1991

Global Commitment in Multidatabase Systems

Ahmed K. Elmagarmid
Purdue University, ake@cs.purdue.edu

Jin Jing

Won Kim

Report Number:
91-017

http://docs.lib.purdue.edu/cstech/866

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
GLOBAL COMMITMENT IN MULTIDATABASE SYSTEMS

Ahmed K. Elmagarmid
Jin Jing
Won Kim

CSD-TR-91-017
March 1991
Global Commitment in Multidatabase Systems

Ahmed K. Elmagarmid and Jin Jing
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Won Kim
UniSQL, Inc.
9390 Research Blvd.
Austin, TX 78759

Abstract

In this paper, we investigate the reliability aspects of transaction processing in multidatabase systems from a formal point of view. We define a new correctness notion called global commitment (GC) for reliable transaction processing in a multidatabase environment in which local DBMSs may not support the prepared state of the two phase commitment protocol. A multidatabase transaction management that produces a globally committable history can guarantee the commitment of global (multi-site) transactions without violating the local autonomy requirements, even in the presence of transaction and system failures. The GC notion, together with serializability and recoverability notions, are used as the correctness criteria for reliable transaction processing in multidatabase systems.

1 Introduction

A multidatabase system (MDBS) is an integration of pre-existing database systems (called local database systems, or LDBSs) supporting global applications accessing more than one LDBS. An important feature of MDBSs is the autonomy of LDBSs. Generally, local autonomy reflects the fact that LDBSs were independently developed and maintained by different organizations. They are later integrated in a bottom-up fashion. Existing applications (called local transactions) are expected to continue to execute after integration. New applications accessing more than one LDBS (called global or multi-site transactions) are decomposed into subtransactions which are then executed at local sites along with local transactions.

The local autonomy requirements can be characterized as design autonomy, execution autonomy, etc. Design autonomy means each of the LDBSs is free to use whatever data models and transaction management algorithms it wishes. Execution autonomy means that each of the LDBSs is free to do anything (e.g., abort the execution of transactions) on any transactions running at its local site. The local autonomy requirements have significant effects on transaction
management in an MDBS, especially true in the presence of failures. This is especially true since the MDBS is not aware of local transactions that are being executed by LDBSs and the LDBSs do not necessarily support the prepared state of the two phase commit (2PC) protocol [DEK90a].

Transaction management in an MDBS coordinates concurrent execution of global and local transactions to ensure the consistency of the MDBS. Transaction management problems in a multidatabase system, namely concurrency control, commitment and recovery, were first presented in [GPZ86], essentially, in the form of problem identification and modeling. Since then, the concurrency control problem in an MDBS environment was extensively studied with the assumption that no failures can occur during the global transaction processing, e.g. [EL87, Pu88, AGMS87, BS88, LT88, DE89, ED90].

Recently, reliable transaction management in MDBSs has started to receive more attention, e.g. [BST90, WV90]. In these proposed approaches, authors have made assumptions that LDBSs neither support the prepared state of the 2PC protocol, nor have an external interface for participating in the 2PC protocol. Furthermore, these authors have used conventional serializability (SR) and recoverability (RC) notions [BHG87] as correctness criteria for transaction processing. Conceptually, these correctness notions require that the execution of each transaction must appear to every other transaction as a single atomic step. Transactions in this framework, therefore, serve at least two distinct purposes [Lyn83]: (1) it is a logical unit that groups together operations that constitute a complete task; and (2) it is an atomicity unit in the sense that it should appear to users of the database that either all of these operations are executed consecutively, without any intervening operations of other transactions, or none are. The two phase commit (2PC) protocol is incorporated into distributed transaction processing so that the "atomicity" assumption is true for multi-site transactions. Unfortunately the local DBMSs in an MDBS may not support the prepared state of the 2PC protocol (or does not support one visible to the MDBS), so 2PC cannot be implemented directly. Attempts to simulate 2PC without violating local autonomy would seem impossible, since, in general, the MDBS cannot guarantee a subtransaction will commit unless it has actually committed it, and the MDBS cannot prevent local transactions from accessing the data items of an aborted or committed subtransaction.

The 2PC Agent Method in [WV90] simulates the 2PC protocol by delaying the execution of local transactions that may update the data items read or written by a subtransaction until the subtransaction is committed. The delay property severely violates the local autonomy requirements. Consider the situation where a subtransaction is aborted because of the failure of a site after MDBS has decided to commit the corresponding global transaction. After the site recovers, it cannot do anything before it executes the corresponding resubmitted subtransaction.
from MDBS successfully. The approaches in [LEM88],[GR89], and [BO90] also require delaying the execution of local transactions for global transaction commitment processing in multidatabase systems. The method in [BST90] implements the 2PC protocol by imposing some restrictions on both local and global transactions. As a result, local (or global) transactions are prohibited from updating the data items that are updatable by global (or local) transactions. In many practical cases this may be too strong a restriction on global transactions accessing autonomous local databases. It is evident that conventional serializability and recoverability notions are not enough for transaction processing in an MDBS environment because it is difficult to commit a multi-site transaction atomically without violating the local autonomy requirements.

