DBMSs may follow different transaction management protocols and may not communicate any control information relevant for transaction management (e.g., information about conflicts between transactions, local logs etc.) to each other, or to the MDBS software. Furthermore, each local DBMS reserves the right to process transactions that access its data based entirely on its own consideration and in particular it may abort the transaction at any time before the transaction is committed at the local DBMS. In such an environment, both the tasks of ensuring serializability as well as that of ensuring atomicity of transactions become very challenging. Schemes for transaction management in distributed databases have traditionally been studied under the assumption that local DBMSs follow the same concurrency control protocol, provide an interface for participation in the 2PC protocol, and share information regarding transaction management with each other. These schemes have not addressed the problems that arise due to the heterogeneity of the transaction management algorithms followed by the different local DBMSs and have not considered the autonomous nature of the local DBMSs.

In this paper, we develop a framework for designing fault-tolerant transaction management algorithms for autonomous MDBS environments. The developed framework builds on our previous work. It uses the approach described in [Mehrotra et al. 1992a] to overcome the problems in ensuring serializability that arise due to heterogeneity of the concurrency control protocols followed by the local DBMSs. Furthermore, it uses a redo approach to recovery for ensuring transaction atomicity [Mehrotra et al. 1992b; Wolski and Veijalainen 1990].

\(^2\)Requirement of the prepared state for transactions is not only the property of 2PC but inherent to any atomic commit protocol. This is a direct consequence of the fact that there does not exist any non-blocking atomic commit protocol [Skeen 1982]. If there existed a commit protocol in which the local DBMSs could unilaterally commit or abort a transaction at any time during its execution, then that protocol can be easily modified into a non-blocking commit protocol, the impossibility of which has been established in [Skeen 1982].
which strives to ensure atomicity of transactions without the usage of the 2PC protocol. We show that if the redo approach to recovery is used, the task of ensuring serializability in the presence of failures reduces to solving three independent subproblems. Solutions to these together constitute a complete strategy for failure-resilient transaction management in MDBS environments where preservation of local autonomy is important. We develop mechanisms using which each of the three subproblems can be solved.

The remainder of the paper is organized as follows. In Section 2, we develop a model for MDBS transaction management software. In Section 3, we identify the difficulties that arise in transaction management due to the heterogeneity of the local DBMSs and the desire to preserve their autonomy. Section 4 develops our basic framework for designing transaction management schemes for MDBSs. In that section, we identify the three subproblems that must be solved to achieve a solution for failure-resilient transaction management in MDBSs. Strategies for addressing these subproblems are developed in Sections 5, 6, and 7. Section 8 is on related and previous work and Section 9 concludes the paper. In the paper, a number of theorems are listed whose proofs are of considerable length and are included separately in the appendix.

2. MULTIDATABASE MODEL

An MDBS consists of a set of autonomous pre-existing centralized local DBMSs that are located at sites \( s_1, s_2, \ldots, s_m, m \geq 2 \). The set of data items at site \( s_i \) is denoted by \( DB_i \). The set of all the data items \( \bigcup_{i=1}^{m} DB_i \) is denoted by \( DB \). Each local DBMS provides access to data through transactions.

A transaction \( T_i \) is a partially ordered set of database operations. More formally, a transaction \( T_i = (O_{T_i}, \prec_{T_i}) \), where \( O_{T_i} \) is finite set of operations and \( \prec_{T_i} \) is a partial order over operations in \( O_{T_i} \). For simplicity, we assume that the only database operations that can be invoked by a transaction \( T_i \) are read (denoted by \( r_i \)), write (denoted by \( w_i \)), begin (denoted by \( b_i \)), commit (denoted by \( c_i \)) and abort (denoted by \( a_i \)). A transaction \( T_i \) that executes at multiple local DBMSs has multiple begin and commit operations, one for each local DBMS at which the transaction executes\(^3\). For a transaction \( T_i \) that executes at multiple sites, the subtransaction of \( T_i \) at site \( s_j \) is denoted by \( T_{ij} \). Formally, \( T_{ij} \) is a restriction of \( T_i \) over the set of operations of \( T_i \) that execute at \( s_j \).\(^4\)

In keeping with the autonomy requirement which dictates that the applications local to a DBMS execute completely under its control, transactions are classified into the following two types:

— **Global transactions**: those that execute at several sites under the control of the MDBS software.

— **Local transactions**: those that execute at a single site outside the control of the MDBS software. Examples of these include pre-existing applications.

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. The GTM may either be distributed over

\(^3\)In contrast, the read and write operations of \( T_i \) on each data item are unique. Since, in this paper, we do not consider the problem of replica control, we consider different copies of the same data item located at different sites as independent data items.

\(^4\)A set \( P_1 \) with a partial order \( \prec_{P_1} \) on its elements is a restriction of a set \( P_2 \) with a partial order \( \prec_{P_2} \) on its elements if \( P_1 \subseteq P_2 \), and for all \( e_1, e_2 \in P_1 \), \( e_1 \prec_{P_1} e_2 \) if and only if \( e_1 \prec_{P_2} e_2 \).
various sites, or centrally located at one site. To execute global transactions the GTM communicates with a local DBMS through the server which acts as the liaison between the GTM and the local DBMSs.

Each local DBMS provides an interface which is used by the server to submit operations of the global subtransactions for execution. Different local DBMSs may support different interfaces [Breitbart et al. 1992]. At one extreme, a local DBMS may provide an interface that permits the server to submit individual database operations (e.g., read and write operations) belonging to global subtransactions for execution, and which acknowledges the execution of the submitted operation. We refer to such an interface as the operation interface. At the other extreme, a local DBMS may support a service interface [Breitbart et al. 1992] in which it only permits servers to submit a request for execution of an existing local application on behalf of the global transaction. In the service interface, the servers submit the entire global subtransaction (and not the individual read and write operations that constitute the application) for execution. Another possible interface may permit the server to request multiple SQL statements (or statements expressed in the local data manipulation language) as part of the global subtransaction, the execution of each being acknowledged by the local DBMS. The submitted SQL statement (or the service request in the case of the service interface) may result in multiple read and write operations over the data and the index structures (e.g., B-trees) maintained by the local DBMS. The MDBS software may be unaware of these resulting operations, as well as of the mechanisms used by the local DBMS for processing the SQL queries (e.g., protocol for B-tree traversal [Gray and Reuter 1993], key range locking for phantom protection [Gray and Reuter 1993]). The nature of the interface supported by the local DBMSs impacts the MDBS transaction management. We will discuss its impact on our solution after developing our approach. We assume that the interface supported by the local DBMSs acknowledges the execution of operations submitted to them.

The concurrent execution of transactions results in a schedule. A schedule \( S = (\tau_S, \prec_S) \), where \( \tau_S \) is a finite set of transactions and \( \prec_S \) is a partial order over the operations belonging to transactions in \( \tau_S \).\(^5\) The partial order \( \prec_S \) satisfies the property that it preserves the order of steps within each transaction (that is, \( \prec_T \subseteq \prec_S \), for each \( T \in \tau_S \)). Let \( S = (\tau_S, \prec_S) \) be a schedule, and let \( \tau \subseteq \tau_S \) be a set of transactions. We denote the projection of \( S \) onto transactions in \( \tau \) by \( S^\tau \), where \( S^\tau = (\tau, \prec_{S^\tau}) \) is a restriction of \( S \).

A conflict relation is associated with the operations in a schedule. Typically, two operations belonging to different transactions in a schedule are said to conflict if they do not commute (e.g., a read and a write operation on the same data item belonging to different transactions). Conflicts between operations induce a conflict relation over transactions. A transaction \( T_i \) is said to conflict with \( T_j \), denoted by \( T_i \sim T_j \), if there exist operations \( o_i \) in \( T_i \) and \( o_j \) in \( T_j \), \( T_i \neq T_j \), such that \( o_i \prec_S o_j \), and \( o_i \) conflicts with \( o_j \).\(^6\) By \( \sim^* \) we denote the transitive closure of the \( \sim \) relation.\(^7\)

A global schedule \( S \) is a schedule resulting from the concurrent execution of all the transactions (both local

---

\(^5\)In the distributed environment of an MDBS, the partial order \( \prec_S \) is the “happened-before” relationship (e.g., see [Lamport 1978]) among the operations.

\(^6\)Note that in our definitions “conflict” is a symmetric notion for operations, but not for transactions.

\(^7\)Note that a transaction \( T_i \) is said to be serialized before a different transaction \( T_j \) if \( T_i \sim T_j \) holds.
and global) at all the local DBMSs. We denote the projection of $S$ over the set of subtransactions accessing the data items in $DB_j$ by $S_j$. We refer to $S_j$ as the local schedule at site $s_j$. Thus, a local schedule consists of operations belonging to local and global transactions that access data items at a single local DBMS. We assume that each local DBMS ensures serializability of the local schedule at its site. Further, we also assume that each local DBMS has pre-existing recovery procedures that ensure the atomicity of the local transactions and the global subtransactions that execute at its site in the presence of failures.

3. TRANSACTION MANAGEMENT ISSUES IN MDBS

The MDBS software, together with the local DBMSs must ensure that the atomicity, consistency, isolation and durability (ACID) properties of transactions [Bernstein et al. 1987] are preserved despite failures. Even though each local DBMS ensures serializability of the local schedule at its site, the resulting global schedules may not be serializable. Thus, an MDBS must provide a mechanism for ensuring global serializability. An MDBS must also provide mechanisms for recovery from failures such that atomicity of the global transactions is ensured.

Problems in Ensuring Serializability: The problem of ensuring global serializability consists of ensuring serializability of the global schedule under the assumption that local schedules at each local DBMS are serializable. In [Breitbart and Silberschatz 1988], it is claimed that global serializability is ensured if there exists a total order defined over global transactions that is consistent with the serialization order of global transactions at each of the local DBMSs. Indeed, it can be shown that the above condition is both necessary and sufficient for ensuring global serializability as is claimed in the following theorem.

**Theorem 1:** Consider an MDBS where each local schedule is serializable. Global serializability is ensured if and only if there exists a total order, $<_G$, such that at each site $s_k$, for all pairs of global transactions $T_i, T_j$ executing at site $s_k$, if $T_i <_G T_j$, then $T_j \triangleright_i T_i$ in $S_k$.

**Proof:** See Appendix A. □

Note that since local transactions execute outside the control of the MDBS software, to ensure the total order mentioned above, the GTM may only control the execution order of the global transactions’ operations. This turns out to be difficult since local transactions may cause indirect conflicts between global transactions that may not conflict directly. In fact, as is illustrated in the following example, even if the GTM permits no concurrency among global transactions and restricts their execution to be serial, loss of global serializability may occur.

**Example 1:** Consider an MDBS environment consisting of two sites: $s_1$ with data items $a$ and $b$, and $s_2$ with data items $c$ and $d$. Consider the following global transactions $T_1$ and $T_2$ and local transactions $T_3$ and $T_4$.

$T_1 : r_1(a), r_1(c) \quad T_2 : r_2(b), r_2(d) \quad T_3 : w_3(a), w_3(b) \quad T_4 : w_4(c), w_4(d)$

Let $T_1$ execute first at sites $s_1$ and $s_2$ followed by $T_2$ at both $s_1$ and $s_2$. It is possible for the local transactions $T_3$ and $T_4$ to execute in such a manner that $T_1$ is serialized before $T_2$ in the local schedule $S_1$ at site $s_1$, and $T_2$ is serialized before $T_1$ in the local schedule $S_2$ at site $s_2$, thus resulting in a non-serializable global schedule illustrated below.

$S_1 : r_1(a), w_3(a), w_3(b), r_2(b) \quad S_2 : w_4(c), r_1(c), r_2(d), w_4(d) \quad □$
Problems in Ensuring Atomicity: The atomicity requirement of a global transaction dictates that either all its subtransactions at the sites at which it executes commit, or all its subtransactions abort. Since the local DBMSs may not participate in the 2PC protocol, and may unilaterally abort a subtransaction of the global transaction at any time during its execution, ensuring atomicity becomes a difficult task. Specifically, the problem arises when a local DBMS aborts a subtransaction (due to a timeout, a failure, etc.) after some other local DBMS has committed a subtransaction of the global transaction. Such a global transaction is committed at some local DBMSs and aborted at other local DBMSs resulting in a loss of atomicity.

Problems in Detecting Deadlocks: Other important issues in developing transaction management techniques for MDBSs is that of detecting and resolving deadlocks. Since more than one local DBMS may be following a lock-based concurrency control in which transactions are made to wait for each other, it is possible that a multi-site deadlock results. One way to detect such deadlocks in distributed DBMSs is for each local DBMSs to maintain and exchange their wait-for graphs [Bernstein et al. 1987]. However, such deadlock detection algorithms are rarely implemented in practice primarily due to their complexity and the infrequent occurrences of multi-site deadlocks [Gray and Reuter 1993]. Instead, systems typically rely on timeout as a mechanism to detect deadlocks [Gray and Reuter 1993] even though timeouts may result in the detection of false deadlocks. The timeout approach is even more attractive for MDBSs since unlike traditional distributed DBMSs, local DBMSs may not possess the capability of exchange information about their wait-for graphs. Theoretically, this problem can be overcome and techniques for detecting deadlocks are discussed in [Breitbart and Silberschatz 1988]. These techniques are quite pessimistic and assume that if two transactions are concurrently active at the same local DBMS, then one of them is waiting for the other. As a result, a large number of false deadlocks are detected. Even timeout based mechanisms may not be simple to implement in MDBS environments. Difficulty may arise in determining appropriate timeouts since in an MDBS environment different local DBMSs may have different processing speeds, and different load characteristics which may not be known to the MDBS software in advance. In this paper, we do not address deadlock detection in MDBS environments.

4. A TRANSACTION MANAGEMENT FRAMEWORK FOR MDBS

In this section, we develop a framework for transaction management for MDBS environments that effectively overcomes the heterogeneity and autonomy induced problems. For reasons of simplicity, we first develop our ideas under the assumption of no failures. We relax that assumption in Sections 4.2 and 4.3. Also for simplicity, we initially assume all DBMSs offer an operation interface. This assumption will be relaxed in later sections.

4.1 Concurrency Control

As stated in Theorem ??, a necessary and sufficient condition for ensuring serializability in an MDBS is that there exists a total order defined over global transactions that is consistent with the serialization order of global transactions at each of the local DBMSs. In order to ensure such a total order over global transactions, the GTM must obtain information about the relative serialization order of the global transactions at the local DBMSs. One mechanism that the GTM may adopt to determine such information is to exploit the knowledge (if available)
of the type of concurrency control protocols used by the local DBMSs. The serialization functions mechanisms discussed below, which is similar to the notion of \textit{o-element} developed in [Pu 1988] and that of the \textit{serialization event} introduced in [Elmagarmid and Du 1990], enables the GTM to exploit such knowledge to determine the relative serialization order among global transactions.

A serialization function is a mapping from a set of transactions in a schedule to one of the operations of a transaction. Let $S = (\tau_S, \prec_S)$ be a serializable schedule, and let $\tau'$ be a subset of transactions in $\tau_S$. A serialization function of a transaction $T_i \in \tau'$ in a schedule $S$ with respect to the set of transactions $\tau'$, denoted by $ser_{S,\tau'}(T_i)$, is a function that maps $T_i \in \tau'$ to some operation in $T_i$ such that the following holds:

For all $T_i, T_j \in \tau'$, if $T_i$ is serialized before $T_j$ in $S$, then $ser_{S,\tau'}(T_i) \prec_S ser_{S,\tau'}(T_j)$

Henceforth, we will denote the function $ser_{S,\tau'}$ by $ser_{\tau'}$. The set of transactions $\tau'$ will be clear from the context.

For numerous concurrency control protocols that generate serializable schedules, it is possible to associate a serialization function with transactions $T_i \in \tau_S$ in the schedule $S$ such that the above property is satisfied. For example, if the \textit{timestamp ordering} (TO) concurrency control protocol is used to ensure serializability of $S$ and the scheduler assigns timestamps to transactions when they begin execution, then the function that maps every transaction $T_i \in \tau_S$ to $T_i$'s begin operation is a serialization function for transaction $T_i$ in $S$ with respect to the set of transactions $\tau_S$.

For a schedule $S$, there may be multiple serialization functions. For example, if $S$ is generated by a \textit{two-phase locking} (2PL) protocol, then a possible serialization function for transactions in $S$ maps every transaction $T_i \in \tau_S$ to the operation that results in $T_i$ obtaining its last lock. Alternatively, the function that maps every transaction $T_i \in \tau_S$ to the operation that results in $T_i$ releasing its first lock is also a serialization function for $T_i$ in $S$.

It is possible that for transactions in a schedule generated by certain concurrency control protocols, no serialization function may exist. For example, in a schedule generated by \textit{serialization-graph testing} (SGT) scheduler, it may not be possible to associate a serialization function with transactions. However, in such schedules, serialization functions can be introduced by forcing direct conflicts between transactions [Georgakopoulos et al. 1991]. Let $\tau' \subseteq \tau$ be a set of transactions in a schedule $S$. If each transaction in $\tau'$ executed a conflicting operation (say a write operation on data item \textit{ticket}) in $S$, then the function that maps a transaction $T_i \in \tau'$ to its write operation on \textit{ticket} is the serialization function for the transactions in $S$ with respect to the set of transactions $\tau'$.

The notion of a serialization function simplifies the GTM's task of controlling the actual serialization order of global transactions at the local DBMSs — only the execution order of the $ser_{S_k}$ operations need be controlled. To describe how such control can be achieved, we define for each global transaction $T_i$, a projection $\hat{T}_i$, consisting of only those operations that are $ser_{S_k}$ operations for some site $k$. Formally, $\hat{T}_i$ is defined as follows.

**Definition 1**: Let $T_i$ be a global transaction. $\hat{T}_i$ is a restriction of $T_i$ consisting of all the operations in the set $\{ser_{S_k}(T_i) \mid T_i \text{ executes at site } s_k \}$. □

Further, for the global schedule $S$, we define a schedule $\hat{S}$ to be the restriction of $S$ consisting of the set of operations belonging to transactions $\hat{T}_i$. Thus, $\hat{S} = (\tau_S, \prec_S)$, where $\tau_S = \{\hat{T}_i \mid T_i \text{ is a global transaction} \}$, and
for all operations \( a, o \) in \( \hat{S} \), \( a \prec \hat{S} o \), iff \( a \prec S o \).

**Example 2:** Consider an MDBS environment consisting of two sites: \( s_1 \) with data items \( a \) and \( b \), and \( s_2 \) with data item \( c \). Suppose that the local DBMS at site \( s_1 \) follows the TO scheme in which a timestamp is assigned to a transaction when it begins execution, and the local DBMS at site \( s_2 \) follows the strict 2PL protocol [Gray and Reuter 1993]. Consider the following global transactions \( T_1 \) and \( T_2 \) and a local transaction \( T_3 \).

\[
T_1 : b_{11}, w_1(a), b_{12}, w_1(c), e_{11}, e_{12} \quad T_2 : b_{21}, r_2(b), b_{22}, r_2(c), e_{21}, e_{22} \quad T_3 : b_3, r_3(a), w_3(b), c_3
\]

Let \( ser_{S_1} \) be the function that maps every transaction in \( \tau_1 \) to its begin operation. Also, let \( ser_{S_2} \) be the function that maps every transaction in \( \tau_2 \) to its commit operation. Thus, \( ser_{S_1}(T_1) = b_{11}, ser_{S_1}(T_2) = b_{21}, ser_{S_2}(T_1) = c_{12} \) and \( ser_{S_2}(T_2) = c_{22} \). As a result, transactions \( \hat{T}_1, \hat{T}_2 \) are as follows.

\[
\hat{T}_1 : b_{11}, e_{12} \quad \hat{T}_2 : b_{21}, e_{22}
\]

Consider the global schedule \( S \) resulting from the concurrent execution of transaction \( T_1, T_2 \) and \( T_3 \) such that the local schedules at sites \( s_1 \) and \( s_2 \) are as follows.

\[
S_1 : b_{11}, b_{21}, w_1(a), r_3(a), w_3(b), c_3, r_2(b), e_{11}, e_{21} \quad S_2 : b_{22}, r_1(b), w_1(c), r_2(c), e_{22}
\]

Schedule \( \hat{S} \) (which is a total order in this case) is as follows:

\[
\hat{S} : b_{11}, b_{21}, c_{12}, c_{22}
\]

In the schedule \( \hat{S} \), the notion of conflict between operations is defined differently as compared to that based on commutativity in standard concurrency control theory.