In this paper, we formulate serializability and recoverability notions so that a global transaction can be defined as a set of atomic subtransactions which are executed at local sites along with local transactions. Defining serializability based on multi-atomic subtransactions is not a "new" method. In fact, authors in [GM83], [Lyn83], and [FO89] defined different extended correctness criteria to increase concurrency by dividing a transaction into a set of atomic steps. Serializability can be viewed as a special case of these extended criteria. Formulating recoverability is also not as difficult. In fact, the purpose of recoverability is to ensure that failures of active transactions cannot affect the semantics of committed transactions. It is sound, therefore, to formulate the notion based only on the atomic units of local transactions and subtransactions. These formulations, however, make it possible to commit global transactions without using a 2PC protocol in a MDBS. To illustrate this point, consider a banking system in which a transfer transaction can be divided into two atomic subtransactions, a withdrawal subtransaction \( G_{1,1} \) at one site \( S_1 \) followed by a deposit subtransaction \( G_{1,2} \) at another site \( S_2 \). MDBS only needs to control the commit order of the two subtransactions such that \( G_{1,1} \) commits before \( G_{1,2} \) commits. Even though site \( S_2 \) may fail after \( G_{1,1} \) has committed, but before \( G_{1,2} \) commits, we can resubmit the deposit subtransaction \( G_{1,2} \) after \( S_2 \) recovers without imposing any restriction on the execution of local transactions at site \( S_2 \). We say a global transaction is locally-committed if one of its subtransactions has committed, and it is globally-committed if all of its subtransactions have committed. Clearly, the transfer transaction can be executed concurrently with a set of local transactions and be globally-committed without the use of the 2PC protocol.

Unfortunately, the above formulations are still not sufficient for a global transaction to be globally-committed in the presence of failures. This is because the failure of a site, for example, may result in the abortion of some subtransactions and after the site recovers it may execute some local transactions before it executes the aborted subtransactions due to the local autonomy requirements. The problem, as a consequence of the failure, is that these subtransactions may not be permitted to execute (and commit) without violating consistency although their executions...
are correct before the failure occurs. In fact, after inserting arbitrary local transactions between any two subtransactions in any serializable execution, we cannot guarantee the execution is still serializable; otherwise, we can maintain the consistency of a \textit{MDBS} by only considering global transaction execution, which is not true. The purpose of this paper is to define a subclass of execution histories of transactions such that if a global transaction in an execution of this subclass is globally-committed then it must remain to be globally-committed even after local transactions are arbitrarily inserted between any two subtransactions in the execution. In this paper, we define a new notion called \textit{global commitment} (GC) for such a subclass. A scheduler in \textit{MDBSs} that produces a globally commitable execution guarantees global transactions to become globally-committed after it has been locally-committed without violating the local autonomy requirements, even in the presence of failures.

The idea of dividing a transaction into a set of atomic steps was used in [GM83], [Lyn83], and [FO89] mainly for the purpose of increasing concurrency. [HS90] illustrated how this idea can be used to eliminate the need for the 2PC protocol. However, these approaches do not deal with the effects of local autonomy on global commitment issues in the presence of failures in an \textit{MDBS} environment. [GM83, GMS87] made use of the notion of \textit{compensating transactions} to guarantee the commitment or abortion of multi-step transactions. Compensating transactions are intended to handle situations where it is required to undo a transaction whose updates have been read by other transactions, without resorting to cascading aborts. The concepts of compensation and compensating transactions were formally defined in [KLS90a]. [KLS90b] described an optimistic commit method based on compensation concept for multidatabase systems. The problem with compensation is that not all transactions are compensatable, especially after their results have been read by other transactions. It is also very difficult to write compensating transactions in a \textit{MDBS} environment. The reason is that compensating transactions depend not only on the original transaction but also on other transactions executed in between. Information about these transactions (especially local ones), however, may not be available to global users. The approach, therefore, is effective only for some specific data processing application environments.

The remainder of this paper is organized as follows. In section 2, we describe a formal transaction model for multidatabase systems. We then formulate, in section 3, the SR and the RC properties based on our formal transaction model and define the notion GC as a new correctness criterion for transaction management in \textit{MDBSs}. In section 4, we discuss how to maintain an GC history in an \textit{MDBS} environment. Some concluding remarks are given in section 5.
2 The Multidatabase Transaction Model

An MDBS consists of a global transaction management system GTMS, a set D of data items and a set T of transactions. The data item set D consists of n pairwise disjoint subsets, D_1, D_2, ..., D_n, called local databases. That is, there exists no data replication in different LDBSs. The transaction set T consists of n + 1 subsets, G, L_1, L_2, ..., L_n, where L_i is a set of local transactions that access D_i only, while G is a set of global transactions that access more than one local database. We use L to denote the set of all local transactions, L_1 ∪ L_2 ∪ ... ∪ L_n. The local database D_i and the local transaction management system LTMS_i, together with the transaction set T_i = L_i ∪ G_i, where G_i is a set of global subtransactions that access D_i only, forms the local database system LDBS_i. Formally,

Definition 2.1 (local transaction or subtransaction) a local transaction or a global subtransaction T_i ∈ T_j is a partial order with ordering relation <_i where

1. T_i ⊆ \{r_i(x), w_i(x) | x ∈ D_j\} ∪ \{a_i, c_i\};
2. a_i ∈ T_i iff c_i ∉ T_i;
3. if t is c_i or a_i, and t ∈ T_i, for any other operation p ∈ T_i, p <_i t;
4. if r_i(x), w_i(x) ∈ T_i, then either r_i(x) <_i w_i(x) or w_i(x) <_i r_i(x).