**Definition 2:** Let \( S \) be a global schedule. Operations \( ser_{S_k}(T_i) \) and \( ser_{S_l}(T_j) \) in schedule \( \hat{S}, T_i \neq T_j \), are said to conflict if \( k = l \).

Therefore, in Example 2, operations \( b_{11} \) and \( b_{21} \) conflict in \( \hat{S} \), whereas operations \( b_{11} \) and \( c_{22} \) do not conflict in \( \hat{S} \). Note that operations \( b_{11} \) and \( b_{21} \) do not conflict in \( S \).

It is not too difficult to show that the serializability of the schedule \( S \) (based on the standard notion of conflicts between operations) can be ensured by ensuring the serializability of the schedule \( \hat{S} \) (based on the notion of conflicts between operations in \( \hat{S} \) as discussed in Definition 2). Essentially, ensuring serializability of \( \hat{S} \) based on the notion of conflicts in Definition 2 enforces a total order over global transactions, such that if \( T_i \) occurs before \( T_j \) in the total order, then \( ser_{S_k} \) operation of \( T_i \) occurs before \( ser_{S_k} \) operation of \( T_j \) for all sites \( s_k \) at which they execute in common, thereby ensuring serializability of \( S \).

**Theorem 2:** Consider an MDBS where each local schedule is serializable. Global serializability is ensured if \( \hat{S} \) is serializable.

**Proof:** See Appendix B.
How this can be achieved— that is, how the MDBS software can ensure serializability of $S$ will be discussed in Section 5. We now turn our attention to developing a mechanism for ensuring atomicity of the transactions in MDBS environments.

4.2 Redo Approach to Ensuring Atomicity

As discussed earlier, in the absence of an atomic commit protocol it is possible that certain subtransactions of a global transaction commit, whereas others abort, thereby violating the atomicity property. One approach to ensure atomicity of such a global transaction, is to reexecute the writes done by the aborted subtransactions as a separate transaction. Commitment of the redo transaction consisting of updates performed by the aborted subtransactions will ensure the atomicity of the original global transaction. This is, in essence, the redo approach to ensuring atomicity. Note that in the redo approach the read operations of the aborted subtransaction and the writes done by the redo transactions constitute a single transaction. However, since the local DBMS considers the two transactions as distinct, loss of consistency may occur. Below we describe the redo approach in greater detail, and develop notions of $M$-recoverability and $M$-serializability which characterize when consistency is preserved if the redo approach is used to preserve transaction atomicity.

In the redo approach, to ensure global transaction atomicity, a global commit protocol is used in which the servers, rather than the local DBMSs, participate. Each server maintains a server log in which it stores the update operations of the subtransactions of the global transactions that execute at its site. On completion of the operations of a global transaction $T_i$, the GTM initiates a global commit protocol by sending a $\langle vote_{req}, T_i \rangle$ message to the servers at the sites at which $T_i$ executed. Upon receipt of a $\langle vote_{req}, T_i \rangle$ message, a server responds by sending the GTM a message containing its vote to commit or to abort $T_i$. The GTM collects the votes from all the servers. If all the votes are to commit $T_i$, then the GTM decides to commit $T_i$ and sends a $\langle commit, T_i \rangle$ message to all the participating servers. Otherwise, it decides to abort $T_i$ and sends an $\langle abort, T_i \rangle$ message to all the servers. On receiving a $\langle commit, T_i \rangle$ message from the GTM, the server submits the commit operation for the subtransaction of $T_i$ to the local DBMS; else, on receiving an $\langle abort, T_i \rangle$ message, it submits an abort operation for the subtransaction of $T_i$ to the local DBMS.

Note that the above protocol for committing global transactions is the standard 2PC protocol [Bernstein et al. 1987] in which the servers (and not the local DBMSs) participate. Since a local DBMS does not participate in the protocol, it can abort the subtransaction of a global transaction at any time during the transaction’s execution—hence, the autonomy of the local DBMS is not violated. If the local DBMS at site $s_j$ aborts the subtransaction of a global transaction $T_i$ before the server at $s_j$ votes to commit $T_i$, then the server at $s_j$ votes to abort $T_i$, resulting in the abortion of transaction $T_i$. On the other hand, if the local DBMS at $s_j$ aborts the transaction $T_i$ after the server at $s_j$ votes to commit $T_i$, then it is possible that subtransactions of $T_i$ at other local DBMSs commit (since the GTM could have decided to commit $T_i$), but its subtransaction at $s_j$ aborts. In this case, the server at site $s_j$ uses its log to construct a redo transaction. The redo transaction, consisting of all the writes

---

8Recall that we have currently made an assumption that the local DBMS supports an operation interface. Hence, construction of such a redo transaction is an easy task. However, as mentioned earlier, not every local DBMS may support an operation interface. We will discuss how the redo approach can be modified to work if a local DBMS does not support an operation interface in Section 6.
performed by the subtransaction of $T_i$ at site $s_j$, 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.

Notice that the redo approach assumes that the redo transaction, if submitted sufficiently often, will eventually commit. Unfortunately, it is possible that the updates being performed by the redo transaction results in a violation of the integrity constraints at the local DBMS and, as a result, the local DBMS will repeatedly abort the redo transaction. One way to prevent such an occurrence is to modify the global commit protocol described above as follows. When a server receives the `vote_req` message for a transaction from the GTM, it checks to see if the updates made by the transaction at its site results in a violation of any local integrity constraint\(^9\) before replying to the GTM with its vote. If the integrity constraints are being violated, then the server returns a vote to abort the transaction. This modification of the basic redo approach will prevent the situation in which the redo transaction of a global transaction is repeatedly aborted by the local DBMS due to the violation of the local integrity constraints. The above modification, however, requires that the local integrity constraints be known to the server. If this assumption does not hold, then the only recourse is for the server to raise an exception condition in case the redo transaction repeatedly fails and the loss of atomicity to be handled using some external means (possibly, by executing compensating transactions to undo the effects of the transaction at the sites at which it committed [Gray and Reuter 1993]).

4.2.1 $M$-recoverability. If the redo approach is used to ensure the atomicity of the global transactions, then it must be the case that before the server sends to the GTM its vote to commit transaction $T_i$, each transaction from which $T_i$ read some data item should have previously committed. If this were not the case, then it is possible that the GTM decides to commit a global transaction that reads data items written by an aborted transaction resulting in the loss of atomicity of transactions.

**Example 3:** Consider an MDBS consisting of two sites: site $s_1$ with data item $x$, and site $s_2$ with data item $y$. Let $T_1$ be a global transaction and $T_2$ be a local transaction:

\[
T_1 : r_1(x), w_1(y) \quad T_2 : w_2(x).
\]

Suppose that GTM decides to commit $T_1$ (using the global commit protocol). Further suppose that $T_1$ successfully commits at $s_2$, but the local DBMS at $s_1$ aborts both the subtransaction of the global transaction $T_1$ and the local transaction $T_2$. This may result in the following local schedules at sites $s_1$ and $s_2$.

\[
S_1 : w_2(x), r_1(x), a_1, a_2 \quad S_2 : w_1(y), c_1
\]

In the above schedules, $T_1$ reads the value of data item $x$ from the local transaction $T_2$ which aborts. Since it is possible that the value that $T_1$ writes for data item $y$ depends upon the value of $x$ it read, and since $T_1$ commits at site $s_2$, effects of the aborted transaction $T_2$ persist, thereby violating the atomicity property. □

Note that in the above example, even though the local schedules at both sites $s_1$ and $s_2$ are recoverable [Bernstein et al. 1987], the effects of aborted transaction $T_2$ persists. To prevent such executions, we require that the local schedules at each site be *multidatabase-recoverable* ($M$-recoverable) which is defined below\(^{10}\). The

---

\(^9\)This may be achieved by augmenting the subtransaction with the code to check for the violation of the integrity constraints.

\(^{10}\)In contrast, in a homogeneous distributed database system the success of the 2PC protocol requires each local schedule to be only recoverable [Bernstein et al. 1987]. Every $M$-recoverable schedule is also recoverable.
definition of the M-recoverability uses the \textit{reads\_from} relation [Bernstein et al. 1987] over transactions that is as follows: A transaction $T_i$ is said to read from $T_j$, denoted by \textit{reads\_from}(T_i, T_j) in a schedule $S$, if there exist operations $o_i$ in $T_i$ and $o_j$ in $T_j$, $o_i = r_i(x)$, and $o_j = w_j(x)$ for some data item $x$, $o_j \prec_S o_i$, such that the following holds:

- For all operations $o_k$ such that $o_j \prec_S o_k \prec_S o_i$, $o_k \neq o_j$.
- If there exists an operation $o_k = w_k(x)$ for some transaction $T_k$ such that $o_j \prec_S o_k \prec_S o_i$, then there exists an operation $o_l = a_k$ such that $o_j \prec_S o_l \prec_S o_i$.

\textbf{Definition 3:} A local schedule $S_k$ is said to be M-recoverable, if, for all transactions $T_i, T_j$ in the global schedule $S$, $i \neq j$, if \textit{reads\_from}(T_i, T_j), then transaction $T_j$ commits in $S$ and, further, if $T_i$ commits in $S_k$ then, $c_j \prec_S c_i$. \hfill $\Box$

M-recoverability prevents globally committed transactions from seeing the effects of aborted local and global transactions. In the remainder of the paper, we assume that the schedules produced by the local DBMSs are M-recoverable. This is not an unreasonable assumption since most DBMSs ensure cascadeless executions (that is, the generated schedules avoid cascading aborts (ACA) [Bernstein et al. 1987]). It can be easily shown that an ACA schedule is also M-recoverable.

Unfortunately, even though local schedules are M-recoverable, and the redo approach preserves atomicity of global transactions, it may result in a loss of database consistency. This is illustrated in the following example.

\textbf{Example 4:} Consider an MDBS consisting of two sites: site $s_1$ with data items $x, y$, and site $s_2$ with data item $z$. Let $T_1: r_1(y), w_1(x), w_1(z)$ be a global transaction. Suppose that the GTM decides to commit $T_1$ (using the global commit protocol). Further suppose that $T_1$ successfully commits at $s_2$, but the local DBMS at $s_1$ aborts $T_1$. In order to ensure the atomicity of $T_1$, the server at $s_1$ constructs a redo transaction $T_2$ from the log and submits it to the local DBMS, where $T_2 : w_2(x)$. However, before $T_2$ is executed, a local transaction $T_3$, where $T_3 : r_3(x), w_3(y)$, executes at $s_1$. This results in the following local schedule at $s_1$ (the write operation of $T_1$ that aborted at $s_1$ is included in square brackets for clarity).

$S_1 : r_1(y), [w_1(x)], a_1, r_3(x), w_3(y), c_3, w_2(x), c_2.$

Since the local DBMS at $s_1$ considers $T_1$ and $T_2$ to be separate transactions, schedule $S_1$ 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 the read done by $T_1$ and the write performed by $T_2$ are part of one transaction, $S_1$ is not serializable. \hfill $\Box$

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 \textit{multidatabase serializable} (M-serializable) if it is serializable in the global view. Below we formalize the notion of M-serializability.

4.2.2 \textit{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 \textit{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 local DBMS at $s_j$. 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 \emph{globally complete}. Thus, a globally complete 
schedule is also locally complete.

Let \( S_j \) be a globally complete schedule at site \( s_j \). We denote by \( Gcc \) the set of global subtransaction in \( S_j \) that are committed by both the GTM and the local DBMS. The notation \( Gcc \) can be memorized by understanding the intuition behind it which is as follows - \( G \) stands for the global transaction, the first \( c \) means that the transaction in this set have been committed by the GTM, and the second \( c \) means that the transaction has also been committed by the local DBMS. Similar notation will be used to denote other transaction sets as well. For example, \( Gca \) denotes 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 \( Gca \) into two subsets: \( Gear \) and \( Gcaw \). \( Gear \) contains all the transactions \( T_i \in Gca \) that are read-only transactions. \( Gcaw \) contains all the transactions \( T_i \in Gca \) that are not read-only transactions. Corresponding to each transaction \( T_i \in Gca \), there is a redo transaction (denoted by \( W_i \)) in \( S_j \) that is committed by the local DBMS at \( s_j \). We denote the set of redo transactions and local transactions that are committed by the local DBMS by \( Wc \) and \( Lc \) respectively\(^{11}\). Finally, corresponding to each global subtransaction \( T_i \in Gca \), we define \( R_i \) to be a transaction consisting of all operations of \( T_i \) except the write operations. Note that for each transaction \( T_i \in Gear \), \( R_i \) is the transaction \( T_i \) itself. The set of all \( R_i \) is denoted by \( Rea \). To illustrate the above notation, in Example 4 the schedule \( S_1 \) is globally complete. Further, \( Gca = \{ T_1 \}, Gcaw = \{ T_1 \}, Gear = \{ \}, Gcc = \{ \}, Wc = \{ T_2 \}, Lc = \{ T_3 \}, \) and \( Rea = \{ R_1 \}, \) where \( R_1 : r_1(y), a_1 \).

We define two operators on transactions, \emph{operations} and \emph{decision} where \( \text{operations}(T_i) \) correspond to the set of read and write operations performed by a transaction \( T_i \) and the \emph{decision}(\( T_i \)) is either a commit or abort decision corresponding to \( T_i \). Thus, a transaction \( T_i = \text{operations}(T_i) \circ \text{decision}(T_i) \), where “\( \circ \)” is the concatenation of two sequences of operations.

**Definition 4:** Let \( S_j \) be a globally complete schedule at site \( s_j \) and \( GV(S_j) \) be the projection of \( S_j \) onto transactions in \( \tau_1 \) (that is, \( S_j^\tau_1 \)), where

\[
\tau_1 = Lc \cup Gcc \cup \{ \text{operations}(R_i) \circ W_i \mid T_i \in Gcaw \} \cup Gear
\]

Notice that \( GV(S_j) \) represents the execution in schedule \( S_j \) from the \emph{global view} in which operations belonging to the pair \( R_i \) and \( W_i \) are part of the same transaction. \( S_j \) is said to be M-serializable if and only if for all transactions \( T_i \in GV(S_j) \), \( T_i \not\prec T_i \)\(^{12}\). \( \square \)

We next define a conflict relation over transactions, referred to as the \emph{M-conflict} relation, that identifies when transactions conflict in the global view. The notion of M-Conflict will be useful in identifying conditions under which schedules are M-serializable. To formalize the notion of M-conflicts, we define a relation \emph{pair} such that for each global subtransaction \( T_i \in Gcaw \), \((R_i, W_i) \in \text{pair}\), and \((W_i, R_i) \in \text{pair}\). In Example 4, since \( W_1 \) is the redo transaction \( T_2 \), \emph{pair} = \{ \((R_1, T_2), (T_2, R_1)\) \}. Notice that for a global transaction \( T_i \) that has committed at the GTM but aborted at the local DBMS, \emph{pair} relates the set of reads done by \( T_i \) before it aborted with the

\(^{11}\) Similar to the other notation used in the paper, this notation can be memorized by noting that \( W \) in the notation above stands for redo transactions, \( L \) for local transactions, and \( c \) for commit.

\(^{12}\) \( T_i \not\prec T_i \) means not \( T_i \not\rightarrow T_i \).
updates done by the redo transaction corresponding to \( T_i \). That is, the set of operations corresponding to the pair are considered as part of the same transaction in the global view. We next define the notion of M-conflict which identifies when transactions conflict in the global view (that is, when \( R_i \) and \( W_i \) are considered as part of the same transaction).

**Definition 5:** Let \( S_j \) be a globally complete schedule at site \( s_j \) and \( LV(S_j) \) be the schedule \( S_j^{\tau_i} \), where \( \tau_i = Le \cup Ge \cup Wc \cup Rca \). Note that \( LV(S_j) \) represents the schedule \( S_j \) from the local DBMS’s perspective or the local view in which the redo transactions corresponding to aborted global transactions are considered as separate transactions. Let \( T_1, T_2 \) be two distinct transactions in \( LV(S_j) \). Transaction \( T_1 \) is said to M-conflict with \( T_2 \) in \( LV(S_j) \) (denoted by \( T_1 \overset{M}{\sim} T_2 \)) if either of the following holds:

1. \( T_1 \sim T_2 \) and \( \neg \text{pair}(T_1, T_2) \).
2. \( T_1 \sim T_3 \) and \( \text{pair}(T_3, T_2) \).
3. \( \text{pair}(T_1, T_3) \) and \( T_3 \sim T_2 \).
4. \( \text{pair}(T_1, T_3) \) and \( T_3 \sim T_4 \) and \( \text{pair}(T_4, T_2) \). \( \square \)

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. Below we state a theorem that acyclicity of the transitive closure of the M-conflict relation (denoted by \( \overset{M}{\sim} \)) implies the M-serializability of \( S_j \). To see this, consider the execution in Example 4. In Example 4, \( LV(S_1) \) corresponding to the local schedule \( S_1 \) is as follows.

\[
LV(S_1) : r_1(y), a_1, r_2(x), w_2(y), c_3, w_2(x), c_2
\]

Further, for the execution in Example 4, \( \text{pair} = \{(R_1, T_2), (T_2, R_1)\} \). Note that \( W_1 \) above is transaction \( T_2 \) in Example 4. Since \( R_1 \sim T_3 \) in \( LV(S_1) \), \( R_1 \overset{M}{\sim} T_3 \). Further, since \( T_3 \sim W_1 \) in \( LV(S_1) \), \( T_3 \overset{M}{\sim} R_1 \). Thus, we can derive \( R_1 \overset{M}{\sim} R_1 \) which means that we have a cycle in the transitive closure of the M-conflict relation \( \overset{M}{\sim} \). Hence schedule \( S_1 \) in Example 4 is not M-serializable.

**Theorem 3:** Let \( S_j \) be a globally complete schedule at site \( s_j \). \( S_j \) is M-serializable if for all \( T_i \) in \( LV(S_j) \), \( T_i \not\overset{M}{\sim} T_i \). \( \square \)

**Proof:** See Appendix C. \( \square \)

In fact, we can establish a stronger result that \( \overset{M}{\sim} \) is acyclic if and only if \( S_j \) is M-serializable. Since we will only use the if part of the result in proving schedules are M-serializable under appropriate restrictions on transactions and schedules, we restrict ourselves to proving Theorem 3.

### 4.3 Failure-Resilient Transaction Management in MDBSs

We next study how the redo approach to ensuring atomicity of the global transactions can be combined with the concurrency control mechanism developed in Section 4.1 to achieve a fault-tolerant mechanism for ensuring global

\[ T_i \not\overset{M}{\sim} T_i \] means not \( T_i \overset{M}{\sim} T_i \).
serializability. One would expect that if we were to augment our mechanism for ensuring global serializability in the absence of failures with techniques for ensuring M-serializability of the local schedules, then it would suffice to achieve a solution for ensuring global serializability in the presence of failures. Unfortunately, this may not be the case. Let us illustrate the problems that arise through the following example.

**Example 5:** Consider an MDBS located at two sites: $s_1$ with global data items $x$ and $y$, and $s_2$ with global data items $u$ and $v$. Let $T_1$ and $T_2$ be global transactions and $T_3$ and $T_4$ be local transactions:

$T_1 : b_{11}, w_1(x), b_{12}, w_1(u)$  
$T_2 : b_{21}, w_2(y), b_{22}, w_2(v)$  
$T_3 : b_{32}, r_3(x), r_3(y)$  
$T_4 : b_4, r_4(u), r_4(v)$

Let the local DBMSs at both sites $s_1$ and $s_2$ follow the TO protocol for concurrency control. Thus, the serialization functions $ser_{S_1}$ and $ser_{S_2}$ are the functions that map transactions to their begin operations. As a result the transactions $\hat{T}_1$ and $\hat{T}_2$ are as follows:

$\hat{T}_1 : b_{11}, b_{12}$  
$\hat{T}_2 : b_{21}, b_{22}$

Suppose that the GTM decides to commit both $T_1$ and $T_2$, but the local DBMS at $s_1$ aborts $T_1$. Thus, a redo transaction $T_3$, where $T_3 : b_3, w_3(x)$, is executed to redo the updates of $T_1$. Assume that the above execution results in the following local schedules $S_1$ and $S_2$ at sites $s_1$ and $s_2$ respectively.