That is, a local transaction or a subtransaction is a partial order of read, write, commit, and abort operations, which must specify the order of conflicting operations (two operations conflict if they both access the same data item and at least one of them is a write operation) and must contain exactly one termination operation: commit or abort. The definition of a local transaction or a subtransaction is exactly the same as that of a conventional transaction in [BH87].

Definition 2.2 (global transaction) A global transaction G_i ∈ G is a partial order with ordering relation <_g where

1. G_i ⊆ ∪_{j=1}^n \{G_{i,j}^a, G_{i,j}^c\}, where G_{i,j}^a, G_{i,j}^c are two same subtransactions of G_i in LDBS_i except a_{i,j} ∈ G_{i,j}^a and c_{i,j} ∈ G_{i,j}^c;
2. <_g ⊇ ∪_{j=1}^n <_{i,j}, where <_{i,j} is the ordering relation of subtransaction G_{i,j}^a or G_{i,j}^c in LDBS_i; and
3. let $T_{i,j}$ be $G_{i,j}^a$ or $G_{i,j}^c$ and $T_{i,k}$ be $G_{i,k}^a$ or $G_{i,k}^c$, ($i \neq k$). If the value written by $w_{i,k}(y) \in T_{i,k}$ is an arbitrary function of the values read by $r_{i,j}(x) \in T_{i,j}$, then $r_{i,j}(x) <_{g} w_{i,k}(y)$.

That is, a global transaction is also a partial order of read, write, commit, and abort operations. As far as the structure of computation is concerned, the definition of a global transaction is the same as that of a local transaction (because all read and write operations in a global transaction also form a partial order). However, our definition of a global transaction captures two new features of a global transaction execution in an $MDBS$ environment. That is, a global transaction contains more than one termination operation, one such operation for one subtransaction (condition 2) and specifies explicitly the function-dependency relation between two subtransactions (condition 3).

Modeling these features in the definition of a global transaction is necessary because we assume that pre-existing $LDBS$s may not support the prepared state of the 2PC protocol, and may not support communication facilities between two $LDBS$s. Notice that by condition 3 the value written by $w_{i,k}(y)$ in a committed subtransaction $G_{i,k}^c$ may be a function of the values read by $r_{i,j}(x)$ in an aborted subtransaction $G_{i,j}^a$. This may occur because, in an $MDBS$ environment, the $GTMS$ only has the abilities of submitting subtransactions and maintaining the function-dependency relation between two subtransactions specified by condition 3, but no total control ability to commit or abort these subtransactions. However, an aborted subtransaction does not make any effect on the local database that it has accessed.

The architecture of transaction management in an $MDBS$ is shown in Figure 1. A $LTMS_i$ controls the execution of local transactions and global subtransactions which are submitted to the $LDBS_i$, while $GTMS$ is responsible for the submission of all global transactions and maintains all function-dependency relations among subtransactions at the global level.

3 Global Commitment Theory

In this section, firstly, we define local and global histories and formulate both serializable and recoverable histories based on our transaction model. We then define a new correctness notion called global commitment. It is assumed that the reader is familiar with the basic theory and notations of serializability and recoverability in conventional database systems (see, e.g., [BH87]).
Figure 1: The architecture of transaction management for an MDBS

3.1 Histories

An execution of a set of transactions can be described by a history. A history contains read, write, abort, and commit operations and their execution orders. The execution of local transactions and global subtransactions at $LDBS_i$ constitutes the local history $h_i$. A global history $H$ over $G \cup \mathcal{L}$ is the set of all local histories with all the function-dependency relations between any two subtransactions. Formally,

**Definition 3.1 (local history)** let $T_i = \{T_1, T_2, ..., T_m\}$ be a set of all local transactions and subtransactions in $LDBS_i$. A local history $h_i$ over $T_i$ is a partial order with ordering relation $<_{h_i}$ where:

1. $h_i = \{T_1, T_2, ..., T_m\}$;
2. $<_{h_i} \supseteq \bigcup_{j=1}^{m} <_{j}$, where $<_{j}$ is the ordering relation of local transaction or subtransaction $T_j$ in $LDBS_i$; and
3. for any two conflicting operations $p, q \in h_i$, either $p <_{h_i} q$ or $q <_{h_i} p$.

$\square$

**Definition 3.2 (global history)** A global history $H$ over $G \cup \mathcal{L}$ is a partial order with ordering relation $<_H$ where:
1. $H = \{h_1, h_2, ..., h_n\}$, where $h_i$ is a local history in $LDBS_i$;

2. $<_H \supseteq \bigcup_{i=1}^n <_{h_i}$, where $<_{h_i}$ is the ordering relation of local history $h_i$ in $LDBS_i$; and

3. let $T_{i,j}$ be $G_{i,j}^c$ or $G_{i,j}^g_i$ and $T_{i,k}$ be $G_{i,k}^c$ or $G_{i,k}^g_i$, $(i \neq k)$. If the value written by $w_{i,k}(y) \in T_{i,k}$ is an arbitrary function of the values read by $r_{i,j}(x) \in T_{i,j}$, then $r_{i,j}(x) <_H w_{i,k}(y)$.

A local transaction $L_i$ is committed in global history $H$ if $c_i \in H$. A global transaction $G_i$ that accesses local databases $D_1, D_2, ..., D_n$ is locally-committed in global history $H$ if $c_{i,j} \in H$ for some $j$, $1 \leq j \leq n$, and is globally-committed in global history $H$ if $c_{i,j} \in H$ for all $j$, $1 \leq j \leq n$. Given a global history $H$, the committed projection of $H$, denoted $C(H)$, is the history obtained from $H$ by deleting all operations that do not belong to local transactions committed in $H$ or global transactions locally-committed in $H$. The reason that a locally-committed global transaction is included into $C(H)$ is that once a global transaction becomes locally-committed the position of the global transaction relative to other transactions in $H$ is fixed because other committed transactions may have read the result of the locally-committed global transaction.

**Example 3.1** Consider an MDBS consisting of two $LDBS$s: $LDBS_1$ and $LDBS_2$, where data items $a$ and $b$ belong to database $D_1$ at $LDBS_1$, and $c$ and $d$ belong to database $D_2$ at $LDBS_2$. Let $L_1$ and $L_2$ be two local transactions submitted at $LDBS_1$ and $LDBS_2$, respectively:

$L_1 : r_{l1}(b), w_{l1}(b), c_{l1}$

$L_2 : w_{l2}(d), c_{l2}$

Let $G_1$ be a global transaction consisting of four subtransactions, $G_{1,1}^a$, $G_{1,1}^g$, $G_{1,2}^a$ and $G_{1,2}^g$ where both $G_{1,1}^a$ and $G_{1,1}^g$ are the two same subtransactions submitted at $LDBS_1$ except that $G_{1,1}^g$ is aborted and $G_{1,1}^a$ is committed for $i=1,2$. There is a function-dependency relation $r_{g1,2}(d) <_H w_{g1,1}(b)$ between $G_{1,1}^g$ and $G_{1,1}^a$.

$G_{1,1}^a: r_{g1,1}(a), w_{g1,1}(b), a_{g1,1}$; $G_{1,1}^g: r_{g1,1}(a), w_{g1,1}(b), a_{g1,1}$;

$G_{1,2}^a: r_{g1,2}(d), c_{g1,2}$; $G_{1,2}^g: r_{g1,2}(d), c_{g1,2}$;

Let $h_1$ and $h_2$ be two local histories at $LDBS_1$ and $LDBS_2$, respectively:

$h_1: r_{g1,1}(a), w_{g1,1}(b), a_{g1,1}, r_{l1}(b), w_{l1}(b), c_{l1}, r_{g1,1}(a), w_{g1,1}(b), a_{g1,1}$

$h_2: r_{g1,2}(d), a_{g1,2}, r_{l2}(c), w_{l2}(d), c_{l2}, r_{g1,2}(d), a_{g1,2}$;
Then a global history $H$ and its committed projection $C(H)$ over $\{ L_1, L_2, G \}$ with $r_{g1,2}^a(d) <_H w_{g1,1}^c(b)$ between $G_2^c$ and $G_1^c$ are

$$
H=C(H): \begin{array}{c}
G_1^c, G_2^c, L_1, L_2, G_{i,1}, G_{i,2}, G_{i,2}, G_{i,1},
\begin{array}{c}
G_1^c, G_2^c, L_1, L_2, G_{i,1}, G_{i,2}, G_{i,2}, G_{i,1},
\end{array}
\end{array}
$$

\[ r_{g1,1}^a(a), w_{g1,1}^c(b), a_{g1,1}, r_{g1,2}^a(d), a_{g1,2}, r_{l2}^c(c), w_{l2}^c(d), c_{l2},
\end{array}
\begin{array}{c}
G_1^c, G_2^c, L_1, L_2, G_{i,1}, G_{i,2}, G_{i,2}, G_{i,1},
\end{array}
\end{array}
$$

3.2 Serializability and Recoverability

We define two global histories $H$ and $H'$ to be conflict equivalent, denoted $H \equiv_c H'$, if:

1. they are defined over the same set of transactions and have the same operations; and

2. if $p_i <_H q_j$ then $p_i <_{H'} q_j$, for any conflicting operations $p_i$ and $q_j$ belonging to local transactions or subtransactions $T_i$ and $T_j$ (respectively) where either:

   (a) $a_i, a_j \notin H$, or

   (b) $a_i \in H$ only if $p_i$ is a read operation specified in a function-dependency relation of a global transaction.

A global history $H$ is serial if, for every two (local or global) transactions $TR_i$ and $TR_j$ that appear in $H$, either all operations of $TR_i$ appear before all operations of $TR_j$ or vice versa. A global history $H$ is serializable (SR) if its committed projection, $C(H)$, is conflict equivalent to a serial history.

The global serialization graph (SG) for $H$, denoted $SG(H)$, is a directed graph whose nodes are all local or global transactions $TR$s in $G \cup \mathcal{L}$ that are committed (or locally-committed) in $H$, and whose edges are all $TR_i \rightarrow TR_j (i \neq j)$ such that:

1. one of $TR_i$’s operations, $o_i$, precedes and conflicts with one of $TR_j$’s operations, $o_j$, and

2. if $TR_i$ or $TR_j$ are global transactions, then either:

   (a) $o_i$ or $o_j$ do not belong to some aborted subtransactions of $TR_i$ or $TR_j$, respectively, or

   (b) $o_i$ is a read operation specified in a function-dependency relation of global transaction $TR_i$.