$S_1 : b_{11}, [w_1(x)], a_{11}, b_{21}, w_2(y), c_{21}, b_3, r_3(x), r_3(y), c_3, b_5, w_5(x), c_5$

$S_2 : b_{12}, w_1(u), c_{12}, b_4, r_4(v), c_4, b_{22}, w_2(v), c_{22}$

In the above execution, the schedule $\hat{S}$ is as follows: $\hat{S} : b_{11}, b_{21}, b_{12}, b_{22}$. Thus, in the above execution, each local schedule is M-serializable and the schedule $\hat{S}$ is serializable, but the global schedule is not serializable. □

In the execution illustrated in the above example, the operations $ser_{S_1}(T_1)$ (that is, $b_{11}$) occurs before $ser_{S_1}(T_2)$ (that is, $b_{21}$). Thus, a scheme based on ensuring serializability of $\hat{S}$ assumes that the subtransaction of $T_1$ is serialized before the subtransaction of $T_2$ at site $s_1$. However, in the view of the MDBS, due to the presence of the failure of $T_1$ at site $s_1$, the subtransaction of $T_1$ at site $s_1$, (which consists of the reads done by $R_1$ and the writes performed by $W_1$), is serialized after the subtransaction of $T_2$ at site $s_1$, thereby resulting in a violation of global serializability. Below we discuss how the above problem can be dealt with.

Let $T_{ij}$ be a subtransaction of a global transaction $T_i$ that executes at sites $s_j$. Let $Gcaw_j$, $Gca_j$, and $Gcc_j$ refer to the sets $Gca$, $Gear$, and $Gcc$ for site $s_j$ respectively. We denote by $\bar{T}_{ij}$:

\[
\bar{T}_{ij} = \begin{cases} 
\text{operations}(R_{ij}) \circ W_{ij} & \text{if } T_{ij} \in Gcaw_j \\
R_{ij} & \text{if } T_{ij} \in Gca_j \\
T_{ij} & \text{if } T_{ij} \in Gcc_j
\end{cases}
\]

For a global transaction $T_i$, we denote by $\bar{T}_i$ the transaction such that for all sites $s_j$, the projection of $\bar{T}_i$ onto data items at site $s_j$ is the transaction $T_{ij}$. We can now define when global serializability is ensured in the presence of failures:

**Definition 6:** Let $S$ be a global schedule such that for each site $s_j$, $S_j$ is globally complete. Let $\bar{S}$ be the schedule $S^\tau$, where

$\tau_S = \{ \bar{T}_i \mid T_i \text{ is a global transaction committed by the GTM} \} \cup \{ T_i \mid T_i \in Lc_j, \text{for some site } s_j \}$

Schedule $S$ is globally serializable in the presence of failures if $\bar{S}$ is serializable. □
Recall that in a failure free environment, as is shown in [Breitbart and Silberschatz 1988], the necessary and sufficient condition for ensuring global serializability is that there exists a total order defined over global transactions that is consistent with the serialization order of global transactions at each of the local DBMSs. In the presence of failures, if the redo approach to ensuring atomicity is used, that result can be rephrased as follows.

**Theorem 4:** Global serializability is ensured if the following two conditions hold:

1. For each site $s_j$, schedule $S_j$ is M-serializable.
2. For all pairs of global transactions $T_i, T_k$, there exists a total order $\prec_G$ such that for all sites $s_j$, if $T_i \prec_G T_j$, then $T_i \prec_G T_j$.

**Proof:** See Appendix D. □

Note that schemes that ensure serializability of $\hat{S}$ only ensure a total order $\prec_G$ such that for all sites $s_j$, if $T_i \prec_G T_k$, then $T_i \prec_G T_j$ and may not ensure (2) resulting in the loss of global serializability. One way to ensure that (2) holds is to augment our scheme to further ensure that the local schedule $S_k$, at each site $s_k$ preserves the following property which we refer to as order preservation property:

For each pair of global transactions $T_i, T_j$ that execute at $s_k$, if $T_{ik}$ is serialized before $T_{jk}$, then $\text{ser}_{s_k}(T_{ik}) \prec_{s_k} \text{ser}_{s_k}(T_{jk})$.

**Theorem 5:** Global serializability is ensured if each of the following holds:

1. Schedule $\hat{S}$ is serializable.
2. For each site $s_k$, $S_k$ is M-serializable.
3. For all sites $s_k$, $S_k$ satisfies the order preservation property.

**Proof:** See Appendix E. □

The above theorem establishes our framework for ensuring global serializability in MDBSs. In the following three sections we develop mechanisms to ensure serializability of the schedule $\hat{S}$, M-serializability of local schedules, and the order preservation property. Solutions to these problems together constitute a mechanism for failure-resilient transaction management in MDBSs.

5. **ENSURING SERIALIZABILITY OF $\hat{S}$**

To ensure serializability of $\hat{S}$, the MDBS software must control the order in which the operations belonging to $\hat{S}$ execute at the local DBMSs. It can control the execution order of operations in $\hat{S}$ by controlling the order in which it submits the global transactions’ operations for execution to the local DBMSs. Below we describe the mechanism that the GTM can use to ensure serializability of $\hat{S}$ in detail. The described mechanism initially assumes that the local DBMSs support an operation interface. We relax the assumption and discuss how our mechanism can be modified in case the operation interface is not supported later.
Servers and Local DBMSs

Figure 1. GTM components.

To control the execution of the operations in $S$, the GTM consists of two components — GTM$_1$ and GTM$_2$. For every global transaction $T_i$, using the knowledge of the concurrency control protocol followed by the local DBMS, GTM$_1$ determines the operations $ser_{S_k}(T_i)$, and submits them to GTM$_2$ for processing. The remaining global transaction operations (that are not $ser_{S_k}(T_i)$) are directly submitted to the local DBMS through the servers. If, for example, site $s_k$ follows the TO protocol and the timestamps are assigned to transactions when they begin execution, $ser_{S_k}(T_i)$, for a global transaction $T_i$ is $b_{ik}$ operation (that is, $T_i$'s begin operation at site $s_k$). Thus, in this case, GTM$_1$ forwards the begin operation of transactions at site $s_k$ to GTM$_2$. All other operations are submitted directly to the servers for execution.

GTM$_2$, in turn, submits the operations forwarded to it by GTM$_1$ for execution at the local DBMSs. Note that the operation $ser_{S_k}(T_i)$ executes at the local DBMS at $s_k$ only after GTM$_2$ submits it for execution to the requisite server. Furthermore, after the completion of the execution of $ser_{S_k}(T_i)$, GTM$_2$ receives an acknowledgement for it from the server at the local DBMS. Thus, GTM$_2$ may control the order in which the $ser_{S_k}(T_i)$ operations execute in $S$ (and thus in $\hat{S}$) by controlling the submissions of the $ser_{S_k}(T_i)$ operations to the local DBMSs. In particular, it ensures serializability of $\hat{S}$ (which consists of the $ser_{S_k}$ operations) by using a concurrency control protocol (e.g., TO, 2PL, SGT) to control the order in which it submits the operations to the local DBMS for execution. Specific protocols suited for ensuring serializability of $\hat{S}$ can be found in [Mehrotra et al. 1992a; Mehrotra 1993].

To see how the approach works consider the execution in Example 2. Let us assume that GTM$_2$ follows the 2PL protocol to control the submission order of the $ser_{S_k}$ operations to the local DBMS for execution. Assume
that $T_1$ requests execution of operation $b_{11}$ first. Recall that the begin operations of transactions are the $ser_{S_k}$ operations at site $s_1$. Thus, GTM$_1$ forwards the operation $b_{11}$ to GTM$_2$. Since no other transaction holds a conflicting lock, GTM$_2$ submits the operation $b_{11}$ for execution to the local DBMS (via the server at $s_1$). Let us assume that $T_2$ next requests execution of operation $b_{21}$. Since operation $b_{21}$ is the $ser_{S_k}(T_2)$ operation, GTM$_1$ forwards the operation to GTM$_2$. The submission of the operation will be delayed since $\hat{T}_1$ holds a conflicting lock. Once $\hat{T}_1$ releases the lock (according to the 2PL protocol), GTM$_2$ may submit $b_{21}$ for execution.

The above description of the concurrency control mechanism to ensure serializability of the schedule $\hat{S}$ implicitly makes the following two assumptions:

1. The $ser_{S_k}(T_i)$ operations can be associated with each global transaction $T_i$ for all local DBMSs.

2. The operation interface supported by the local DBMSs for the global transactions is such that the MDBS software submits each database operation, including the $ser_{S_k}(T_i)$ operations, explicitly for execution to the local DBMSs, and the local DBMSs acknowledge the execution of the submitted operation.

The basis of the first assumption has been discussed earlier. Depending upon the concurrency control protocol followed by the local DBMS, it may or may not be possible to associate a serialization function with the transactions. If the concurrency control scheme followed by the local DBMS is such that serialization function cannot be associated with transactions, serialization functions can be artificially introduced for global transactions by forcing every two global transactions that execute at some common sites to conflict directly at those sites. This can be accomplished by augmenting global transactions to execute a write operation on a common data item ticket at the site. It should always be possible to add a data item to the local DBMS, but in the case that neither the concurrency control protocol used by the local DBMS supports a serialization function, and nor does the local DBMS provide a mechanism for defining a new data item, the scheme developed in this paper, as well as other approaches to concurrency control in MDBSs developed previously [Georgakopoulos et al. 1991; Breitbart et al. 1992; Elmagarmid and Du 1990; Pu 1988] will not be usable to ensure global serializability. Such a situation is extremely unlikely to occur in practice and, thus, the first assumption is reasonable from the practical standpoint.

Unfortunately, the second assumption may not be valid since some local DBMSs may only support a SQL interface or a service interface. If the local DBMSs do not support an operation interface, the MDBS software does not have direct control over when $ser_{S_k}(T_i)$ operations execute at the local DBMSs. However, the relative order in which $ser_{S_k}$ operations execute can still be controlled by controlling the submission of operations that cause the execution of the $ser_{S_k}(T_i)$ operation at the local DBMS. To see this, consider a local DBMS at site $s_k$ that supports an SQL interface. Furthermore, assume that the local DBMS at $s_k$ follows a TO protocol that assigns timestamps to transactions when they begin execution. That is, $ser_{S_k}(T_i)$ is the first database operation belonging to $T_i$ at site $s_k$. The GTM can control the relative order in which $ser_{S_k}(T_i)$ operations execute at $s_k$ by controlling the order in which it submits the first SQL query for each global transaction $T_i$ to the local DBMS at site $s_k$ (via the servers). This is possible since $ser_{S_k}(T_i)$ for a global transaction $T_i$ executes only after the GTM submits the first SQL query of $T_i$ for execution to, and before receiving an acknowledgment from, the
local DBMS at $s_k$ (via the servers). Thus, if local DBMSs do not support an operation interface, our scheme can still be used to ensure serializability for $\hat{S}$ with the following modification: GTM$_1$ forwards the operation that will cause the execution of $ser_{S_k}(T_i)$ at the local DBMS to GTM$_2$ for processing. GTM$_2$, in turn forwards the operation for execution to the local DBMS (via the servers). As before, GTM$_2$ uses a concurrency control protocol (e.g., TO, 2PL, SGT) to control the order in which it submits the operations to the local DBMS for execution, thereby ensuring serializability of $\hat{S}$.

Notice that the nature of the interface supported by the local DBMS affects the degree of concurrency afforded by the developed approach. For example, in the case of a service interface, the entire service or the subtransaction is considered as a single operation by GTM$_1$, and it forwards the request for service invocation to GTM$_2$ for execution. Since GTM$_2$ uses a concurrency control protocol (e.g., 2PL) to control the order in which it forwards the service request to the local DBMS for execution, the service request at the local DBMS causes the execution of the $ser_{S_k}$ operation for the transaction, and the $ser_{S_k}$ operations of two different transactions at the same site conflict, only a single service request is allowed to execute at the same DBMS at a given time. Thus, the scheme essentially results in global transactions executing sequentially at each local DBMS. In contrast, in the case of the operation interface, multiple global transactions may execute concurrently at a given time at each local DBMS as long as the concurrently executing operations are not the $ser_{S_k}(T_i)$ operations.

6. ENSURING M-SERIALIZABILITY OF LOCAL SCHEDULES

Recall that a complete strategy for failure-resilient transaction management requires solutions to 1) serializability of $\hat{S}$, 2) M-serializability of the local schedules, and 3) order preservation. In the previous section, we discussed how serializability of $\hat{S}$ can be ensured. In this section, we identify conditions under which local schedules are M-serializable and develop mechanisms for ensuring M-serializability. Order preservation is discussed in the following section. To ensure M-serializability, we need to: 1) impose restrictions on the data items accessed by global transactions. 2) require local schedules to satisfy certain properties. 3) 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.

6.1 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. It is not too difficult to see that in the presence of only a single global transaction $T_1$, loss of M-serializability can result only if either of the following holds: 1) $R_1 \not\rightarrow R_1$, 2) $R_1 \not\rightarrow W_1$, or 3) $W_1 \not\rightarrow R_1$, where $T_1 \not\rightarrow T_2$ if $T_1 \not\rightarrow T_2$ and $T_1 \not\rightarrow T_2$. Notice that loss of M-serializability due to $W_1 \not\rightarrow W_1$ is not possible since that would imply a $W_1 \not\rightarrow W_1$ conflict in $S_j$ which is a contradiction since the local DBMS ensures that $S_j$ is serializable.

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. However, imposing restrictions on the global transactions restricts the generality of the solution. Thus,
we 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.

Examples of global data items are the set of data items over which inter-site integrity constraints exist (e.g., replicated data items). Local transactions do not write on such data items 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, data not accessed by any legacy applications running on the local systems will be part of the exclusive data items.

In order to prevent \( R_1 \prec_1 W_1 \) and \( W_1 \prec_1 R_1 \) 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 \( s_j \).*

Notice that the above does not restrict local transactions at the local DBMS. Nor does it restrict global read-only transactions, or global transactions that do not read local data items. Only global transactions that read local data items are restricted to write exclusive data items. It is easy to see that relaxing the above restriction may result in either \( R_1 \prec_1 W_1 \) or \( W_1 \prec_1 R_1 \) conflicts and thus loss of M-serializability. To illustrate, consider again Example 4. 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, unless otherwise stated, transactions are assumed to be restricted as above.

### 6.2 Requirements of Local Schedules

Though the restrictions placed on the set of data items accessed by global transactions prevent \( R_1 \prec_1 W_1 \) and \( W_1 \prec_1 R_1 \) conflicts, they may not prevent a \( R_1 \prec_1 R_1 \) conflict, as the following example illustrates.

**Example 6:** Consider an MDBS consisting of two sites \( s_1 \) and \( s_2 \). Let data items \( x, y \in LD(s_1) \), and data item \( z \in LD(s_3) \). Let \( T_1 \) be a global transaction, and let \( T_2 \) be a local transaction that executes at site \( s_1 \).

\[ T_1 : r_1(x), r_1(y), w_1(z) \quad T_2 : w_2(x), w_2(y) \]

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 may result 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 $L(V(S_1))$, since $R_1 \leadsto T_2$, $R_1 \cong T_2$. Also, since $T_2 \leadsto R_1$, $T_2 \cong R_1$. Thus, $R_1 \cong R_1$ and hence $S_1$ is not M-serializable.

To prevent $R_1 \leadsto T_1$ 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 write on $x$ 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 on 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 on 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* [Breitbart et al. 1991] if it is $AROW + AWOR + AWOW$\(^{15}\). Similarly, a schedule is *strict* [Bernstein et al. 1987] if it is AWOR

\(^{14}\)The notation ROW is an acronym for *read ordered before write*. The notation should be interpreted as stating that the commit/abort order of transactions is analogous to the order of read-write conflicts between transactions. Similarly, WOR and WOW refer to *write ordered before read* and *write ordered before write* with the similar interpretation.

\(^{15}\)We use the symbol '$+$' to mean "and". For example, $AROW + AWOR$ means $AROW$ and $AWOR$. 

![Figure 2. Classification of Schedules](image)
Suppose that the GTM decides to commit $S$. In the following example we show that loss of M-serializability may occur even if $S$ is semi-rigorous if it is MSER. Figure 2 shows how the above properties relate to each other. Note that every region shown in the figure represents a non-empty set of schedules. For example, the following schedule illustrates a semi-rigorous schedule that is not rigorous.

$$S_1 : r_1(x), w_1(y), w_2(x), c_1, c_2$$

Later examples in the paper will illustrate other schedules belonging to various classes identified in the figure above.

**Theorem 6:** Let $S_j$ be a globally complete schedule resulting from the execution of local transactions and a single global subtransaction. If $S_j$ is semi-rigorous, then $S_j$ is M-serializable.

**Proof:** See Appendix F. □

Relaxing the requirement of $S_j$ to be semi-rigorous in Theorem 6 may result in a loss of M-serializability. To see this, consider the non M-serializable schedule $S_1$ in Example 6 which is AROW + WOR + AWOW. Since the schedule $S_1$ is not AWOR and thus not semi-rigorous, loss of M-serializability occurs. The following examples further illustrates that if $S_j$ is either not WOW or not ROW, then M-serializability may not be ensured.

**Example 7:** Consider an MDBS consisting of two sites $s_1$ and $s_2$. Let data items $x, y, z \in LD(s_1)$ and $u \in GD(s_2)$. Let $T_1$ be a global transaction and $T_2, T_3$ be local transactions that execute at $s_1$.

$$T_1 : r_1(x), r_1(y), w_1(u) \quad T_2 : w_2(z), w_2(y) \quad T_3 : w_3(z), w_3(x)$$

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 $T_1$. The above scenario results in the following local schedule at site $s_1$.

$$S_1 : w_2(x), w_3(x), c_3, r_1(x), r_1(y), w_1(y), c_2$$

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

In Example 7, the non M-serializable schedule $S_1$ is AWOR + AROW, thus illustrating the need for $S_j$ to be WOW. In the following example we show that loss of M-serializability may occur even if $S_j$ is AWOR + AWOW. Thus, relaxing the requirement of $S_j$ to be ROW may also result in a loss of M-serializability.

**Example 8:** Consider an MDBS consisting of two sites $s_1$ and $s_2$. Let data items $x, y, z \in LD(s_1)$ and $u \in GD(s_2)$. Let $T_1$ be a global transaction and $T_2, T_3$ be local transactions that execute at $s_1$.

$$T_1 : r_1(x), r_1(y), w_1(u) \quad T_2 : r_2(z), w_2(y) \quad T_3 : w_3(z), w_3(x)$$

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 $T_1$. The above scenario results in the following local schedule at site $s_1$.

$$S_1 : r_2(z), w_3(x), w_3(x), c_3, r_1(x), r_1(y), a_1, w_1(y), c_2$$

In $LV(S_1)$, since $R_1 \sim T_2, R_1 \not\sim T_2$. Similarly, we can derive $T_2 \not\sim T_3$ and $T_3 \not\sim R_1$. Thus, $R_1 \not\sim R_1$. Hence,
$S_j$ is not M-serializable. □

6.3 Restrictions On the Execution of Global Transactions

While Theorem 6 establishes conditions for ensuring M-serializability in the presence of a single global transaction, unfortunately it does not hold if multiple global subtransactions are present in $S_j$.

**Example 9:** 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 $s_1$.

$T_1 : r_1(x), w_1(z), w_1(u) \quad T_2 : r_2(y), w_2(x), w_2(v) \quad T_3 : w_3(y), r_3(z)$

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 redo transactions $T_4 : w_4(z)$ and $T_5 : w_5(x)$ for $T_1$ and $T_2$ respectively. The above scenario results in the following local schedule at site $s_1$.

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

In $S_1$, $W_1$ and $W_2$ are the redo transactions $T_4$ and $T_5$ respectively. In $LV(S_1)$, since $T_3 \leadsto T_4$ and $pair(T_4, R_1)$, $T_3 \leadsto M R_1$. Since $R_2 \leadsto T_3$, $R_2 \leadsto M T_3$. Thus, $R_2 \leadsto R_1$. Further, since $R_1 \leadsto T_5$ and $pair(T_5, R_2)$, $R_1 \leadsto M R_2$ and hence $R_1 \leadsto R_2$. Thus, $R_1 \leadsto R_1$, 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 9. 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 $\pi_4$; that is, $GS_j = S_j^4$, where

$$\pi_4 = Gcc \cup \{operations(T_i) \circ decision(W_i) \mid T_i \in Gcaw \} \cup \{R_i \mid T_i \in Gca\}.$$

To ensure M-serializability, the GTM needs to ensure that the schedule $GS_j$ is rigorous. 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\(^\text{16}\). In Example 9, $GS_1$ corresponding to the local schedule $S_1$ is as follows:

$GS_1 : r_1(x), w_1(z), r_2(y), [w_2(x)], c_4, c_5$

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.