Example 3.2 In Example 3.1, $r_{g1,2}^a(d)$ is read operation specified in the function-dependency relation $r_{g1,2}^a(d) <_H w_{g1,1}^c(b)$ and $r_{g1,2}^a(d)$ precedes and conflicts with $w_{l2}$, so $G_1 \rightarrow L_2$. On the
other hand, \( w_2 \) precedes and conflicts with \( r_{g1,2}(d) \), so \( L_2 \rightarrow G_1 \). That is, there exists a cycle between \( L_2 \) and \( G_1 \). The global serialization graph \( SG(H) \) is as follows:

\[
L_1 \rightarrow G_1 \Rightarrow L_2
\]

\( \Box \)

The global serializability theorem can then be stated as follows:

**Theorem 3.1** A history \( H \) is serializable iff \( SG(H) \) is acyclic.

**Proof:** The proof is analogous to the proof of theorem 2.1 in [BHG87] and is omitted here. \( \Box \)

The read-from relation will be used to define another important property of a “correct” history, namely recoverability. A transaction \( T_i \) reads data item \( x \) from \( T_j \) in history \( H \) if (1) \( w_j(x) < r_j(x) \); (2) \( a_j \not< r_j(x) \); and (3) if there is some \( w_k(x) \) such that \( w_j(x) < w_k(x) < r_i(x) \), then \( a_k < r_i(x) \).

Let \( T_i, T_j \in T_k \) be local transactions or subtransactions in \( LDBS_k \). A global history \( H \) is called recoverable if, whenever \( T_i \) reads from \( T_j \) \((i \neq j)\) in \( H \), then \( c_j < c_i \). The set of recoverable global histories is denoted \( RC \). A global history \( H \) avoids cascading aborts if, whenever \( T_i \) reads from \( T_j \) \((i \neq j)\) in \( H \) and \( c_j < r_i(x) \). The set of global histories that avoid cascading aborts is denoted \( ACA \). A global history \( H \) is strict if, whenever \( w_j(x) < o_i(x)(i \neq j) \), either \( a_j < o_i(x) \) or \( c_j < o_i(x) \) where \( o_i \) is \( r_i(x) \) or \( w_i(x) \). The set of strict global histories is denoted \( ST \). A global history \( H \) is rigorous if (1) it is strict, and (2) whenever \( r_j(x) < w_i(x)(i \neq j) \), either \( a_j < w_i(x) \) or \( c_j < w_i(x) \). The set of strict global histories is denoted \( RG \) (the RG notion was first defined in [BGRS90, GRS91]).

The next theorem says that recoverability, avoiding cascading aborts, strictness, and rigorousness are increasingly restrictive properties.

**Theorem 3.2** \( RG \subset ST \subset ACA \subset RC \)

**Proof:** The proof of \( RG \subset ST \) appears in [GRS91], and the proof of \( ST \subset ACA \subset RC \) appears in [BHG87]. \( \Box \)

It is easy to verify that \( SR, RC, ACA, ST, \) and \( RG \) are prefix commit-closed properties. A property of histories is called prefix commit-closed if, whenever the property is true of history \( H \), it is also true of history \( C(H') \), for any prefix \( H' \) of \( H \). A correctness criterion for histories that accounts for transaction and system failures must be described by such a property [BHG87].

Note that by the definition of \( C(H) \), we know that all locally-committed global transactions are included in the definition of a serializable history. This is because the result of committed
subtransactions in a locally-committed global transaction has become persistent in databases and will affect the concurrent execution of transactions. On the other hand, The purpose of recoverability (or ACA, ST, RG) is to ensure that the failure of active transactions cannot cause the semantics of committed (or locally-committed) transactions to change so it is sound to define a recoverable (or ACA, ST, RG) history based on the atomic units of local transactions and subtransactions.

### 3.3 Global Commitment

Recall that a global transaction is divided into several atomic subtransactions. The commitment of a global transaction, therefore, means all its subtransactions are committed. In this paper, we restrict the failure atomicity of the global transaction is restricted to be “if one subtransaction commits, then all other subtransactions will eventually commit”[HS90]. The implication of the restriction is that every global transaction must be designed to be globally committable if different subtransactions of the global transaction will not use different values of a data item. How to divide a global transaction into atomic subtransactions is related to semantics of global transactions, which is beyond the scope of this paper. In the rest of the paper, we assume that all global transactions are globally committable when they are run one by one. The serializability and recoverability notions, however, are not enough for a global transaction to become globally-committed in an MDBS environment with the presence of failures. Consider the following examples.

**Example 3.3** Consider an MDBS consisting of two LDBSs: LDBS1 and LDBS2, where data items a and b belong to database D1 at LDBS1, and c and d belong to database D2 at LDBS2. Let G1 be a global transaction consisting of three subtransactions, G1,1, G1,2, and G1,2. There is a function-dependency relation \( r_{g1,2}(d) \prec w_{g1,1}(b) \) between G1,2 and G1,1.

\[
\begin{align*}
G_{1,1}^c: & r_{g1,1}(a), w_{g1,1}(b), c_{g1,1}; \\
G_{1,2}^c: & r_{g1,2}(d), c_{g1,2}; \\
G_{1,2}^a: & r_{g1,2}(d), a_{g1,2};
\end{align*}
\]

Then, the following global execution is serializable and recoverable (so are any of its prefixes) because it contains only G1, i.e. G1 executes in isolation.

\[
\begin{array}{cccc}
H: & r_{g1,2}(d), a_{g1,2}, r_{g1,1}(a), w_{g1,1}(b), c_{g1,1}, r_{g1,2}(d), c_{g1,2}; \\
\end{array}
\]

The above execution occurs when G1,1 successfully commits at LDBS1, but G1,2 is aborted because of the failure of site S2 after GTMS decides to commit global transaction G1. However, after S2 recovers it may run another local transaction L1 before it runs the resubmitted subtransaction G1,2. After the local transaction is executed the global history becomes:
That is, there will not be a chance for subtransaction \( G_{1,2} \) to commit. Otherwise, the global transaction \( G_1 \) will see an inconsistent data item \( d \).