**Theorem 7:** Let $S_j$ be a globally complete schedule resulting from the execution of local and global transactions. If $GS_j$ is rigorous and $S_j$ is semi-rigorous, then $S_j$ is M-serializable.

**Proof:** See Appendix E. □

\(^\text{16}\)Recall again that we are currently assuming that the local DBMSs support an operation interface. We will discuss how the properties of $GS_j$ can be ensured if the assumption does not hold after we have identified all the required properties.
6.4 Necessity of the Restrictions on $GS_j$

By Theorem 7, if the schedule $S_j$ is semi-rigorous, and the schedule $GS_j$ is rigorous, then the local schedule $S_j$ is M-serializable. Further, we have seen that even in presence of a single global transaction, relaxing the requirement of the schedule $S_j$ to be semi-rigorous may result in the loss of M-serializability (Examples 6, 7, 8). In this section, we show that relaxing the requirement of $GS_j$ to be rigorous may also result in a loss of M-serializability. In Example 9 the non M-serializable schedule $GS_j$ is ROW + AWOR + AWOW. Thus, Example 9 demonstrates that if $GS_j$ is not AROW, then it may result in a loss of M-serializability. The following examples further illustrate the requirement of $GS_j$ to be AWOW and AWOR for the purpose of ensuring M-serializability.

**Example 10:** Consider an MDBS consisting of two sites $s_1$ and $s_2$. Let data items $x \in ED(s_1), y \in GD(s_1)$ and $z \in LD(s_1)$, 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 $s_1$.

$T_1: w_1(y), w_1(x), w_1(u)$  
$T_2: r_2(z), w_2(x), w_2(v)$  
$T_3: w_3(z), r_3(y)$

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 $s_1$ aborts both $T_1$ and $T_2$ and schedules operations of $T_3$ 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 $T_4: w_4(y), w_4(x), T_5: w_5(x)$ for $T_1$ and $T_2$ respectively. The above scenario results in the following local schedule at site $s_1$.

$S_1: [w_1(y)], [w_1(x)], [r_2(z)], [w_2(x)], [w_2(z)], r_3(y), c_3, w_4(y), w_4(x), c_4, w_5(x), c_5$

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

In Example 10, the schedule $GS_1$ is as follows:

$GS_1 : w_1(y), w_1(x), r_2(z), w_2(x), c_4, c_5$

The above schedule $GS_1$ is AROW + AWOR + WOW. Thus, the example illustrates that if $GS_j$ is not AROW, it may result in a loss of M-serializability.

**Example 11:** Consider an MDBS consisting of sites $s_1$ and $s_2$. Let data items $x, y, z \in GD(s_1)$ and $u, v \in GD(s_2)$. Let $T_1$ and $T_2$ be global transactions and $T_3$ be a local transaction that executes at site $s_1$.

$T_1: w_1(x), w_1(y), w_1(u)$  
$T_2: w_2(z), r_2(x), w_2(v)$  
$T_3: r_3(y), r_3(z)$

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 $s_1$ aborts both $T_1$ and $T_2$ and schedules operations of $T_3$ 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 $T_4: w_4(x), w_4(y), T_5: w_5(z)$ for $T_1$ and $T_2$ respectively. The above scenario results in the following local schedule at site $s_1$.

$S_1: [w_1(x)], [w_1(y)], [w_2(z)], [r_2(x)], [w_4(x)], [r_3(y)], [r_3(z)], [w_5(z)], c_5$

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

In Example 11, the schedule $GS_1$ is as follows:

$GS_1 : w_1(x), w_1(y), w_2(z), r_2(x), c_4, c_5$
The above schedule $GS_j$ is AROW + WOR + AWOW. Thus, the example illustrates that if $GS_j$ is not AWOR, it may result in a loss of M-serializability.

6.5 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. Since the concurrency control protocols followed by the pre-existing local DBMSs may not generate semi-rigorous schedules, in this section, we examine mechanisms to relax the restrictions imposed on the local schedules to be semi-rigorous.

One way to relax the requirement of local DBMSs to produce semi-rigorous schedules is to impose further restrictions on the data items accessed by global transactions. We consider the following restrictions on the data items that global transactions can access:

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

Restricting transactions as above alone does not result in the relaxation of the requirement of $S_j$ to be semi-rigorous. For example, the global transactions in Example 6 satisfy the above listed restriction but loss of M-serializability occurs. To prevent loss of M-serializability, we need to further modify the GTM commit protocol as follows.

The GTM, in the first phase of the global commit protocol, instead of sending a $\langle \text{vote}_{\text{req}}, T_i \rangle$ message, sends a $\langle \text{commit}, T_i \rangle$ 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 $\langle \text{vote}_{\text{req}}, T_i \rangle$ message. An $r$-server, on receipt of a $\langle \text{commit}, T_i \rangle$ 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 $\langle \text{ack}_{\text{commit}}, T_i \rangle$ message to the GTM; if the subtransaction is aborted by the local DBMS the server sends a $\langle \text{ack}_{\text{abort}}, T_i \rangle$ message to the GTM. A $w$-server behaves as in the previous case and sends its vote to commit or to abort $T_i$. If the GTM receives a commit vote from each $w$-server and a $\langle \text{ack}_{\text{commit}}, T_i \rangle$ from each $r$-server, it decides to commit the transaction and sends a $\langle \text{commit}, T_i \rangle$ message to each of the $w$-servers. Else, if it receives a vote to abort $T_i$ from some $w$-server or an $\langle \text{ack}_{\text{abort}}, T_i \rangle$ from some $r$-server, it decides to abort the transaction and sends an $\langle \text{abort}, T_i \rangle$ 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 6 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 $T_1$. In fact, it is not too difficult to show that if the GTM uses the EC protocol to commit global transactions, M-serializability of schedules containing a single global transaction is preserved. Unfortunately, in the presence of multiple global subtransactions, unless additional restrictions are placed both on $GS_j$ and on $S_j$, M-serializability is not ensured even if the GTM follows the EC protocol. Consider the following example.


**Example 12:** 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 $s_1$.

$T_1 : r_1(x), w_1(y), w_1(u)$  
$T_2 : w_2(x), w_2(v)$  
$T_3 : w_3(z), r_3(x)$  
$T_4 : r_4(y), r_4(z)$

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 $T_5 : w_5(y)$ for $T_1$. 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 $s_1$.

$S_1 : r_1(x), [w_1(y)], a_1, w_3(z), r_4(y), r_4(z), c_4, w_5(y), c_5, w_2(x), c_2, r_3(x), c_3$

In $S_1$, $W_1$ is the redo transaction $T_5$. In $LV(S_1)$, 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 12, schedule $GS_1$ is serial and thus no restriction on $GS_j$ will prevent the non M-serializable execution above. To prevent the loss of M-serializability, $S_j$ is required to be strongly recoverable. Recall that a schedule is strongly recoverable, if it is recoverable as well as ROW + WOW.

**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 EC protocol, $GS_j$ is ROW + AWOR, and $S_j$ is strongly recoverable, then $S_j$ is M-serializable.

**Proof:** See Appendix E. \(\square\)

Since every semi-rigorous schedule is also strongly recoverable (see Figure 3), we have, by placing further restrictions on the data items accessed by the global transactions, effectively relaxed the requirement of $S_j$ to be semi-rigorous to strong recoverability. We next show that relaxing the requirement of $S_j$ to be strongly recoverable, or the requirement of $GS_j$ to be ROW + AWOR may result in a loss of M-serializability in sequence.

In Example 12, since $S_1$ is not WOR, it is not strongly recoverable resulting in a loss of M-serializability. In the following two examples, we further illustrate that if $S_j$ is either not ROW or not WOW, then loss of M-serializability may result.

**Example 13:** Consider an MDBS consisting of two sites $s_1$ and $s_2$. Let data items $x, y \in GD(s_1)$ and $u,v \in GD(s_2)$. Let $T_1$ and $T_2$ be global transactions and $T_3$ be a local transaction that executes at site $s_1$.

$T_1 : r_1(x), w_1(y), w_1(u)$  
$T_2 : w_2(x), w_2(v)$  
$T_3 : r_3(y), r_3(x)$

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 $T_3$. Since the GTM considers $T_1$ to be committed, the server at site $s_1$ executes the following redo transaction $T_4 : w_4(y)$ for $T_1$. 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 $s_1$.

$S_1 : r_1(x), [w_1(y)], a_1, r_3(y), w_4(y), c_4, w_2(x), c_2, r_3(x), c_3$

In $S_1$, $W_1$ is the redo transactions $T_4$. In $LV(S_1)$, since $R_1 \sim T_2$, $R_1 \sim T_2$. Since $T_2 \sim T_3$, $T_2 \sim T_3$. Since $T_3 \sim T_4$, $T_3 \sim R_1$. Thus, $R_1 \sim R_1$, hence $S_1$ is not M-serializable. \(\square\)
Note that in the above schedule $S_1$ is AWOR + AWOW, and $GS_1$ is rigorous. Thus, $S_1$ needs to be ROW to ensure M-serializability.

**Example 14:** Consider an MDBS consisting of two sites $s_1$ and $s_2$. Let data items $x \in GD(s_1)$, $y \in LD(s_1)$, and $u, v \in GD(s_2)$. Let $T_1$ and $T_2$ be global transactions and $T_3$ be a local transaction that executes at site $s_1$. $T_1 : r_1(x), w_1(y), w_1(u) \quad T_2 : w_2(x), w_1(v) \quad T_3 : w_3(y), r_3(x)$

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 $T_3$. Since the GTM considers $T_1$ to be committed, the server at site $s_1$ executes the following redo transaction $T_4 : w_4(y)$ for $T_1$. 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 $s_1$. $S_1 : r_1(x), [w_1(y)], a_1, w_3(y), w_4(y), c_4, w_2(x), c_2, r_3(x), c_3$ In $S_1$, $W_1$ is the redo transactions $T_4$. In $LV(S_1)$, since $R_1 \sim T_2$, $R_1 \sim T_2$. Since $T_2 \sim T_3$, $T_2 \sim T_3$. Since $T_3 \sim T_4$, $T_3 \sim R_1$. Thus, $R_1 \sim R_1$, hence $S_1$ is not M-serializable.

Note that in the above schedule $S_1$ is AROW + AWOR, and $GS_1$ is rigorous. Thus, $S_1$ needs to be WOW to ensure M-serializability.

We next show that the requirement of $GS_j$ to be ROW + AWOR in Theorem 8 cannot also be relaxed. Note that in Example 11, since the schedule $GS_1$ is not AWOR, loss of M-serializability occurred. Since the local schedule $S_1$ is serial, the example illustrates that the requirement of $GS_j$ to be AWOR cannot be relaxed. In the following example we further illustrate that the restriction on $GS_j$ to be ROW cannot, also, be relaxed.

**Example 15:** Consider an MDBS consisting of two sites $s_1$ and $s_2$. Let data items $x, y \in GD(s_1)$ and $u, v \in GD(s_2)$. Let $T_1$ and $T_2$ be global transactions and $T_3$ be a local transaction that executes at site $s_1$. $T_1 : r_1(x), w_1(y), w_1(u) \quad T_2 : w_2(x), w_1(v) \quad T_3 : r_3(x), r_3(y)$

Suppose that the GTM decides to commit both $T_1$ and $T_2$. Further suppose that $T_2$ commits at both $s_1$ and $s_2$ and $T_1$ commits at $s_2$, but the local DBMS at $s_1$ aborts $T_1$ and schedules operations of $T_3$ for execution. Since the GTM considers $T_1$ to be committed, the server at site $s_1$ executes the following redo transaction $T_4 : w_4(x), w_4(y)$ for $T_1$. The above scenario results in the following local schedule at site $s_1$. $S_1 : r_1(x), [w_1(y)], a_1, w_2(x), c_2, r_3(x), r_3(y), c_3, w_4(y), c_4$ In $S_1$, $W_1$ is the redo transactions $T_4$. In $LV(S_1)$, since $R_1 \sim T_2$, $R_1 \sim T_2$. Since $T_2 \sim T_3$, $T_2 \sim T_3$. Since $T_3 \sim T_4$, $T_3 \sim R_1$. Thus, $R_1 \sim R_1$, hence $S_1$ is not M-serializable.

6.6 Discussion

So far in this section we have identified conditions under which M-serializability, (that is, serializability of the local schedule in the viewpoint of the MDBS), can be ensured if the redo approach to recovery is used to ensure atomicity of the global transactions. For M-serializability to be ensured, we observed that three things need to be done:

— Restrictions need to be imposed on the data items accessed by global transactions.
— Local schedules are required to satisfy certain properties.
— Restrictions need to be imposed on the execution of global transactions.
Restrictions on the global transactions limit the generality of the redo approach. Furthermore, requirements of the local schedules restrict applicability of the solution to domains where local DBMSs ensure that the schedules satisfy the required properties. We first identified minimal restrictions that need to be imposed on the global transaction's data accesses for the redo approach to correctly ensure atomicity of the global transactions. However, under the minimal restrictions on the global transactions we established that for M-serializability to be ensured, the local schedule \( S_j \) must be semi-rigorous and the schedule \( GS_j \), which is a projection of \( S_j \) onto global transactions, must be rigorous (Theorem 7). Furthermore, we showed that relaxing either the restriction on \( GS_j \) or on \( S_j \) may result in a loss of M-serializability.

While rigorousness of \( GS_j \) can be ensured by the MDBS software (we will soon see how this can be achieved), semi-rigorousness of \( S_j \) cannot be directly ensured by the MDBS without controlling the execution of local transactions. Since the necessity of the restrictions on \( S_j \) holds only under the minimal restrictions on the nature of the global transactions, we imposed further restrictions on data accesses of the global transactions. Under the new restrictions on the global transactions, we showed that the requirement of the local schedule to produce semi-rigorous schedules can be relaxed to strong recoverability. The restrictions on transactions and schedules establish the theoretical boundaries within which the redo approach can be used to ensure global transaction atomicity. Whether or not a significant number of MDBS application domains lie within those boundaries and will thus gain directly by the usage of the redo approach requires careful study of the application domains and the nature of the global transactions that execute in those domains. Such a study is beyond the scope of this paper.

We now consider how the MDBS software can restrict the execution of the schedule \( GS_j \) to satisfy the properties identified in Theorems 7 and 8. The mechanism that can be used to ensure these properties of the schedule \( GS_j \) depends upon the nature of the interface supported by the local DBMSs. In the case that the local DBMSs support an operation interface and the MDBS software (that is, the GTM and the local servers) control the submission of the read and write operations of the global transactions to the local DBMS, ensuring that \( GS_j \) satisfies the requisite property is relatively straightforward. The server can ensure the required property by controlling the submission order of the operations belonging to the global transactions to the local DBMS for execution. For example, the server can ensure rigorousness of \( GS_j \) by disallowing submission of the operation \( o_i \) belonging to \( T_i \) to the local DBMS if it has already submitted a conflicting operation \( o_j \) belonging to \( T_j \) for execution to the local DBMS until \( T_j \) commits at the local DBMS.

Unfortunately, if the local DBMS does not support an operation interface, the MDBS has no direct control over the execution of the individual read and write operations. For example, if the local DBMS supports an SQL interface, the MDBS cannot exercise direct control over individual read and write operations that result at the local DBMS from the execution of the SQL statement. The MDBS can nevertheless ensure the required properties of \( GS_j \) by being pessimistic and controlling the submission order of the SQL statements to the local DBMSs. For example, in order to ensure rigorousness, the SQL statement corresponding to a transaction will be blocked at the server and not submitted to the local DBMS if its execution may result in an operation
that conflicts with some other operation that results from a previously submitted SQL statement of another transaction. Note that the server can exploit the information about the relations being accessed by the SQL statement, and the predicates associated with the statement to decide whether or not two statements may result in conflicting operations at the local DBMSs. In the worst case, the server can assume that any two requests for different global transactions result in conflicting operations at the local DBMS (such an assumption may be required in case of the service interface, if no information about the application that executes on the service request is available).

Notice that the redo approach makes a fundamental assumption that it is always possible to construct an appropriate redo transaction for an aborted global subtransaction. Again, if the local DBMS supports an operation interface such an assumption is reasonable since each read and update operation of the global transaction is processed by the MDBS software. Thus, a local server can store all the update values, along with the identity of the object being updated, in its server logs, which can be used to construct the redo transaction. Unfortunately, this may not be possible if the interface is not an operation interface (say it is an SQL interface). Simply executing the SQL statements corresponding to the transaction again will not be correct since the reexecuted SQL statements may not see the same state of the database as seen by the transaction when it had originally executed. As a result, reexecuting SQL statements as a redo transaction may result in a different set of updates as compared to the original execution.

A way to construct the correct redo transaction that consists of exactly the same updates as had originally executed, is to modify the SQL statements of the original global transaction such that they return all the data they read at the local DBMSs to the MDBS software (that is, the local servers). The local servers store the data that the original transaction reads into its local logs. Furthermore, it also stores the original SQL statements that are submitted to the local DBMSs for execution in its logs. On failure, using the stored SQL statements, and the values read by the SQL statement at the local DBMSs when it originally executed, the server constructs a redo transaction that consists of exactly the same updates as had been performed by the original subtransaction.

The strategy is illustrated by the following example:

**Example 16:** Consider a local DBMS at site $s_1$ that supports an SQL interface. Consider a global transaction $T_1$ that at site $s_1$ reads records of all employees who work in the Murray Hill location from the $emp$ relation and inserts their names and salaries into another relation $MH-Emp$. The SQL statement corresponding to the subtransaction is as follows:

```sql
insert into MH-emp
select name, salary
from emp
where emp.location = "Murray Hill"
```

Such an SQL statement will be restructured as follows:
select name, salary
from emp
where emp.location = "Murray Hill"
insert into MH-emp
select name, salary
from emp
where emp.location = "Murray Hill"

The values returned by the SQL query will be stored by the local server at site $s_1$ in the server logs. Let us assume that the local DBMS returns the following as a result of the evaluation of the query:

Paul 50K
Ringo 10K
George 50K

In the case of a failure of the subtransaction, the server will construct the following redo transaction using the above table and the original SQL statement:

insert into MH-emp
values {
(Paul, 50K), (Ringo, 10K), (George, 50K)
}

Notice that the above redo transaction will result in exactly the same updates to the local DBMS as had resulted from the original subtransaction.

7. ENSURING ORDER PRESERVATION

In the previous two sections, we have developed mechanisms for ensuring serializability of $\mathcal{N}$, and M-serializability of the local schedules. In order to ensure global serializability in the presence of failures, we also need to ensure that the local schedules at each local DBMSs satisfy the order preservation property. To ensure order preservation property, the server needs to ensure that if $T_i$ is serialized before $T_j$, then $\text{ser}_{s_k}(T_i) \prec_{s_k} \text{ser}_{s_k}(T_j)$. The server can do so using either of the following two approaches:

—pessimistic approach: The server at site $s_k$ delays the execution of $\text{ser}_{s_k}(T_i)$ until all the transaction that have previously executed $\text{ser}_{s_k}(T_j)$ have committed at the local DBMS at $s_k$.

—optimistic approach: The server allows the execution of $\text{ser}_{s_k}(T_i)$ operation but delays its vote to commit transaction $T_i$ until all transactions $T_j$ for which it has previously executed a $\text{ser}_{s_k}(T_j)$ operation have committed at the local DBMS. If, in case, the subtransaction of $T_j$ at site $s_k$ (that is, $T_{jk}$) is aborted by the local DBMS at $s_k$ after the GTM decides to commit $T_j$, then before the redo transaction for the subtransaction $T_{jk}$ executes, the subtransaction $T_{ik}$ is aborted by the server.

Using either of the above two approaches to ensuring order preservations, along with the mechanisms for ensuring serializability of $\mathcal{N}$ discussed in Section 5, and M-serializability of local schedules discussed in Section 6, global serializability can be ensured in MDBSs in the presence of failures.

8. RELATED WORK

The initial work on multibase systems did not consider the transaction management issues and, as a result, most of the prototype systems that were built (e.g., MULTIBASE [Landers and Rosenberg 1982], MERMAID [Templeton et al. 1983], ADDS [Breitbart and Tieman 1985]) permitted global transactions to only retrieve
data. Also, these systems did not have any concurrency control schemes to coordinate the execution of global transactions. Therefore, as demonstrated in Example 1, it was possible in such systems for global schedules to be non-serializable and global queries to retrieve inconsistent data. Transaction management issues in MDBS environments were first discussed by Gligor and Popescu-Zeletin [Gligor and Popescu-Zeletin 1986] in which the authors outlined the basic requirements and the inherent difficulties in transaction management in MDBSs. Since then, active research has been done to overcome the problems that arise in ensuring serializability, as well as ensuring atomicity of global transactions.