Example 3.4 Consider an \( MDBS \) consisting of two \( LDBSs \): \( LDBS_1 \) and \( LDBS_2 \), where data items \( a \) and \( b \) belong to database \( D_1 \) at \( LDBS_1 \), and \( c \) and \( d \) belong to database \( D_2 \) at \( LDBS_2 \).

Let \( G_1 \) be a global transaction consisting of two subtransactions, \( G_{1,1} \) and \( G_{1,2} \):

\[
\begin{align*}
G_{1,1} &: w_{g_{1,1}}(a), c_{g_{1,1}}; \\
G_{1,2} &: w_{g_{1,2}}(c), c_{g_{1,2}};
\end{align*}
\]

Let \( G_2 \) be another global transaction consisting of two subtransactions, \( G_{2,1} \) and \( G_{2,2} \):

\[
\begin{align*}
G_{2,1} &: w_{g_{2,1}}(b), c_{g_{2,1}}; \\
G_{2,2} &: w_{g_{2,2}}(d), c_{g_{2,2}};
\end{align*}
\]

Then, the following global execution is serializable and recoverable (so are any of its prefixes) because there is no conflicting operation between the two global transactions.

\[
H: \begin{align*}
g_{1,1} &: w_{g_{1,1}}(a), c_{g_{1,1}}; \\
g_{2,2} &: w_{g_{2,2}}(d), c_{g_{2,2}}; \\
g_{2,1} &: w_{g_{2,1}}(c), c_{g_{2,1}}; \\
g_{1,2} &: w_{g_{1,2}}(b), c_{g_{1,2}};
\end{align*}
\]

Suppose that \( G_{1,1} \) and \( G_{2,2} \) successfully commit, but \( G_{1,2} \) and \( G_{2,1} \) are aborted because of the failures of site \( S_1 \) and site \( S_2 \) after \( GTMS \) decides to commit both \( G_1 \) and \( G_2 \). However, after the two sites recover they may run other local transactions \( L_1 \) and \( L_2 \) at \( S_1 \) and \( S_2 \) before they run the resubmitted subtransactions \( G_{1,2} \) and \( G_{2,1} \). After the local transactions are executed the global history becomes

\[
H': \begin{align*}
g_{1,1} &: w_{g_{1,1}}(a), c_{g_{1,1}}; \\
g_{2,2} &: w_{g_{2,2}}(d), c_{g_{2,2}}; \\
s_{1,2} &: \tau_{i1}(a), \omega_{i1}(b), c_{i1}; \\
s_{2,1} &: \tau_{12}(d), \omega_{12}(c), c_{12}; \\
l_1 &: \tau_{11}(a), \omega_{11}(b), c_{11}; \\
l_2 &: \tau_{22}(d), \omega_{22}(c), c_{22};
\end{align*}
\]

That is, there will not a chance for subtransactions \( G_{1,2} \) and \( G_{2,1} \) to commit. Otherwise, transactions \( G_1, G_2, L_1, \) and \( L_2 \) will not be serializable.

The above examples show that a new notion for reliable transaction executions appropriate for an \( MDBS \) environment must be defined so that every global transaction is globally-committed. From the observation of above examples, we have already noted that local transactions may arbitrarily be appended to the committed projection of any prefix \( H' \) of a history \( H \), i.e. \( C(H') \) due to the local autonomy requirements. Thus, in order to account for the effect of local autonomy,
a global history must be *prefix local extension-closed*, i.e. every global transaction remains to
become globally-committed after local transactions are arbitrarily appended to $C(H')$ of any
prefix $H'$ of a history $H$.

First, we consider the characteristics of a single global transaction with local extensibility. We
observe from Example 3.1 that local transactions may result in the inconsistent use of data items
in a global transaction. If all function-dependency relations within a global transaction can be
maintained within committed subtransactions, then the incorrect use can be avoided. Formally,

**Definition 3.3 (primitive transaction)** A global transaction $G_i$ is primitive if, for every two
subtransactions $G_{i,j}$ in $LDBS_j$ and $G_{i,k}$ in $LDBS_k$ ($j \neq k$) of $G_i$ where:

1. $r_{i,j}(x) < G_i w_{i,k}(y)$, where $r_{i,j}(x) \in G_{i,j}$ and $w_{i,k}(y) \in G_{i,k}$, and
2. $c_{i,k} \in G_{i,k}$ (i.e. $G_{i,k}$ commits)

then $c_{i,j} \in G_{i,j}$.

That is, if a global transaction is primitive, then any two subtransactions (in two sites) of a
global transaction will not see different values of a data item. This is because function-dependency
relations are the only sources from which two subtransactions in different sites may use the same
data item (note that we assume no data replication exists in an $MDBS$). However, It is easy to
see that this property is not prefix extension-closed. Thus, a slightly stronger definition must be
defined so that it is prefix extension-closed.

**Definition 3.4 (dependency-equivalent)** Two global histories $H$ and $H'$ are said to be depen­
dency equivalent, denoted $H \equiv_d H'$, if (1) they are defined over the same set of transactions
and have the same operations; and (2) for any operations $r_{i,j}(x)$ of $G_{i,j}$ and $w_{i,k}(y)$ of $G_{i,k}$ in $G_i$,
if $r_{i,j}(x) < H w_{i,k}(y)$ then $r_{i,j}(x) < H' w_{i,k}(y)$.