**Concurrency Control.** Research on concurrency control in MDBSs has been done along two complementary directions. Significant work has gone into developing techniques to relax the serializability requirement (e.g., [Du and Elmagarmid 1989; Wu et al. 1992; Mehrotra et al. 1991]). Furthermore, extensive research has been conducted to develop mechanisms for ensuring serializability in MDBSs [Breitbart and Silberschatz 1988; Georgakopoulos et al. 1991; Batra et al. 1992; Elmagarmid and Du 1990; Mehrotra et al. 1992a; Pu 1988; Raz 1992].

Using the framework developed in this paper, schemes proposed in the literature that ensure global serializability can be viewed as adaptations of the concurrency control schemes for traditional databases to the MDBS environment. For example, the scheme proposed in [Elmagarmid and Du 1990] can be viewed as the TO scheme for ensuring serializability of the schedule $\hat{S}$. Whereas, schemes proposed in [Pu 1988; Georgakopoulos et al. 1991] can be viewed as the SGT certification schemes [Bernstein et al. 1987] for ensuring serializability of $\hat{S}$. In [Batra et al. 1992], the authors develop a distributed concurrency control technique for ensuring serializability of $\hat{S}$.

Besides the above approaches to ensuring global serializability, certain other techniques have been developed under the assumption that the schedules produced by the local DBMSs satisfy certain specific properties. Examples of such approaches include [Breitbart et al. 1991; Raz 1992], where the authors develop techniques under the assumption that the schedules are *rigorous*, *strongly recoverable*, and *commit ordered* respectively. The notion of commit ordered is similar to that of strong recoverability. A schedule is commit ordered if the serialization order of the transactions is analogous to their commitment order. In case the local schedules are strongly recoverable/commit ordered, then the papers show that serializability can be ensured by only controlling the order in which transactions commit [Breitbart and Silberschatz 1992; Raz 1992].

Besides the schemes that ensure global serializability, a large body of work exists on relaxing the serializability requirements for MDBS environments. This work can be classified into the following three approaches:

---

**Schemes that Exploit the Knowledge of Integrity Constraints:** [Du and Elmagarmid 1989; Mehrotra et al. 1991; Rastogi et al. 1993]. These schemes are based on the assumption that the integrity constraint in an MDBS environment are of a restricted nature. The fundamental technique used is to partition the set of data in an MDBS into relatively independent subdatabases, each with its own consistency constraints. The restricted nature of integrity constraints between these subdatabases, as well as the restrictions on the local and global transactions permitted in the system are exploited to relax the serializability requirement. Different schemes
based on this approach differ from each other in the class of schedules they permit, and the assumptions they make about the MDBS model.

**Schemes that Exploit the Semantics of Transactions:** [Garcia-Molina 1983; Farrag and Oszu 1989; Rastogi et al. 1992]. These schemes consider a global transaction to consist of a number of subtransactions with each of which a type is associated. It is assumed that the application administrator *a priori* specifies the various subtransaction types along with the set of interleavings of the subtransactions that does not result in a loss of database consistency. A transaction manager utilizes this specification to permit only acceptable, and prevent unacceptable interleavings of the transactions. Different schemes in this category differ from one another in the mechanism they employ to specify the interleavings and the algorithm that the transaction manager uses to ensure that undesirable interleavings are not permitted.

**Schemes that Tolerate Bounded Inconsistency:** [Pu and Leff 1991; Wu et al. 1992; Wong and Agrawal 1992]. These schemes relax the serializability requirement in that they permit transactions to interleave as long as the degree of inconsistency introduced due to the interleaving of the transactions is bounded. Different schemes in this category differ from each other in the mechanism they use to quantify the degree of inconsistency (for example, in [Pu and Leff 1991], the degree of inconsistency is quantified as the count of the number of conflicts a read-only query is involved in, which if not present, would render the schedule serializable).

The assumptions made by the above approaches limit their potential applicability. For example, schemes that tolerate inconsistency may be applicable only in domains where exact values of data are not required and uncertainty and inconsistency in the data is tolerable (e.g., statistical information gathering systems). Similarly, schemes based on exploiting transaction semantics can be used only in domains where the types and semantics of the transactions that execute are known *a priori* and where user-defined transactions are not permitted (e.g., in systems that provide a fixed set of services). Finally, schemes that exploit knowledge of integrity constraints are applicable in systems where the set of integrity constraints as well as the nature of local and global transactions are appropriately restricted.

Another issue with the schemes based on exploiting the consistency constraints in the system is that the preservation of integrity constraints, by itself, may not be a sufficient consistency guarantee for transactions [Mehrotra et al. 1992a]. Consider, for example, an MDBS consisting of two banking databases located at sites $s_1$ and $s_2$. Further, let $A_1$ and $A_2$ be two accounts belonging to banking databases at sites $s_1$ and $s_2$ respectively such that there is no integrity constraint that relates the two accounts. In such a case, if a transaction that transfers money from one account to the other executes concurrently with an audit transaction that reads both the accounts, then it is possible that the audit transaction sees a sum that differs from the true balance of the two accounts. Thus, even though each transaction may see a consistent database state (i.e., a state in which no integrity constraint is violated) and the final state of the database is consistent, the execution is still undesirable since the audit transaction sees a wrong sum of balance in the accounts.

**Ensuring Atomicity.** Approaches that have been studied for ensuring atomicity of transactions, in the absence of the 2PC protocol, can be classified into three types:
—Redo Approach: The redo approach, as discussed in this paper, was originally introduced in [Breitbart et al. 1990]. A transaction management technique for MDBS environments was developed using the redo approach under the assumption that each local DBMS follows a strict 2PL protocol for concurrency control. The restrictions imposed on global transactions were more severe compared to the restrictions imposed in this paper. The redo approach was also independently developed by [Wolski and Veijalainen 1990]. A decentralized transaction management scheme was developed under the assumption that each local DBMS produces rigorous schedules.

—Retry Approach: Unlike the redo approach where only the updates are resubmitted for execution, in the retry approach, the entire global subtransaction is resubmitted for execution at the local DBMS. Commitment of the resubmitted subtransaction establishes atomicity of the transaction. The approach assumes that the subtransaction will commit within a bounded number retrials. Since different retrials of the same subtransaction may observe different database states, care must be taken if the updates made by some other subtransaction of the transaction are dependent upon the values read by the failed subtransaction. Retry approach has been studied in [Mehrotra et al. 1992b; A. et al. 1994].

—Compensate Approach: Unlike the redo and the retry approach that strive to commit all the subtransactions of the partially-committed global transactions, in the compensate approach, attempt is made to roll-back the effects of such a transaction by executing compensating transactions to undo the effects of the committed subtransactions. The compensate approach has been studied in [Levy et al. 1991; Muth and Rakow 1991; Mehrotra et al. 1992b]. The compensate approach depends upon the existence of a suitable compensating transaction to undo the effects of the committed subtransaction. Notice that since the effects of the committed subtransaction may have been observed by other transactions before the compensating transaction executes, a compensating transaction is not simply an inverse of the original forward subtransaction. While most of work on compensate approach has assumed that the compensating transactions for subtransactions are known a priori, some work on automatically generating compensating transactions exists in [Korth et al. 1990].

An interesting characteristic of the above described approaches to ensuring atomicity of global transactions is that these approaches are complementary and can be supported together in the same system. This fact has been exploited in [Mehrotra et al. 1992b] to develop a recovery mechanism for MDBS environments that combines each of the above three approaches. This is important since each of the recovery mechanisms make orthogonal assumptions about the nature of the transactions and schedules in the MDBS environment and can only be used under appropriate conditions. Combining them into a single paradigm results in a powerful model for recovery in MDBS environments. Other work that uses such a combined approach to recovery includes [A. et al. 1994].

Another interesting observation is that the above recovery techniques can also be combined with the basic 2PC protocol. Hence it is possible to integrate local DBMSs that support a prepared state and are willing to participate in the 2PC with local DBMSs that are unwilling/incapable of participating in the 2PC and which use the above mentioned recovery techniques for transaction atomicity. Obviously, the subtransactions that execute at the autonomous local DBMSs will have to be appropriately restricted such that the recovery technique (that is,
redo, retry, and compensation) associated with the subtransaction can be used. Furthermore, it is also possible to construct MDBS environments in which local DBMSs allow certain global transactions to use the basic 2PC protocol for atomicity (and thus no restrictions need to be imposed on such transactions), while the other global transactions use the redo, retry or the compensate approaches to ensuring global transaction atomicity.

Restricted Autonomy. Unlike the mechanism developed in this paper and the various schemes discussed above that strive to preserve local autonomy, some schemes have been developed that only preserve a restricted notion of autonomy. In [Pu 1990; Tal and Alonso 1992], the authors assume that each participating local DBMS supports a prepared state for the execution of the atomic commit protocol. Systems may, however, be following heterogeneous commit protocols. Thus, the problem of interoperability among databases reduces to designing appropriate gateways that translate between these heterogeneous commit protocols supported by the local DBMSs. Design of such gateways in case local DBMSs support different implementations of the two-phase and three-phase commit protocols [Skeen 1982] have been studied in [Pu 1990; Tal and Alonso 1992]. In [Perrizo et al. 1991], the authors describe mechanisms for simulating a prepared state for local DBMSs by rerouting both the local and global transactions to a common interface which can be controlled by the MDBS software. Once the prepared state is simulated, the 2PC protocol can be used to ensure atomicity of global transactions. Unlike approaches based on preserving autonomy, the objective of these schemes is to identify minimal changes that can be made to the local DBMSs such that the standard protocols used in distributed databases can be used to ensure consistency.

9. CONCLUSIONS

In this paper, we studied the problem of transaction management in multidatabase systems (MDBSs). An MDBS is an integration of a number of existing database management systems (local DBMSs) that may belong to different autonomous business organizations and may be dispersed over the nodes of a world-wide computer network. The problem of transaction management in MDBSs is to develop appropriate software, on top of the existing transaction management software of the local DBMSs, that enables the users to execute transactions that span multiple local DBMSs without resulting in the violation of database consistency. We developed a mechanism using which the problem of ensuring global serializability in an MDBS environment reduces to the problem of ensuring serializability in a traditional database system. Since concurrency control in traditional database systems is a well studied problem, the development of concurrency control schemes for MDBSs is simplified. To address the problem of ensuring atomicity of the global transactions, we studied the redo approach to recovery that can be used to ensure the atomicity of the global transactions in an MDBS environment without the requirement of an atomic commit protocol. If an atomic commit protocol is not used to process the commit of a global transaction, it is possible that certain subtransactions of a global transaction commit, whereas others abort. The redo approach ensures atomicity of the transaction by resubmitting the updates of the aborted subtransaction of the global transaction for execution to the local DBMS as a separate new transaction. We showed that if the redo approach to recovery is used, the task of ensuring serializability in the presence of failures reduces to solving three independent subproblems.
—Ensuring serializability of the schedule $\hat{S}$.
—Ensuring schedules at the local DBMSs to be $M$-serializable.
—Ensuring that the schedules at the local DBMSs satisfy the order preservation property.

We study mechanisms for solving each of the above subproblems. The developed mechanisms together constitute a complete strategy for failure-resilient transaction management in MDBS environments where preservation of local autonomy is important.

References


APPENDIX

A. PROOF OF THEOREM 1

To show the necessity of the conditions, we need to establish that if $S$ is globally serializable, then there exists a order $\prec_G$ among global transactions that satisfies the condition specified in the theorem. Consider a globally serializable schedule $S$. Let $T_1, T_2, \ldots, T_n$ be global transactions and $T_{n+1}, T_{n+2}, \ldots, T_m$ be local transactions in $S$. By definition of serializability, since $S$ is globally serializable, there exists a total order $\prec_S$ such that for all transactions $T_i, T_j$, $i, j \in [1, m]$, if $T_i \prec_S T_j$, then $T_j \prec T_i$ in $S$. Specifically, for all global transactions $T_i, T_j$, $i, j \in [1, n]$, for all sites $s_k$, if $T_i \prec_S T_j$, then $T_j \prec T_i$ in $S_k$. As a result, $\prec_S$ (projected to global transactions) satisfies the requirement of total order $\prec_G$ in the theorem above. Hence, if $S$ is globally serializable, then there exists a total order $\prec_G$ among global transactions such that at each site $s_k$, for all pairs of global transactions $T_i, T_j$ executing at site $s_k$, if $T_i \prec_G T_j$, then $T_j \prec T_i$ in $S_k$.

We next establish the sufficiency of the condition specified in the theorem. We need to show that if a total order $\prec_G$ exists, then the schedule $S$ is globally serializable. Assume that $S$ is not serializable. Thus, since each of the local schedules are serializable, there must exist global transactions $T_1, T_2, \ldots, T_n$, $n \geq 2$, such that $T_1 \prec T_2$ at site $s_{i_1}$, $\ldots$, $T_{n-1} \prec T_n$ at site $s_{i_{n-1}}$, $T_n \prec T_1$ at site $s_{i_n}$. Since $T_1 \prec T_2$ at site $s_{i_1}$, it must be the case that $T_1 \prec_G T_2$. Similarly, $T_2 \prec_G T_3$, $\ldots$, $T_n \prec_G T_1$. That is, $T_1 \prec_G T_2 \prec_G \ldots \prec_G T_n \prec_G T_1$ which is a contradiction since $\prec_G$ is a total order. Thus, $S$ must be globally serializable. □

B. PROOF OF THEOREM 2

Before proving Theorem 2, we first prove the following lemma that illustrates a sufficient condition for ensuring global serializability. In the proof, we denote the set of sites at which a global transaction $T_i$ executes by $exec(T_i)$.

**Lemma 1:** Consider an MDBS where each local schedule is serializable. Global serializability is ensured if there exists a total order $\prec_G$ on global transactions such that at each site $s_k$, for all pairs of global transactions $T_i, T_j$ executing at site $s_k$, if $ser_{S_i}(T_i) \prec ser_{S_j}(T_j)$, then $T_i \prec_G T_j$.

**Proof:** Let us assume that global schedule $S$ is not serializable. Since each of the local schedules is serializable, there must exist a cycle consisting of global transactions, say, $T_1, T_2, \ldots, T_r$, $r > 1$, such that $T_{i_1}$ is serialized before $T_{i_2}$ at site $s_{i_1}$, $T_{i_2}$ is serialized before $T_{i_3}$ at site $s_{i_2}$, $\ldots$, $T_{i_r}$ is serialized before $T_{i_1}$ at site $s_{i_r}$. If $T_{j_1}$ is serialized before $T_{j_2}$ at site $s_{j_1}$, then by the definition of serialization functions, $ser_{S_{i_1}}(T_{i_1}) \prec ser_{S_{i_2}}(T_{i_2}) \prec ser_{S_{i_3}}(T_{i_3}) \prec \ldots \prec ser_{S_{i_r}}(T_{i_1})$ — which implies $T_j \prec_G T_k$. As a result, $T_1 \prec_G T_2 \prec_G \ldots \prec_G T_r \prec_G T_1$ holds — a contradiction, since $\prec_G$ is a total order. Therefore, $S$ is serializable. □

**Proof of Theorem 2:** Assuming that $\hat{S}$ is serializable, by Lemma 8, it suffices to show that there exists a requisite total order $\prec_G$ on global transactions. That is, we exhibit a total order $\prec_G$ on global transactions such that for each site $s_k$, for all global transactions $T_i, T_j$ that execute at $s_k$, if $ser_{S_i}(T_i) \prec ser_{S_j}(T_j)$ then $T_i \prec_G T_j$. Since $\hat{S}$ is serializable, there exists a total order $\prec_{\hat{S}}$ on all the transactions $\hat{T}_i$ such that for all sites $s_k$, for all transactions $T_i, T_j$ that execute at $s_k$, if $ser_{S_i}(T_i) \prec ser_{S_j}(T_j)$ then $\hat{T}_i \prec_{\hat{S}} \hat{T}_j$. Notice that the above follows from the definition of serializability (e.g., see [Bernstein et al. 1987]) given that in the schedule $\hat{S}$ two
operations \( \text{ser}_{S_k}(T_i) \) and \( \text{ser}_{S_k}(T_j) \) are defined to conflict. Thus, \( S \) can be shown to be serializable by defining \( \prec_G \) as follows: \( T_i \prec_G T_j \) if and only if \( \hat{T}_i \prec_G \hat{T}_j \). □

C. PROOF OF THEOREM 3

To prove Theorem 3, we need to define a notion of a conflict sequence over transactions in schedule \( LV(S_j) \).

**Definition 7**: A sequence \( CS = T_1, T_2, \ldots, T_n, n \geq 2 \), of transactions in \( \pi_2 \) is said to be a conflict sequence if it satisfies both of the following:

- For all \( T_i, T_{i+1} \) in \( CS \), \( 1 \leq i < n \), either \( T_i \sim T_{i+1} \) or \( \text{pair}(T_i, T_{i+1}) \).
- For all \( T_{i-1}, T_i, T_{i+1} \) in \( CS \), \( 1 < i < n \), either \( \neg \text{pair}(T_{i-1}, T_i) \) or \( \neg \text{pair}(T_i, T_{i+1}) \). □

We further need to define the following functions \( LVtoS \), \( LVtoGV \) and \( GVtoLV \) that map transactions in the schedules \( S_j \), \( LV(S_j) \) and \( GV(S_j) \) to the corresponding transactions in the other schedules. Recall that the schedule \( GV(S_j) \) represents the execution in schedule \( S_j \) from the global view in which operations belonging to the pair \( R_t \) and \( W_t \) are considered to be part of the same transaction. Furthermore, schedule \( LV(S_j) \) represents the schedule \( S_j \) from the local DBMS's (local) view in which the redo transactions corresponding to aborted global transactions are considered as separate transactions. The functions \( LVtoS \), \( LVtoGV \) and \( GVtoLV \) map transactions in one view to their counterparts in the other view. For example, the function \( LVtoGV \) maps a transaction \( R_t \) or a \( W_t \) in \( LV(S_j) \) to its corresponding transaction \( R_t \circ W_t \) in the global view defined by \( GV(S_j) \). The formal definitions of the three functions is given below. In the definitions, \( \text{operations}(R_t) \circ W_t \) denotes the set \{operations\}(\( R_t \) \circ \( W_t \) | \( T_i \in \text{Gca} \)).

\( LVtoS : \pi_2 \rightarrow \text{Gca} \cup \text{Le} \cup \text{Gca} \).

\( LVtoS(T) = \begin{cases} T_k & \text{if } T \in \text{Rca} \text{ and } T = R_k \\ T_k & \text{if } T \in \text{Wca} \text{ and } T = W_k \\ T & \text{otherwise} \end{cases} \)

\( LVtoGV : \pi_2 \rightarrow \pi_1 \).

\( LVtoGV(T) = \begin{cases} \text{operations}(R_k) \circ W_k \text{ if } LVtoS(T) = T_k \in \text{Gca} \\ T \text{ otherwise} \end{cases} \)

\( GVtoLV : \pi_1 \rightarrow 2^{\pi_2} \).

\( GVtoLV(T) = \begin{cases} \{R_k, W_k\} \text{ if } T \in \text{Rwc} \text{ and } T = \text{operations}(R_k) \circ W_k \\ \{T\} \text{ otherwise} \end{cases} \)

Our strategy for proving Theorem 3 is as follows: we first establish that a schedule \( S_j \) is M-serializable if there does not exist transactions \( T_1, T_2, \ldots, T_n \) such that \( T_1, T_2, \ldots, T_n, T_1 \) is a conflict sequence. We then show that such conflict sequences do not exist if the M-conflict relation is acyclic. Recall that the schedule \( S_j \) is M-serializable, if for all transactions \( T_i \) in \( GV(S_j) \), \( T_i \not\prec T_i \) (see definition of M-serializability). In the following lemma, we establish a relationship between the conflicts between transactions in the schedule \( GV(S_j) \) and the conflict sequences defined over transactions in \( LV(S_j) \).

**Lemma 2**: Let \( T_1, T_n \in \pi_1, T_i \in GVtoLV(T_1) \) and \( T_i \in GVtoLV(T_n) \). If \( T_i \sim T_n \) in \( GV(S_j) \), then there exist transactions \( T_i', T_2', \ldots, T_n', \pi_2 \) such that \( CS = T_1, T_1', T_2', \ldots, T_n', T_1 \) is a conflict sequence.

**Proof**: By induction on the length \( m \) of the derivation of \( T_1 \sim T_n \).

**Basis (m = 1)**: Thus, \( T_1 \sim T_n \). There are four cases to consider.
\( -(T_1, T_n \notin RWc): GV_{toLV}(T_1) = \{T_1\} \) and \( GV_{toLV}(T_n) = \{T_n\} \). Note that \( T_1, T_n \) is a conflict sequence.

\( -(T_1 \notin RWc \text{ and } T_n \in RWc): \) Let \( T_n = R_k \circ W_k \). \( GV_{toLV}(T_1) = \{T_1\} \) and \( GV_{toLV}(T_n) = \{R_k, W_k\} \). If \( T_1 \sim R_k \), then \( T_1, R_k \) and \( T_1, R_k, W_k \) are conflict sequences. Else, if \( T_1 \sim W_k \), then \( T_1, W_k \) and \( T_1, W_k, R_k \) are conflict sequences.

\( -(T_1 \in RWc \text{ and } T_n \notin RWc): \) Let \( T_1 = R_k \circ W_k \). \( GV_{toLV}(T_1) = \{R_k, W_k\} \) and \( GV_{toLV}(T_n) = \{T_n\} \). If \( R_k \sim T_n \), then \( R_k, T_n \) and \( W_k, R_k, T_n \) are conflict sequences. Else, if \( W_k \sim T_1 \), then \( W_k, T_1 \) and \( R_k, W_k, R_k, T_n \) are conflict sequences.

\( -(T_1 \in RWc \text{ and } T_n \in RWc): \) Let \( T_1 = R_k \circ W_k \) and \( T_n = R_l \circ W_l \). \( GV_{toLV}(T_1) = \{R_k, W_k\} \) and \( GV_{toLV}(T_n) = \{R_l, W_l\} \). If \( R_k \sim W_l \), then \( R_k, W_l, R_k, W_l \) and \( W_k, R_k, W_l, R_k, W_l \) are conflict sequences. If \( W_k \sim W_l \), then \( W_k, W_l, R_k, W_l, R_k, W_k, W_l \) and \( R_k, W_k, R_k, W_l, R_k, W_k, W_l \) are conflict sequences.

**Induction:** Assume that the lemma holds for conflicts of length \( m \) + 1. Let the length of \( T_1 \prec T_n \) be \( m + 1 \) and further \( T_1 \sim T_2 \) and \( T_2 \prec T_n \), where length of the derivation of \( T_2 \prec T_n \) is \( m \). Let \( T_q \in GV_{toLV}(T_2) \). By IH, there exist transactions \( T_{i_1}^T, T_{i_2}^T, \ldots, T_{i_r}^T \in T_2 \), such that \( CS^T = T_q, T_{i_1}^T, T_{i_2}^T, \ldots, T_{i_r}^T, T_1 \) is a conflict sequence. Further, by base case, there exist transactions \( T_{i_1}^T, T_{i_2}^T, \ldots, T_{i_r}^T \in T_2 \), such that \( CS^{T_1} = T_q, T_{i_1}^T, T_{i_2}^T, \ldots, T_{i_r}^T, T_1 \) is a conflict sequence. If either, \( \neg\text{pair}(T_{i_2}^T, T_q) \), or \( \neg\text{pair}(T_q, T_{i_1}) \), then \( T_1, T_{i_1}^T, T_{i_2}^T, \ldots, T_{i_r}^T, T_1, T_2, \ldots, T_n \) is a conflict sequence. If, however, \( \text{pair}(T_{i_2}^T, T_q) \) and \( \text{pair}(T_q, T_{i_1}) \), then \( T_1, T_{i_1}^T, T_{i_2}^T, \ldots, T_{i_r}^T, T_1, T_{i_2}^T, \ldots, T_{i_r}^T, T_1 \) is a conflict sequence. □

**Lemma 3:** \( S_j \) is M-serializable if there does not exist transactions \( T_1, T_2, \ldots, T_n \in T_2 \) such that \( T_1, T_2, \ldots, T_n, T_1 \) is a conflict sequence.

**Proof:** Assume that there does not exist \( T_1, T_2, \ldots, T_n \in T_2 \) such that \( T_1, T_2, \ldots, T_n, T_1 \) is a conflict sequence. If \( S_j \) is not M-serializable, then \( GV(S_j) \) is not serializable. Thus, there exists a \( T_i \in T_2 \) such that \( T_i \sim T_i \). Let \( T_i \in GV_{toLV}(T_i) \). By Lemma 2, there exist transactions \( T_{i_1}^T, T_{i_2}^T, \ldots, T_{i_r}^T \in T_2 \) such that \( T_{i_1}^T, T_{i_2}^T, \ldots, T_{i_r}^T, T_1 \) is a conflict sequence which is a contradiction. Hence, \( S_j \) is M-serializable. □

Above, we have established that a schedule \( S_j \) is M-serializable, if the set of transactions in the schedule \( LV(S_j) \) do not form any cyclic conflict sequences (that is, there does not exist transactions \( T_1, T_2, \ldots, T_n \in T_2 \) such that \( T_1, T_2, \ldots, T_n, T_1 \) is a conflict sequence). We next establish that such conflict sequences do not appear if the M-conflict relation is acyclic. The proof of this will use the following lemma that relates the notion of conflict sequences to the M-conflict relation.

**Lemma 4:** If \( CS = T_1, T_2, \ldots, T_n \), where \( T_1, T_2, \ldots, T_n \in T_2 \), is a conflict sequence such that there exists a \( T_i, 1 \leq i < n, \neg\text{pair}(T_i, T_{i+1}) \), then \( T_1 \prec T_n \).

**Proof:** The proof is by induction on the number of pairs \( m \) in \( CS \).

**Basis (\( m = 0 \))** Since \( CS \) contains no pairs, by definition of conflict sequence, \( T_1 \sim T_2 \sim \cdots \sim T_n \). Since, by definition of \( \prec \), for all \( T_i, T_{i+1}, 1 \leq i < n, T_i \prec T_{i+1} \), it can be trivially shown by induction over \( n \) that \( T_1 \prec T_n \). Thus, the basis holds.
**Induction:** Assume the lemma is true for conflict sequences containing \( m \) pairs. We prove the lemma for conflict sequences containing \( m+1 \) pairs. Let \( CS = T_1, T_2, \ldots, T_n \) be a conflict sequence containing \( m+1 \) pairs. Without loss of generality, \( CS = T_1, T_2, \ldots, T_r, T_{r+1}, \ldots, T_n \), where \( T_r, T_{r+1} \) is a pair, and \( T_{r+2}, \ldots, T_n \) contains \( m \) pairs. We need to consider two cases.

• \((r = 1)\): Since \( CS \) is a conflict sequence and there exists a \( T_i, 1 \leq i < n, \neg \text{pair}(T_i, T_{i+1}), n \geq 3 \). By definition of M-conflict, since \( T_2 \sim T_3 \) and \( \text{pair}(T_1, T_2) \), we have that \( T_1 \sim M T_3 \). Hence, if \( n = 3 \), then \( T_1 \sim T_n \). Else, if \( n = 4 \), there are two cases. If \( T_3 \sim T_4 \), then by definition of M-conflict, \( T_3 \sim T_4 \). Since \( T_1 \sim T_3 \), we have that \( T_1 \sim T_4 \). That is, \( T_1 \sim T_n \). On the other hand if \( \text{pair}(T_3, T_4) \), then by the definition of M-conflict \( T_1 \sim M T_3 \) and hence \( T_1 \sim T_n \). If \( n > 4 \), then since \( T_3, T_4, \ldots, T_n \) is a conflict sequence with \( m \) pairs, by IH, \( T_3 \sim T_n \). Since \( T_1 \sim T_3 \), we have that \( T_1 \sim T_n \).

• \((r > 1)\): By definition of M-conflict \( T_{r-1} \sim T_{r+1} \). If \( r + 1 = n \), we have \( T_{r-1} \sim T_n \). Else, if \( r < n - 1 \), \( T_{r+1}, T_{r+2}, \ldots, T_n \) is a conflict sequence. Hence by IH, \( T_{r+1} \sim T_n \). Since \( T_{r-1} \sim T_{r+1} \) and \( T_{r+1} \sim T_n \), we have that \( T_{r-1} \sim T_n \). If \( r - 1 = 1 \), then \( T_1 \sim T_n \). Else, if \( r - 1 > 1 \), then since \( T_1 \sim T_2 \sim \ldots T_{r-1} \), using simple induction we can show that \( T_1 \sim T_{r-1} \). Hence, we get \( T_1 \sim T_n \). □

Having established the relationship between the M-conflict relation and the notion of conflict sequences, we can now prove Theorem 3.

**Proof of Theorem 3:** We need to show that \( S_j \) is M-serializable if for all \( T_i \in \tau_i, T_i \not\sim M T_i \). Assume that for all \( T_i \in \tau_i, T_i \not\sim M T_i \). If \( S_j \) is not M-serializable, then by Lemma 3, there exist transactions \( T_1, T_2, \ldots, T_n \) such that \( CS = T_1, \ldots, T_n, T_1 \) is a conflict sequence. Since length of \( CS \) is greater than 2, by definition of conflict sequence, there exists a \( T_i \) in \( CS \), \( 1 \leq i < n \), such that \( \neg \text{pair}(T_i, T_{i+1}) \). Hence by Lemma 4, \( T_1 \sim M T_1 \). Thus, there exists a transaction \( T_i \in \tau_i \) such that \( T_i \sim T_i \) which is a contradiction to our assumption. Hence, \( S_j \) is M-serializable. □

**D. PROOF OF THEOREM 4**

Let us assume that global serializability is not ensured. Thus, schedule \( S \) is not serializable. Since each of the local schedules is M-serializable, there must exist global transactions, \( T_1, T_2, \ldots, T_r \), \( r > 1 \), such that \( T_1 \) is serialized before \( T_2 \) at site \( s \), \( T_2 \) is serialized before \( T_3 \) at site \( s \), \ldots, \( T_r \) is serialized before \( T_1 \) at site \( s \). If \( T_j \) is serialized before \( T_k \) at site \( s \), then \( T_j \sim G T_k \). As a result, \( T_1 \sim G T_2 \sim G \cdots \sim G T_r \sim G T_1 \), a contradiction, since \( \sim G \) is a total order. Thus, \( S \) is serializable. □

**E. PROOF OF THEOREM 5**

Since by \( (2) \) \( \hat{S} \) is serializable, there exists a total order \( \sim_G \) on all the transactions \( \hat{T}_i \) such that for all sites \( s_k \), for all global transactions \( T_i, T_j \), if \( \text{ser}_{s_k}(T_i) \sim s_k \text{ser}_{s_k}(T_j) \), then \( \hat{T}_i \sim_G \hat{T}_j \) (since \( \text{ser}_{s_k}(T_i) \) and \( \text{ser}_{s_k}(T_j) \) are assumed to conflict). By \( (3) \) above, we have that for all sites \( s_k \), for all global transactions \( T_i, T_j \), if \( \hat{T}_{ik} \) is serialized before \( \hat{T}_{jk} \), then \( \text{ser}_{s_k}(T_{ik}) \sim s_k \text{ser}_{s_k}(T_{jk}) \). Hence, for all sites \( s_k \), for all global transactions \( T_i, T_j \), if \( \hat{T}_{ik} \) is serialized before \( \hat{T}_{jk} \), then \( \hat{T}_i \sim_G \hat{T}_j \). Thus, there exists a total order, \( \sim_G \) defined as follows: \( T_i \sim_G T_j \) if \( \hat{T}_i \sim_G \hat{T}_j \) such that for each site \( s_k \), for all global transactions \( T_i, T_j \) if \( \hat{T}_{ik} \sim \hat{T}_{jk} \) then \( T_i \sim_G T_j \). Thus, by
Theorem 4, since by (1) above, for each site $s_k$, $S_k$ is M-serializable, $\overline{S}$ is serializable. Hence proved. □

F. PROOF OF THEOREMS 6, 7, AND 8

sharad

The proofs of Theorems 6, 7 and 8 are complex. To prove the theorems we need to show that under the stated hypothesis, the M-conflict relation $\sim_M$ is acyclic. This turns out to be a difficult task. To simplify the proofs we first define another notion of conflict between transactions which is stronger than the M-conflict relation defined earlier in the paper. We refer to the conflict relation defined below as the M-conflict relation as well since, as will be shown, for the purpose of proving schedules are M-serializable both the notions of M-conflicts are equivalent—that is, the M-conflict relation defined earlier is acyclic if and only if the conflict relation defined below is acyclic.

Our strategy to proving Theorems 6, 7 and 8 will be to show that the conflict relation defined below is acyclic. This will imply that the M-conflict relation defined earlier is acyclic which by Theorem 3 implies that schedules are M-serializable.

F.1 M-Conflict Relation

Consider a partition of the relation pair into two disjoint relations—left-pair ($lpair$), and right-pair ($rpair$) such that $lpair \cup rpair = pair$. The new notion of M-conflicts between transactions is defined as follows:

**Definition 8:** Let $S_j$ be a globally complete schedule at site $s_j$ and $LV(S_j)$ be the schedule $S_j^{\tau_i}$, where $\tau_i = Lc \cup Gec \cup Ec \cup Rea$. Let $T_1, T_2$ be two distinct transactions in $LV(S_j)$. For a given partitioning $P$ of pair into $lpair$ and $rpair$, transaction $T_1$ is said to M-conflict with $T_2$ in $LV(S_j)$ (denoted by $T_1 \sim^M_P T_2$)\(^{17}\) if either of the following holds:

1. $T_1 \sim T_2$ and $\neg rpair(T_1,T_2)$.
2. $T_1 \sim T_2$ and $rpair(T_3,T_2)$.
3. $lpair(T_1,T_3)$ and $T_3 \sim T_2$.
4. $lpair(T_1,T_3)$ and $T_3 \sim T_4$ and $rpair(T_4,T_2)$. □

Note that various distinct partitionings of pair into $lpair$ and $rpair$ may exist. Different choices of $lpair$ and $rpair$ will result in different conflict relations. Next we show that for any partitioning of pair into $lpair$ and $rpair$, the relation $\sim^P_P$ is acyclic if and only if the M-conflict relation defined earlier (that is, $\sim_M$) is acyclic, where by $\sim^P_P$ we denote the transitive closure of the $\sim_P$ relation. Note that since $\sim^P_P$ relation is a subset of $\sim_M$ relation (that is, if $T_i \sim^P_P T_j$, then $T_i \sim_M T_j$), it trivially follows that if the conflict relation $\sim_M$ is acyclic, then the conflict relation $\sim^P_P$ is also acyclic. So to prove equivalence of the two conflict relations, we only need to show that if $\sim^P_P$ is acyclic, then $\sim_M$ is acyclic. We will need the following lemmas.

**Lemma 5:** Let $P$ be a partitioning of pair into $lpair$ and $rpair$. If $T_1 \sim^P_P T_2$ in $LV(S_j)$ and $rpair(T_2,T_3)$, then $T_1 \sim^M_P T_3$ in $LV(S_j)$.

\(^{17}\) Notice that the new M-conflict relation being defined is denoted by $\sim^M_P$ in contrast to the earlier defined conflict relation that is denoted by $\sim_M$. 
Proof: The proof is by induction on the length of the derivation of $T_1 \sim_P T_2$.

**Basis** ($l = 1$): We need to consider the following cases corresponding to the way in which $T_1 \sim_P T_2$ could be derived.

- **T₁ ~ T₂**: Since $rpair(T_2, T_3)$, by definition of $\sim_P$, $T_1 \sim_P T_2$.
- **T₁ ~ T₃ and rpair(T₂,T₃)**: Since $T_1 \sim T_3$, by definition of $\sim_P$, $T_1 \sim_P T_3$.
- **lpair(T₁,T₄) and T₄ ~ T₂**: Since $rpair(T_2, T_3)$ by definition of $\sim_P$, $T_1 \sim_P T_2$.
- **lpair(T₁,T₄), T₄ ~ T₃ and rpair(T₃,T₂)**: Since $lpair(T_1, T_4)$ and $T_4 \sim T_3$ by definition of $\sim_P$, $T_1 \sim_P T_3$.

**Induction**: Assume the lemma holds for all derivations of length $l$ or less. Consider a derivation of length $l + 1$. Hence, there exists a $T_r$ such that $T_1 \sim T_r$ and $T_r \sim T_2$, where the length of derivation of $T_1 \sim T_r$ and $T_r \sim T_2$ is each less than or equal to $l$. Thus, by IH, $T_r \sim_P T_3$, where $rpair(T_2, T_3)$. Hence, by transitivity $T_1 \sim_P T_3$. Hence proved. □

**Lemma 6**: Let $P$ be a partitioning of $pair$ into $lpair$ and $rpair$. If $T_1 \sim_P T_2$ in $LV(S_j)$ and $lpair(T_3, T_1)$, then $T_3 \sim_P T_2$ in $LV(S_j)$.

**Proof**: Similar to the proof of Lemma 5. □

**Lemma 7**: Let $P$ be a partitioning of $pair$ into $lpair$ and $rpair$. If $T_1 \sim T_2$ in $LV(S_j)$, then there exist transactions $T_3$ and $T_4$ such that $T_3 \sim_P T_4$ in $LV(S_j)$, where $T_3 \in GVTtoLV(LVtoGV(T_1))$ and $T_4 \in GVTtoLV(LVtoGV(T_2))$.

**Proof**: The proof is by induction on the length of derivation $l$ of the $M$-conflict relation $\sim$.

**Basis** ($l = 1$): We need to consider the following cases corresponding to the way in which $T_1 \sim_P T_2$ could be derived.

- **T₁ ~ T₂**: By definition of $\sim_P$, $T_1 \sim_P T_2$, and further since $T_1 \in GVTtoLV(LVtoGV(T_1))$ and $T_2 \in GVTtoLV(LVtoGV(T_2))$, the basis holds.
- **pair(T₅,T₁) and T₅ ~ T₂**: By definition of $\sim_P$, $T_5 \sim_P T_2$, and further since $T_5 \in GVTtoLV(LVtoGV(T_1))$ and $T_2 \in GVTtoLV(LVtoGV(T_2))$, the basis holds.
- **T₁ ~ T₅ and pair(T₂,T₅)**: By definition of $\sim_P$, $T_1 \sim_P T_5$, and further since $T_1 \in GVTtoLV(LVtoGV(T_1))$ and $T_5 \in GVTtoLV(LVtoGV(T_2))$, the basis holds.
- **pair(T₁,T₃), T₅ ~ T₆ and pair(T₆,T₂)**: By definition of $\sim_P$, $T_5 \sim_P T_6$, and further since $T_5 \in GVTtoLV(LVtoGV(T_1))$ and $T_6 \in GVTtoLV(LVtoGV(T_2))$, the basis holds.

**Induction**: Assume the lemma holds for all derivations of length $l$ or less. We show that it also holds for derivations of length $l + 1$. Let the length of the derivation of $T_1 \sim T_2$ be $l + 1$. Thus, there exists a $T_r$ such that $T_1 \sim T_r$ and $T_r \sim T_2$ each or length $l$ or less. Hence, by IH there exist transactions $T_3, T_4, T_5$ and $T_6$ such that $T_3 \sim_P T_4$ and $T_6 \sim_P T_4$ where $T_3 \in GVTtoLV(LVtoGV(T_1))$, $T_4 \in GVTtoLV(LVtoGV(T_1))$, and $T_5, T_6 \in GVTtoLV(LVtoGV(T_2))$. If $T_5$ and $T_6$ are the same transaction then by transitivity, we have, $T_3 \sim_P T_4$. Else, since $pair(T_5, T_6)$, either $rpair(T_5, T_5)$ or $lpair(T_5, T_6)$. If $rpair(T_5, T_5)$, then by Lemma 5, $T_3 \sim_P T_5$. Since $T_5 \sim_P T_4$, by transitivity we get $T_3 \sim_P T_4$. Else, if $lpair(T_5, T_6)$, then by Lemma 6, $T_5 \sim_P T_6$. Since $T_3 \sim_P T_5$, by transitivity we get $T_3 \sim_P T_4$. Hence proved. □
Lemma 8: Let $P$ be a partitioning of pair into $lpair$ and $rpair$. If $T_1 \sim P T_2$, then there exists a transaction $T_3$ such that $T_2 \sim P T_3$.