**Definition 3.5 (intra-serial)** A global history $H$ is intra-serial if for every two subtransactions
$G_{i,j}$ and $G_{i,k}$ of global transactions $G_i$ in $H$, either all operations of $G_{i,j}$ appear before all opera­
tions of $G_{i,k}$ such that $r_{i,k} \not< H w_{i,j}$ for any $w_{i,j}$ in $G_{i,j}$ and any $r_{i,k}$ in $G_{i,k}$, or vice versa.

**Definition 3.6 (intra-committable)** A global history $H$ is intra-committable if every global
transaction $G_i$ in $C(H)$ is primitive and the committed projection of $H$, $C(H)$, is depen­
dency-equivalent to an intra-serial history $H^*_s$ such that, for every two committed subtransactions $G_{i,j}$ in
$LDBS_j$ and $G_{i,k}$ in $LDBS_k$ of $G_i$, i.e. $c_{i,j} \in G_{i,j}$ and $c_{i,k} \in G_{i,k}$, $G_{i,j} < H^*_s G_{i,k}$ iff $c_{i,j} < H c_{i,k}$.
That is, if a global history is intra-committable, then all function-dependency relations of every global transaction is correctly maintained, i.e. a global transaction can always be globally-committed when it runs concurrently with a set of local transactions (note that every global transaction is designed to be globally committable if the data items that its subtransactions use are consistent). It is easy to see that this property is prefix extension-closed. Next, we will define the interleaving relationships between two different global transactions so that two or more global transactions could become globally-committed when they run concurrently. One way to globally commit every global transaction is not to allow global transactions to be interleaved at all.

**Definition 3.7 (inter-serial)** A global history $H$ is inter-serial if for every global transactions $G_i$ and every other (global or local) transaction $T_j$ in $H$, either all operations of $G_i$ appear before all operations of $T_j$, or vice versa.

**Definition 3.8 (inter-committable)** A global history $H$ is inter-committable if its committed projection, $C(H)$ is conflict-equivalent to an inter-serial history $H^*$ such that, for every committed subtransaction $G_{i,k}$, i.e. $c_{i,k} \in G_{i,k}$, and every other committed local or subtransaction $T_{j,k}$, i.e. $c_{j,k} \in T_{j,k}$, in every LDBS$_k$, $G_i < H^* T_j$ iff $c_{i,k} < H c_{j,k}$.

That is, if a global history is inter-committable, then the commitment of any subtransaction of one global transaction will not prevent other global transactions from becoming globally-committed. This is because in an inter-committable history every global transaction is executed as if it were executed alone in an MDDBS. This property is also prefix extension-closed. In summary, we have,

**Definition 3.9 (Global Commitment)** A global history $H$ is globally committable if it is both intra-committable and inter-committable. The set of globally committable histories is denoted $GC$.

Intra-commitment means that if, for two global subtransactions $G_{i,j}$ in LDBS$_j$ and $G_{i,k}$ in LDBS$_k$ ($i \neq k$) of $G_i$, the value written by $w_{i,k}(y) \in G_{i,k}$ is an arbitrary function of the values read by $r_{i,j}(x) \in G_{i,j}$, i.e. $r_{i,j}(x) < H w_{i,k}(y)$, then $G_{i,j}$ should commit before $G_{i,k}$ commits. Inter-commitment limits the commit order of subtransactions in a global history such that all subtransactions in a global transaction are committed before all subtransactions in the other global transaction, or vice versa. If a global history is globally committable then every global transaction in the history could be globally-committed.

We can determine whether a history is globally committable by analyzing two graphs derived from the history called intra-commitment graph and inter-commitment graph.
Definition 3.10 (intra-commitment graph) The intra-commitment graph (DCG) for $H$, denoted $\text{DCG}(H)$, is a directed graph whose nodes are all $T$s and whose edges are all $T_{i,j} \rightarrow T_{i,k}$ for every global transaction $G_i$ such that (1) $r_{i,j}(x) < H w_{i,k}(y)$, i.e. the value written by $w_{i,k}(y)$ in $G_i^c$ or $G_i^a$ is an arbitrary function of the values read by $r_{i,j}(x)$ in $G_i^c$ or $G_i^a$; or (2) $c_{i,j} < H c_{i,k}$ when both $G_{i,j}$ and $G_{i,k}$ are in $G_i$. 

Definition 3.11 (inter-commitment graph) The inter-commitment graph (CCG) for $H$, denoted $\text{CCG}(H)$, is a directed graph whose nodes are all the local or global transactions $T$s in $G \cup \mathbb{L}$ that are committed (or locally-committed) in $H$ and whose edges are all $T_i \rightarrow T_j (i \neq j)$ such that (1) one of $T_i$'s operations precedes and conflicts with one of $T_j$'s operations, or (2) $c_{i,k} < H c_{j,k}$ when both $G_{i,k}$ and $G_{j,k}$ are in $G_i$. 

Theorem 3.3 A global history $H$ is intra-committable iff $\text{DCG}(H)$ is acyclic.

Proof: Without loss of generality, assume $H$ is a global history with only one committed (or locally-committed) global transaction $G_i$. 