Proof: Since $T_1 \sim P T_1$, by Lemma 7, $T_3 \sim P T_4$, where $T_3, T_4 \in GVtoLV(LVtoGV(T_1))$. If $rpair(T_3, T_4)$, by Lemma 5, $T_3 \sim P T_3$. Else, if $lpair(T_3, T_4)$, then by Lemma 6, $T_4 \sim P T_4$. Hence proved. □

Lemma 8 states that if the $M$-conflict relation $\sim$ contains a cycle, then $\sim P$ contains a cycle for any partitioning $P$ of pair into $lpair$ and $rpair$. In other words, if for any partitioning $P$ into $lpair$ and $rpair$, for all transaction $T_1 \in LV(S_j)$, if $T_1 \sim P T_i$, then $T_i \sim P T_1$. We can therefore restate Theorem 3 using the above described stronger definition of $M$-conflicts as follows:

Theorem 3': Let $S_j$ be a globally complete schedule at site $s_j$. Let $P$ be any arbitrary partitioning of pair into $lpair$ and $rpair$. $S_j$ is $M$-serializable if for all $T_i$ in $LV(S_j)$, $T_i \not\sim P T_1$. □

To prove that $S_j$ is $M$-serializable, by Theorem 3', our task is to show that for some partitioning $P$ of the pair relation into $lpair$ and $rpair$, for all transactions $T_i \in \tau_j, T_i \not\sim P T_i$. The partitioning $P$ we will choose to show the above under the hypothesis of Theorems 6, 7, and 8 is as follows:

---$(R_i, W_i) \in rpair$, if $R_i$ reads local data items (and thus $W_i$ writes only exclusive data items).

---$(R_i, W_i) \in lpair$, if $R_i$ does not read any local data items.

F.2 Proof of Theorem 6

To prove Theorem 6, we first establish that under the hypothesis of the theorem, for any pair of transactions $T_i, T_j$ in $LV(S_j)$, if $T_i \not\sim P T_j$, then transaction $T_i$ commits/aborts before $T_j$ commits/aborts. This is stated in the following lemma.

Lemma 9: Let $S_j$ be a semi-rigorous schedule consisting of a single global transaction. If $T_1 \sim P T_2$ in $LV(S_j)$, then decision($T_1$) $\sim_{LV(S_j)}$ decision($T_2$). □

As a result, if there exists an $M$-conflict $T_i \not\sim P T_i$, then transaction $T_i$ commits/aborts before $T_i$ commits/aborts, which is a contradiction. Hence, the schedule $S_j$ is $M$-serializable. Thus, to prove Theorem 6, our task reduces to proving Lemma 9. The proof requires the following lemma which states that if the local schedule $S_j$ is semi-rigorous, then the order in which transactions commit/abort in $LV(S_j)$ is analogous to the order in which the transactions are serialized in $LV(S_j)$.

Lemma 10: Let the local schedule $S_j$ at site $s_j$ be semi-rigorous. If $T_1 \sim T_2$ in $LV(S_j)$, then decision($T_1$) $\sim_{LV(S_j)}$ decision($T_2$).

Proof: Since $T_1 \sim T_2$ in $LV(S_j)$, there exist conflicting operations $o_1, o_2$ such that $o_1 \in T_1, o_2 \in T_2$, and $o_1 \not\sim_{LV(S_j)} o_2$. Thus, $o_1 \not\sim_{S_j} o_2$. By the definition of $\sim$ there are following cases to consider.

---$(o_1 = r_1(x)$ and $o_2 = w_2(x))$: In this case, decision($T_2$) = $c_2$. Since $S_j$ is ROW, decision($T_1$) $\sim_{S_j}$ decision($T_2$).

Hence decision($T_1$) $\sim_{LV(S_j)}$ decision($T_2$).

---$(o_1 = w_1(x)$ and $o_2 = w_2(x))$: In this case, decision($T_1$) = $c_1$ and decision($T_2$) = $c_2$. Since $S_j$ is WOW, decision($T_1$) $\sim_{S_j}$ decision($T_2$). decision($T_1$) $\sim_{LV(S_j)}$ decision($T_2$).
\[ (o_1 = w_1(x) \text{ and } o_2 = r_2(x)) \]: Since \( S_j \) is AWOR, \( \text{decision}(T_1) \prec_{S_j} o_2 \). However, \( o_2 \prec_{S_j} \text{decision}(T_2) \). Thus, \( \text{decision}(T_1) \prec_{S_j} \text{decision}(T_2) \). Hence, \( \text{decision}(T_1) \prec_{\text{LV}(S_j)} \text{decision}(T_2) \). \( \square \)

**Proof of Lemma 9**: We prove the lemma using induction on the length, \( l \), of the derivation of \( T_1 \sim_P T_2 \).

**Basis**: \( (l = 1) \): By the definition of \( \sim_P \), and since \( S_j \) contains only a single global transaction, there are following 3 cases to consider.

1. \((T_1 \sim T_2)\): By Lemma 10, since \( S_j \) is semi-rigorous, \( \text{decision}(T_1) \prec_{\text{LV}(S_j)} \text{decision}(T_2) \).

2. \((lpair(T_1, T_3) \text{ and } T_3 \sim T_2)\): We first show that it must be the case that \( T_3 \in \text{Wc} \). If \( T_3 \in \text{Rca} \), then since \( lpair(T_1, T_3), x \notin LD(s_j) \). Hence \( T_2 \in \text{Wc} \cup \text{Gcc} \). But such a \( T_2 \) cannot exist since \( S_j \) contains only a single global transaction. Thus, \( T_3 \in \text{Wc} \). Since \( T_3 \in \text{Wc}, \text{decision}(T_1) \prec_{\text{LV}(S_j)} \text{decision}(T_3) \). Furthermore, by Lemma 10, since \( S_j \) is semi-rigorous, \( \text{decision}(T_3) \prec_{\text{LV}(S_j)} \text{decision}(T_3) \). Hence, \( \text{decision}(T_1) \prec_{\text{LV}(S_j)} \text{decision}(T_2) \).

3. \((T_1 \sim T_3, \text{ and } rpair(T_3, T_2))\): We first show that it must be the case that \( T_3 \in \text{Rca} \). If \( T_3 \in \text{Wc} \), then since \( rpair(T_3, T_2), x \in ED(s_j) \). Thus, \( T_1 \in \text{Gcc} \cup \text{Wc} \cup \text{Rca} \). But such a \( T_1 \) cannot exist since \( S_j \) contains only a single global transaction. Hence, \( T_3 \in \text{Rca} \). Since \( T_3 \in \text{Rca}, \text{decision}(T_3) \prec_{\text{LV}(S_j)} \text{decision}(T_2) \). Furthermore, by Lemma 10, since \( S_j \) is semi-rigorous, \( \text{decision}(T_1) \prec_{\text{LV}(S_j)} \text{decision}(T_3) \).

Hence, \( \text{decision}(T_1) \prec_{\text{LV}(S_j)} \text{decision}(T_2) \).

**Induction**: Let the lemma hold for all \( l < L \). We show that it holds for \( l = L + 1 \). Let \( T_1 \prec_P T_2 \) be any arbitrary M-conflict of length \( L + 1 \). Thus, there exist M-conflicts \( T_1 \prec_P T_3 \) of length \( l_1 \) and \( T_3 \prec_P T_2 \) of length \( l_2 \), for some \( l_1, l_2 \leq L \). Thus, by IH, \( \text{decision}(T_1) \prec_{\text{LV}(S_j)} \text{decision}(T_3) \) and \( \text{decision}(T_3) \prec_{\text{LV}(S_j)} \text{decision}(T_2) \). Hence, \( \text{decision}(T_1) \prec_{\text{LV}(S_j)} \text{decision}(T_2) \). \( \square \)

**Proof of Theorem 6**: By Lemma 9, for all \( T_i \in \text{LV}(S_j), T_i \not\sim_P T_i \) (since if there existed a \( T_i \) in \( \text{LV}(S_j) \) such that \( T_i \sim_P T_i \), then by Lemma 9 transaction \( T_i \) commits/aborts before \( T_i \) commits/aborts, which is a contradiction). Hence, by Theorem 3\', \( S_j \) is M-serializable. \( \square \)

**F.3 Notation**

Before we prove Theorem 7 and Theorem 8, we develop some notation that is used in the proofs. We define a schedule \( SS_j \) which is a projection of the local schedule \( S_j \) onto certain operations belonging to global subtransactions. To prove the theorems, we will show that under appropriate restrictions on the schedule \( SS_j \), M-serializability of \( S_j \) is ensured. Further, we will show that the restrictions on \( GS_j \) stated in the hypothesis of the theorems ensure the required properties of the schedule \( SS_j \).

To define \( SS_j \) at site \( s_j \), we first need to define a transaction \( \text{redo}(T_i) \) for each transaction \( T_i \in \text{Gca} \). Note that to redo the writes performed by an aborted transaction, multiple redo transactions may execute (in case the redo transaction containing the writes done by the aborted subtransaction is aborted by the local DBMS). Let \( W_{i_1}, W_{i_2}, \ldots, W_{i_s} \) be the transactions that are used to redo a global transaction \( T_i \), \( T_i \in \text{Gca} \), such that \( W_{i_1}, W_{i_2}, \ldots, W_{i_{(s-1)}} \) abort whereas \( W_{i_s} \) commits. Thus, \( W_{i_s} = W_i \).

\[
\text{redo}(T_i) = \text{operations}(W_{i_1}) \circ \text{operations}(W_{i_2}) \circ \ldots \circ \text{operations}(W_{i_{(s-1)})} \circ W_{i_s}
\]
\( SS_j \) is the projection of \( S_j \) over transactions in \( \tau_j \), that is \( SS_j = S_j^{\tau_j} \), where

\[
\tau_j = \text{Gce} \cup \text{Gca} \cup \{ \text{operations}(T_i) \circ \text{redo}(T_i) \mid T_i \in \text{Gcaw} \}
\]

We next relate the properties satisfied by the schedules \( GS_j \) and the schedule \( SS_j \). To do so, we define functions \( \text{StoSS} \) and \( \text{GstoSS} \) that map transactions in schedule \( S_j \) and \( GS_j \) to corresponding transactions in schedule \( SS_j \) respectively.

\[
\text{StoSS} : \text{Gce} \cup \text{Gca} \rightarrow \tau_j,
\text{StoSS}(T) = \begin{cases} \text{operations}(T) \circ \text{redo}(T) & \text{if } T \in \text{Gcaw} \\ T & \text{otherwise} \end{cases}
\]

\[
\text{GstoSS} : \tau_j \rightarrow \tau_j \text{ such that } \text{GstoSS}(T) = \text{StoSS}(\text{GstoSS}(T)). \text{ Thus,}
\text{GstoSS}(T) = \begin{cases} \text{operations}(T_k) \circ \text{redo}(T_k) & \text{if } \text{GstoSS}(T) \in \text{Gcaw} = T_k \\ T & \text{otherwise} \end{cases}
\]

**Lemma 11:** If \( GS_j \) is AROW + AWOR, then \( SS_j \) is AROW.

**Proof:** If \( SS_j \) is not AROW, then there exist \( o_1, o_2, \) and \( o_3 \), where \( o_1 = r_1(x) \), \( o_2 = w_2(x) \), and \( o_3 = c_1 \), and \( o_1, o_2 \in \text{GstoSS}(T_1) \) and \( o_2 \in \text{GstoSS}(T_2) \), such that \( o_1 \prec_{SS_j} o_2 \prec_{SS_j} o_3 \). Either \( o_2 \in \text{operations}(T_2) \) or \( o_2 \notin \text{redo}(T_2) \). If \( o_2 \in \text{redo}(T_2) \), then there exists an operation \( o_4 = w_3(x) \in \text{operations}(T_2) \) in \( SS_j \). Consider the following three cases.

\(- (o_2 \in \text{operations}(T_2)): As o_2 \in \text{operations}(T_2), o_1 \prec_{GS_j} o_2 \prec_{GS_j} o_3 \) which is a contradiction since \( GS_j \) is AROW.

\(- (o_2 \notin \text{operations}(T_2), \text{and } o_1 \prec_{SS_j} o_4): As o_4 \in \text{operations}(T_2), o_1 \prec_{GS_j} o_4 \prec_{GS_j} o_3 \) which is a contradiction since \( GS_j \) is AROW.

\(- (o_2 \notin \text{operations}(T_2), \text{and } o_3 \prec_{SS_j} o_1): As o_4 \in \text{operations}(T_2), o_4 \prec_{GS_j} o_1 \prec_{GS_j} o_3 \) which is a contradiction since \( GS_j \) is AWOR.

Hence, \( SS_j \) is AROW. \( \Box \)

**Lemma 12:** If \( GS_j \) is AWOW, then \( SS_j \) is AWOW.

**Proof:** Since \( GS_j \) is AWOW, for every pair of operations, \( o_1 \) and \( o_2 \), such that \( o_1 = w_1(x) \), \( o_2 = w_2(x) \), \( o_1 \in T_1 \) and \( o_2 \in T_2 \), \( o_1 \prec_{GS_j} o_3 \prec_{GS_j} o_2 \), where \( o_3 = c_1 \). Thus, \( o_1 \prec_{SS_j} o_3 \prec_{SS_j} o_2 \), \( o_1 \in \text{operations}(T_1) \) and \( o_2 \in \text{operations}(T_2) \). Further, if there exists an \( o_4 = w_3(x) \in \text{redo}(T_1) \), then \( o_4 \prec_{SS_j} o_3 \) and also for any \( o_5 = w_2(x) \) such that \( o_5 \in \text{redo}(T_1) \), \( o_2 \prec_{SS_j} o_5 \), we have that \( SS_j \) is AWOW. \( \Box \)

**Lemma 13:** If \( GS_j \) is AWOR, then \( SS_j \) is AWOR.

**Proof:** If \( SS_j \) is not AWOR, then there exist \( o_1, o_2, o_3 \in SS_j \), where \( o_1 = w_1(x) \), \( o_2 = r_2(x) \), \( o_3 = c_1 \), \( o_2 \in GstoSS(T_1) \), and \( o_1, o_3 \in GstoSS(T_1) \) such that \( o_1 \prec_{SS_j} o_2 \prec_{SS_j} o_3 \). If \( o_1 \in \text{operations}(T_1) \), then \( o_1 \prec_{GS_j} o_2 \prec_{GS_j} o_3 \), which is a contradiction as \( GS_j \) is AWOR. Else, if \( o_1 \notin \text{operations}(T_1) \), then \( o_1 \in \text{redo}(T_1) \). Thus, there exists an operation \( o_4 = w_1(x) \in T_1 \), and \( o_4 \prec_{SS_j} o_1 \). Hence, \( o_4 \prec_{SS_j} o_1 \prec_{SS_j} o_2 \prec_{SS_j} o_3 \). Thus, \( o_4 \prec_{GS_j} o_2 \prec_{GS_j} o_3 \) which is a contradiction since \( GS_j \) is AWOR. Thus, \( SS_j \) is AWOR. \( \Box \)
Lemma 14: If $G_{S_j}$ is ROW + AWOR, then $SS_j$ is ROW.

Proof: If $SS_j$ is not ROW, then there exist $o_1$, $o_2$, $o_3$ and $o_4$, where $o_1 = r_1(x)$, $o_2 = w_2(x)$, $o_3 = c_2$, and $o_4 = c_1$, and $o_1, o_4 \in G_{S_j}SS(T_1)$ and $o_2, o_3 \in G_{S_j}SS(T_2)$, such that $o_1 \prec_{SS_j} o_2 \prec_{SS_j} o_3 \prec_{SS_j} o_4$. Either $o_2 \in \text{operations}(T_2)$ or $o_2 \in \text{redo}(T_2)$. If $o_2 \in \text{redo}(T_2)$, then there exists an operation $o_5 = v_2(x) \in \text{operations}(T_2)$ in $SS_j$. Consider the following three cases.

$- (o_2 \in \text{operations}(T_2))$: As $o_2 \in \text{operations}(T_2)$, $o_1 \prec_{G_{S_j}} o_2 \prec_{G_{S_j}} o_3 \prec_{G_{S_j}} o_4$ which is a contradiction since $G_{S_j}$ is ROW.

$- (o_2 \notin \text{operations}(T_2)$, and $o_1 \prec_{SS_j} o_5$): As $o_5 \in \text{operations}(T_2)$, $o_1 \prec_{G_{S_j}} o_5 \prec_{G_{S_j}} o_3 \prec_{G_{S_j}} o_4$ which is a contradiction since $G_{S_j}$ is ROW.

$- (o_2 \notin \text{operations}(T_2)$, and $o_5 \prec_{SS_j} o_1$): As $o_5 \in \text{operations}(T_2)$, $o_5 \prec_{G_{S_j}} o_1 \prec_{G_{S_j}} o_3 \prec_{G_{S_j}} o_4$ which is a contradiction since $G_{S_j}$ is AWOR.

Hence, $SS_j$ is ROW. □

F.4 Proof of Theorem 7

By Lemmas 11, 12, and 13, if $G_{S_j}$ is rigorous, then the schedule $SS_j$ is also rigorous. Hence, to prove Theorem 7, it suffices to show that if $SS_j$ is rigorous, and $S_j$ is semi-rigorous, then for all transactions $T_i$ in $LV(S_j)$, $T_i \prec_{M} T_j$.

To do so, we first establish that under the hypothesis, for any pair of transactions $T_i, T_j$ in $LV(S_j)$, if $T_i \prec_{M} T_j$, then transaction $T_i$ commits/aborts before $T_j$ commits/aborts. This is stated in the following lemma.

Lemma 15: Let $SS_j$ be rigorous, and $S_j$ be semi-rigorous. If $T_1 \prec_{M} T_2$ in $LV(S_j)$, then $\text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_2)$. □

As a result, if there exists an M-conflict $T_i \prec_{M} T_j$, then transaction $T_i$ commits/aborts before $T_i$ commits/aborts, which is a contradiction. Hence, the schedule $S_j$ is M-serializable. Thus, to prove Theorem 7, our task reduces to proving Lemma 15. To prove Lemma 15, we will use induction over the length of the derivation of $T_1 \prec_{M} T_2$. Recall that the conflict relation $\prec_{M}$ is the transitive closure of the conflict relation $\prec$. Thus, in the base case (that is, the length of derivation is one) we need to show that if $T_1 \prec_{M} T_2$, then $T_1$ commits/aborts before $T_2$. By the definition of $\prec_{M}$, the conflict $T_1 \prec_{M} T_2$ could arise due to either one of the following cases:

1. $T_1 \sim T_2$ and $\neg \text{pair}(T_1, T_3)$.
2. $T_1 \sim T_3$ and $\text{pair}(T_3, T_2)$.
3. $\text{lpair}(T_1, T_3)$ and $T_3 \sim T_2$.
4. $\text{lpair}(T_1, T_3)$ and $T_3 \sim T_4$ and $\text{pair}(T_4, T_2)$.

If the M-conflict $T_1 \prec_{M} T_2$ arises due to a conflict $T_1 \sim T_2$, then by Lemma 10, the basis trivially holds. In the following three lemmas we show that the basis will hold if $T_1 \prec_{M} T_2$ arises due to the remainder of the three cases.

Lemma 16: Let $SS_j$ be ROW and $S_j$ be semi-rigorous. If $\text{lpair}(T_1, T_3)$, and $T_3 \sim T_2$ in $LV(S_j)$, then $\text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_2)$.

Proof: By Lemma 10, since $S_j$ is semi-rigorous, $\text{decision}(T_3) \prec_{LV(S_j)} \text{decision}(T_2)$. There are two cases:
(T_3 \in W_c): Since decision(T_1) \prec_{LV(S_j)} decision(T_3), decision(T_1) \prec_{LV(S_j)} decision(T_2).

(T_3 \in R_{ea}): Since T_3 \sim T_2 in LV(S_j), there exist operations \(o_1 \in T_3, o_2 \in T_2\) such that \(o_1 = r_2(x), o_2 = w_2(x)\) and \(o_1 \prec_{LV(S_j)} o_2\). Since \(\text{lpair}(T_1, T_3), x \notin LD(S_j)\). Thus, \(T_2 \in W_c \cup \text{Gcc}\). Hence, operations \(o_1, o_2\) appear in \(SS_j\) and further \(o_1 \prec_{SS_j} o_2\). Let \(T_k = \text{LVtoS}(T_3)\), and \(T_k = \text{LVtoS}(T_2)\). Since \(SS_j\) is \(\text{ROW}\), and \(o_1 \prec_{SS_j} o_2\), decision(T_1) = decision(StoSS(T_1)) \prec_{SS_j} decision(StoSS(T_k)) = decision(T_2).

Thus, decision(T_1) \prec_{LV(S_j)} decision(T_2). □

Lemma 17: Let \(SS_j\) be rigorous and \(S_j\) be semi-rigorous. If \(T_1 \sim T_3 in LV(S_j)\), and \(\text{rpair}(T_3, T_2)\), then \(\text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_3)\).

Proof: Since \(T_1 \sim T_3\) and \(S_j\) is semi-rigorous, by Lemma 10, \(\text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_3)\). There are two cases:

(T_3 \in W_c): Since decision(T_3) \prec_{LV(S_j)} decision(T_2), decision(T_1) \prec_{LV(S_j)} decision(T_2).

(T_3 \in R_{ea}): Since \(T_1 \sim T_3\), there exist operations \(o_1 \in T_1, o_2 \in T_3\) such that \(o_1 \prec_{LV(S_j)} o_2\), where \(o_1 = w_3(x)\). Let \(T_k = \text{LVtoS}(T_3)\). Thus, there exists an operation \(o_3 = w_k(x) \in T_k\) such that \(o_3 \prec_{S_j} o_2\) and \(o_3 \prec_{SS_j} \text{decision}(T_k) = \text{decision}(T_2)\). Since \(\text{rpair}(T_3, T_2), x \in ED(S_j)\). Thus, \(T_1 \in \text{Gcc} \cup W_c \cup R_{ea}\). Hence, \(o_1 \prec_{SS_j} o_2\). We first show that it is not the case that \(o_3 \prec_{S_j} o_1\).

Suppose that \(o_3 \prec_{S_j} o_1\). Since \(o_3 \prec_{S_j} o_1, o_3 \prec_{SS_j} o_1\). Since \(SS_j\) is \(\text{AWOR} + \text{AWOW}\), \(\text{decision}(\text{StoSS}(T_k)) \prec_{SS_j} o_1\). However, since \(o_2 \prec_{SS_j} \text{decision}(T_3)\) and \(\text{decision}(\text{StoSS}(T_k)) = \text{decision}(T_3)\), we have that \(o_2 \prec_{SS_j} o_1\) which is a contradiction. Hence, \(o_1 \prec_{S_j} o_3\).

Since \(o_1 \prec_{S_j} o_3\) and both \(o_1, o_3\) appear in \(SS_j, o_1 \prec_{SS_j} o_3\). Let \(T_i = \text{LVtoS}(T_1)\). Since \(SS_j\) is \(\text{AROW} + \text{AWOW}\), \(\text{decision}(\text{StoSS}(T_i)) \prec_{SS_j} o_3\). If \(T_1 \in W_c \cup \text{Gcc} \cup \text{Gear}\), then since \(\text{decision}(T_1) = \text{decision}(T_i) = \text{decision}(\text{StoSS}(T_i))\). Thus, \(\text{decision}(T_1) \prec_{S_j} o_3 \prec_{S_j} \text{decision}(T_k) = \text{decision}(T_2)\). Hence, \(\text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_2)\). Else, \((T_1 \in \text{R}_{ea}\) and \(T_1 \notin \text{Gear}\), \(\text{decision}(T_1) = \text{decision}(T_i) \prec_{S_j} \text{decision}(\text{StoSS}(T_i))\).

Since \(\text{decision}(\text{StoSS}(T_i)) \prec_{S_j} o_3 \prec_{S_j} \text{decision}(T_2), \text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_2)\). Hence proved. □

Lemma 18: Let \(SS_j\) be rigorous and \(S_j\) be semi-rigorous. If \(\text{pair}(T_1, T_3), T_3 \sim T_4 in LV(S_j)\), and \(\text{pair}(T_4, T_2)\), then \(\text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_4)\).

Proof: By Lemma 10, since \(S_j\) is semi-rigorous, \(\text{decision}(T_3) \prec_{LV(S_j)} \text{decision}(T_4)\). There are three cases to consider.

(T_3 \in W_c and T_4 \in R_{ea}): In this case since \(\text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_3), \text{decision}(T_4) \prec_{LV(S_j)} \text{decision}(T_2)\), we have that \(\text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_2)\).

(T_3 \in W_c and T_4 \in W_c): In this case, there exist operations \(o_1 \in T_3, o_2 \in T_4\) such that \(o_1 \prec_{LV(S_j)} o_2\), where \(o_1 = w_3(x)\) and \(o_2 = w_4(x)\). Let \(T_k = \text{LVtoS}(T_4)\). Thus, there exists an operation \(o_3 = w_k(x)\) such that \(o_3 \prec_{S_j} o_2\) and further \(o_3 \prec_{S_j} \text{decision}(T_k) = \text{decision}(T_2)\). We first show that \(o_1 \prec_{S_j} o_3\).

Suppose \(o_3 \prec_{S_j} o_1\). In this case, \(o_3 \prec_{SS_j} o_1\). However, since \(SS_j\) is \(\text{AWOW}\), \(\text{decision}(\text{StoSS}(T_k)) \prec_{SS_j} o_1\). Since \(o_2 \prec_{SS_j} \text{decision}(T_4) = \text{decision}(\text{StoSS}(T_k)), o_2 \prec_{SS_j} o_1\), which is a contradiction. Thus, \(o_1 \prec_{S_j} o_3\).
Since \( o_1 \prec_{S_j} o_3, o_1 \prec_{S_j} o_3 \). Let \( T_i = LV_{toS}(T_3) \). Since \( SS_j \) is AWOW, \( \text{decision}(T_3) = \text{decision}(StoSS(T_i)) \prec_{SS_j} o_3 \). Hence, \( \text{decision}(T_3) \prec_{S_j} o_3 \prec_{S_j} \text{decision}(T_2) \). Hence, \( \text{decision}(T_3) \prec_{LV(S_j)} \text{decision}(T_2) \). Since \( \text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_3), \text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_2) \).

\(-T_3 \in \text{Rea} \) and \( T_4 \in \text{We} \): In this case, there exist operations \( o_1 \in T_3 \) and \( o_2 \in T_4 \) such that \( o_1 \prec_{LV(S_j)} o_2 \), where \( o_1 = r_3(x) \) and \( o_2 = w_4(x) \). Let \( T_k = LV_{toS}(T_4) \). Thus, there exists an operation \( o_3 = w_5(x) \) such that \( o_3 \prec_{S_j} o_2 \) and further \( o_3 \prec_{S_j} \text{decision}(T_k) = \text{decision}(T_2) \). We first show that \( o_1 \prec_{S_j} o_3 \).

Suppose \( o_3 \prec_{S_j} o_1 \). In this case, \( o_3 \prec_{SS_j} o_1 \). However, since \( SS_j \) is AWOR, \( \text{decision}(StoSS(T_k)) \prec_{SS_j} o_1 \).

Since \( o_2 \prec_{SS_j} \text{decision}(T_4) = \text{decision}(StoSS(T_k)) \), \( o_2 \prec_{SS_j} o_1 \), which is a contradiction. Thus, \( o_1 \prec_{S_j} o_3 \).

Since \( o_1 \prec_{S_j} o_3, o_1 \prec_{SS_j} o_3 \). Let \( T_i = LV_{toS}(T_3) \). Since \( SS_j \) is AROW, \( \text{decision}(T_1) = \text{decision}(StoSS(T_i)) \prec_{SS_j} o_3 \). Hence, \( \text{decision}(T_1) \prec_{S_j} o_3 \prec_{S_j} \text{decision}(T_2) \). Hence, \( \text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_2) \). Hence proved.

\( \Box \)

**Proof of Lemma 15:** We prove the lemma using induction on the length, \( l \), of the derivation of \( T_1 \prec_{P} T_2 \).

**Basis:** \( (l = 1) \): By the definition of \( \prec_{P} \), there are the following cases to consider.

1. \( (T_1 \prec T_2) \): By Lemma 10, since \( S_j \) is semi-rigorous, \( \text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_2) \).
2. \( (lpair(T_1, T_3) \) and \( T_3 \prec T_2) \): By Lemma 16, \( \text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_2) \).
3. \( (T_1 \prec T_3, \) and \( rpair(T_3, T_2)) \): By Lemma 17, \( \text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_2) \).
4. \( (lpair(T_1, T_3) \) and \( T_3 \prec T_4, \) and \( rpair(T_4, T_2)) \): By Lemma 18, \( \text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_2) \).

**Induction:** Let the lemma hold for all \( l \leq L \). We show that it holds for \( l = L + 1 \). Let \( T_1 \prec_{P} T_2 \) be an arbitrary M-conflict of length \( L + 1 \). Thus, there exist M-conflicts \( T_1 \prec_{P} T_3 \) of length \( l_1 \) and \( T_3 \prec_{P} T_2 \) of length \( l_2 \), for some \( l_1, l_2 \leq L \). Thus, by IH, \( \text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_3) \) and \( \text{decision}(T_3) \prec_{LV(S_j)} \text{decision}(T_2) \). Hence, \( \text{decision}(T_1) \prec_{LV(S_j)} \text{decision}(T_2) \). \( \Box \)

**Proof of Theorem 7:** By Lemmas 11, 12, and 13, if \( GS_j \) is rigorous, then the schedule \( SS_j \) is also rigorous. Thus, due to Lemma 15, for all \( T_i \) in \( LV(S_j) \), \( T_i \not\prec_{P} T_i \) (since if there existed a \( T_i \) in \( LV(S_j) \) such that \( T_i \prec_{P} T_i \), then by Lemma 15, transaction \( T_i \) commits/aborts before \( T_i \) commits/aborts, which is a contradiction). Hence, by 'Theorem 3', \( S_j \) is M-serializable. \( \Box \)

F.5 Proof of Theorem 8

To prove the Theorem 8, we first need to identify the nature of conflicts that occur in the schedule \( LV(S_j) \) if in case an EC protocol is followed to commit global transactions. This is done in the following lemma.

**Lemma 19:** Let the GTM follow the EC protocol. If \( T_i \sim T_k \) in \( LV(S_j) \), then

1. \( \text{If } T_i \in \text{Rea}, \text{then } T_k \in \text{Gcc} \cup \text{We} \).
2. \( \text{If } T_k \in \text{Rea}, \text{then } T_i \in \text{Gcc} \cup \text{We} \).

**Proof:** We only prove (1). The proof of (2) is similar and thus omitted. Let \( T_i \) be a transaction \( T_i \in \text{Gcc} \). Since the GTM uses an EC protocol, \( Gev = \emptyset \), and thus \( T_i \) contains a write operation. 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 exist conflicting operations \( o_1, o_2 \) such that \( o_1 = r_i(x) \), \( o_2 = w_k(x) \), and \( o_1 \prec_{LV(S_j)} o_2 \), where \( x \) is not a local data item. Thus, since \( T_k \) writes \( x \), \( T_k \) is not a local transaction. Hence, \( T_k \in Gcc \cup Wc \). \( \square \)

We next note that if the schedule \( GS_j \) is ROW + AWOR, then by Lemma 13 and Lemma 14, the schedule \( SS_j \) is ROW + AWOR. To prove the Theorem 8, we need to show that if the schedule \( SS_j \) is ROW + AWOR, \( S_j \) is strongly recoverable and the GTM follows an EC protocol, then \( S_j \) is M-serializable. To do so, for each transaction \( T_i \) in \( LV(S_j) \) (that is, \( T_i \in Gcc \cup Wc \cup Lc \cup Rea \)) we define a transaction \( \widetilde{T}_i \) as follows:

\[
\widetilde{T}_i = \begin{cases} 
T_k & \text{if } T_i \in Rea \text{ and } pair(T_i, T_k) \\
T_i & \text{otherwise}
\end{cases}
\]

We will show that under the hypothesis of the theorem, if \( T_1 \sim_{\rho} T_2 \), then transaction \( \widetilde{T}_1 \) commits/aborts before transaction \( \widetilde{T}_2 \) as is stated in the following lemma.

**Lemma 20:** Let the GTM follow the EC protocol, \( SS_j \) be ROW + WOR and \( S_j \) be strongly recoverable. If \( T_1 \sim_{\rho} T_2 \) in \( LV(S_j) \), then \( decision(\widetilde{T}_1) \prec_{LV(S_j)} decision(\widetilde{T}_2) \). \( \square \)

We will prove the above lemma using induction over the length of the derivation of the the conflict \( T_1 \sim_{\rho} T_2 \). Note that since in our partitioning \( P \) of \( pair \) into \( lpair \) and \( rpair \), the set of \( rpair = \emptyset \) and \( lpair = pair \) (due to the restrictions on global transactions, transactions that read local data items do not write any data item at the site), the conflict \( T_1 \sim_{\rho} P T_2 \) could only arise due to the following two cases:

1. \( T_1 \sim T_2 \) and \( \sim pair(T_1, T_2) \).
2. \( lpair(T_1, T_2) \) and \( T_3 \sim T_2 \).

In the following lemma, we show that under the hypothesis of Lemma 20, if \( T_1 \sim T_2 \), then \( \widetilde{T}_1 \) commits/aborts before \( \widetilde{T}_2 \).

**Lemma 21:** Let \( S_j \) be strongly recoverable and \( SS_j \) be ROW + WOR. If \( T_1 \sim T_2 \), then \( decision(\widetilde{T}_1) \prec_{LV(S_j)} decision(\widetilde{T}_2) \).

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

---

1. \( T_1, T_2 \notin Rea \): Note that \( \widetilde{T}_1 = T_1 \), \( \widetilde{T}_2 = T_2 \), \( decision(T_1) = c_1 \) and \( decision(T_2) = c_2 \). Since \( T_1 \sim T_2 \), there exist conflicting operations \( o_1, o_2 \), such that \( o_1 \in T_1 \) and \( o_2 \in T_2 \), \( o_1 \prec_{LV(S_j)} o_2 \). Since \( o_1 \prec_{LV(S_j)} o_2 \), \( o_1 \prec_{S_j} o_2 \). Depending upon \( o_1 \) and \( o_2 \), there are three cases to consider:

   - \( o_1 = r_1(x) \) and \( o_2 = w_2(x) \): Since \( S_j \) is ROW and \( decision(T_2) = c_2 \), \( decision(T_1) \prec_{S_j} decision(T_2) \).
   
   Hence, \( decision(T_1) \prec_{LV(S_j)} decision(T_2) \).

   - \( o_1 = w_1(x) \) and \( o_2 = r_2(x) \): Since \( S_j \) is WOR and \( decision(T_2) = c_2 \), \( decision(T_1) \prec_{S_j} decision(T_2) \).
   
   Hence, \( decision(T_1) \prec_{LV(S_j)} decision(T_2) \).

   - \( o_1 = w_1(x) \) and \( o_2 = w_2(x) \): Since \( S_j \) is WOW and \( decision(T_2) = c_2 \), \( decision(T_1) \prec_{S_j} decision(T_2) \).
   
   Hence, \( decision(T_1) \prec_{LV(S_j)} decision(T_2) \).

Thus, in case \( T_1, T_2 \notin Rea \), \( decision(T_1) \prec_{LV(S_j)} decision(T_2) \). Since \( \widetilde{T}_1 = T_1 \) and \( \widetilde{T}_2 = T_2 \), \( decision(\widetilde{T}_1) \prec_{LV(S_j)} decision(\widetilde{T}_2) \).
( T₁ ∉ Rea and T₂ ∈ Rea): Note that \( \tilde{T}_1 = T_1 \) and \( \text{decision}(T_1) = c_1 \). Since \( T_1 \leadsto T_2 \), there exist conflicting operations \( o_1, o_2 \), such that \( o_1 \in T_1 \) and \( o_2 \in T_2 \), \( o_1 \prec_{LV(S_j)} o_2 \). Let \( o_1 = w_1(x) \), and \( o_2 = r_2(x) \). By Lemma 19, \( T_1 \in \text{Wc} \cup \text{Gcc} \). Hence, \( o_1 \prec_{SS_j} o_2 \). Let \( T_i = LVtoS(T_1) \) and \( T_k = LVtoS(T_2) \). Since the GTM uses an EC protocol, \( T_k \in \text{Gcase} \), and thus \( \text{decision}(StoSS(T_k)) \) is the commit operation. Since \( SS_j \) is \text{ROW}, \( o_1 \prec_{SS_j} o_2 \) and \( \text{decision}(StoSS(T_k)) \) is a commit operation \( \text{decision}(StoSS(T_i)) \prec_{SS_j} \text{decision}(StoSS(T_k)) \). Since \( \text{decision}(StoSS(T_i)) = \text{decision}(\tilde{T}_1) \) and \( \text{decision}(StoSS(T_k)) = \text{decision}(T_2) \), \( \text{decision}(\tilde{T}_1) \prec_{SS_j} \text{decision}(\tilde{T}_2) \). Thus, \( \text{decision}(\tilde{T}_1) \prec_{LV(S_j)} \text{decision}(\tilde{T}_2) \). □

( T₁ ∈ Rea and T₂ ∉ Rea): Note \( \tilde{T}_2 = T_2 \) and \( \text{decision}(T_2) = c_2 \). Since \( T_1 \leadsto T_2 \), there exist conflicting operations \( o_1, o_2 \), such that \( o_1 \in T_1 \) and \( o_2 \in T_2 \), \( o_1 \prec_{LV(S_j)} o_2 \). Let \( o_1 = r_1(x) \) and \( o_2 = w_2(x) \). By Lemma 19, \( T_2 \in \text{Gcc} \cup \text{Wc} \). Let \( T_i = LVtoS(T_1) \) and \( T_k = LVtoS(T_2) \). Since \( SS_j \) is \text{ROW}, \( o_1 \prec_{SS_j} o_2 \) and \( \text{decision}(StoSS(T_k)) \) is a commit operation \( \text{decision}(StoSS(T_i)) \prec_{SS_j} \text{decision}(StoSS(T_k)) \). Since \( \text{decision}(StoSS(T_i)) = \text{decision}(\tilde{T}_1) \) and \( \text{decision}(StoSS(T_k)) = \text{decision}(\tilde{T}_2) \), \( \text{decision}(\tilde{T}_1) \prec_{SS_j} \text{decision}(\tilde{T}_2) \). Thus, \( \text{decision}(\tilde{T}_1) \prec_{LV(S_j)} \text{decision}(\tilde{T}_2) \). □

**Proof of Lemma 20:** The proof is by induction on the length \( l \) of the derivation of \( T_1 \prec_{op} T_2 \).

**Basis (l=1):** Thus, \( T_1 \overset{M}{\sim} P T_2 \). By the definition of \( \overset{M}{\sim} \) there are the following two cases to consider.

1. \( (T_1 \leadsto T_2) \): Trivial by Lemma 21.
2. \( (T_1 \leadsto T_3 \text{ and } \text{pair}(T_3, T_2)) \): By Lemma 21, \( \text{decision}(\tilde{T}_1) \prec_{LV(S_j)} \text{decision}(\tilde{T}_3) \). Since \( \text{pair}(T_2, T_3) \), \( \tilde{T}_3=\tilde{T}_2 \).

Hence, \( \text{decision}(\tilde{T}_2) = \text{decision}(\tilde{T}_3) \). Thus, \( \text{decision}(\tilde{T}_1) \prec_{LV(S_j)} \text{decision}(\tilde{T}_2) \).

**Induction:** Let the lemma hold for all \( l \leq L \). We show that it holds for \( l = L + 1 \). Let \( T_1 \prec_{op} T_2 \) be any arbitrary \( M \)-conflict of length \( L + 1 \). Thus, there exist \( M \)-conflicts \( T_1 \prec_{op} T_3 \) of length \( l_1 \) and \( T_3 \prec_{op} T_2 \) of length \( l_2 \), for some \( l_1, l_2 \leq L \). Thus, by IH, \( \text{decision}(\tilde{T}_1) \prec_{LV(S_j)} \text{decision}(\tilde{T}_3) \), and \( \text{decision}(\tilde{T}_3) \prec_{LV(S_j)} \text{decision}(\tilde{T}_2) \).

Thus, \( \text{decision}(\tilde{T}_1) \prec_{LV(S_j)} \text{decision}(\tilde{T}_2) \). Hence proved. □

**Proof of Theorem 8:** By Lemma 13 and 14, if \( GS_j \) is \text{ROW} + \text{AWOR}, \( SS_j \) is \text{ROW} + \text{AWOR} (and hence \text{ROW} + \text{WOR}). Thus, due to Lemma 20, for all \( T_i \) in \( LV(S_j) \), \( T_i \prec_{op} T_i \). Hence, by Theorem 3', \( S_j \) is \( M \)-serializable. □