(if) Let $T_{i,1}, ..., T_{i,m}$ be the nodes of $\text{DCG}(H)$. Since $\text{DCG}(H)$ is acyclic it may be topologically sorted. Let $j_1, ..., j_m$ be a permutation of $1,2,...,m$ such that $T_{i,j_1}, T_{i,j_2}, ..., T_{i,j_m}$ is a topological sort of $\text{DCG}(H)$. Let $H^d_s$ be the intra-serial history such that $T_{i,j_1}$ precedes $T_{i,j_2}$, $T_{i,j_2}$ precedes $T_{i,j_3}$, ..., and so on. We claim that $H$ is intra-committable. In fact, because $r_{i,k}(x) < H w_{i,l}(y)$ implies $c_{i,k} < H c_{i,l}$ (otherwise, contradicting with the acyclicity of $\text{DCG}(H)$), the $G_i$ is primitive. On the other hand, if $r_{i,k}(x) < H w_{i,l}(y)$, then $T_{i,k} \rightarrow T_{i,l}$ is an edge in $\text{DCG}(H)$ by the definition of $\text{DCG}(H)$. Therefore in any topological sort of $\text{DCG}(H)$, $T_{i,k}$ must appear before $T_{i,l}$, and in particular, $r_{i,k}(x) < H^d_s w_{i,l}(y)$. Thus $C(H) \equiv_d H^d_s$. Furthermore, by the definition of $\text{DCG}(H)$, we have $T_{i,k} < H^d_s T_{i,l}$ iff $c_{i,k} < H c_{i,l}$. Thus, $H$ is intra-committable.

(only if) Suppose $H$ is intra-committable. Let $H^d_s$ be an intra-serial history such that $C(H) \equiv_d H^d_s$. Consider an edge $T_{i,k} \rightarrow T_{i,l}$ in $\text{DCG}(H)$. Thus, either there are two function-dependency operations $r_{i,k}(x), w_{i,l}(y)$ of $T_{i,k}, T_{i,l}$ (respectively) such that $r_{i,k}(x) < H w_{i,l}(y)$, or $c_{i,k} < H c_{i,l}$. Because $C(H) \equiv_d H^d_s$, $r_{i,k}(x) < H w_{i,l}(y)$ implies $r_{i,k}(x) < H^d_s w_{i,l}(y)$, i.e. $T_{i,k} < H^d_s T_{i,l}$. On the other hand, $c_{i,k} < H c_{i,l}$ also implies $T_{i,k} < H^d_s T_{i,l}$ because $H$ is intra-committable. Thus, we've shown that if $T_{i,k} \rightarrow T_{i,l}$ is in $\text{DCG}(H)$ then $T_{i,k} < H^d_s T_{i,l}$. Now suppose there is a cycle in $\text{DCG}(H)$ and without loss of generality let that cycle be $T_{i,1} \rightarrow T_{i,2} \rightarrow ..., \rightarrow T_{i,k} \rightarrow T_{i,1}$. These edges imply that in $T_{i,1} < H^d_s T_{i,2} < H^d_s ..., < H^d_s T_{i,1}$, i.e. $T_{i,1} < H^d_s T_{i,1}$, a contradiction! So no cycle can exist in $\text{DCG}(H)$.

Because the function-dependencies only exist within global transactions, the above proof is valid for any global history with more than one committed (or locally-committed) global transactions. 

\(\square\)
Theorem 3.4 A global history $H$ is inter-committable iff $CCG(H)$ is acyclic.

Proof: (if) Suppose $H$ is a global history over all local or global transactions in $T = L \cup G$. Let $T = \{T_1, T_2, ..., T_n\}$. Without loss of generality, assume $T_1, T_2, ..., T_m$ ($m \leq n$) are all committed local or global transactions in $T$. Let $T_1, T_2, ..., T_m$ be the nodes of $CCG(H)$. Since $CCG(H)$ is acyclic it may be topologically sorted. Let $i_1, ..., i_m$ be a permutation of $1,2, ..., m$ such that $T_{i_1}, T_{i_2}, ..., T_{i_m}$ is a topological sort of $CCG(H)$. Let $H^*_i$ be the inter-serial history $T_{i_1}, T_{i_2}, ..., T_{i_m}$. We claim that $H$ is inter-committable such that $T_i <_{H^*_i} T_j$ iff $c_{i,k} <_H c_{j,k}$ for every committed subtransaction $G_{i,k}$ in $C(H)$ and every other committed local transaction or subtransaction $T_{j,k}$ in $C(H)$ in every LDBS. To see this, let $p_i \in T_i$ and $q_j \in T_j$ where $T_i, T_j$ are committed in $H$. Suppose $p_i, q_j$ conflict and $p_i <_H q_j$. $p_i <_H q_j$ implies $T_i \rightarrow T_j$ is an edge in $CCG(H)$ and $c_{i,k} <_H c_{j,k}$ for all $k$ (otherwise, contradicting with the definition of $CCG(H)$). Therefore in any topological sort of $CCG(H)$, $T_i$ must appear before $T_j$, and in particular, $p_i <_{H^*_i} q_j$. Thus $C(H) \equiv_c H^*_i$. Furthermore, by $CCG(H)$ is acyclic we have $p_i <_H q_j$ iff $c_{i,k} <_H c_{j,k}$ for all $k$, and by $C(H) \equiv_c H^*_i$, $T_i <_H T_j$ iff $c_{i,k} <_H c_{j,k}$ for all $k$. Thus, $H$ is inter-committable.

(only if) Suppose $H$ is an inter-committable. Let $H^*_i$ be an inter-serial history such that $C(H) \equiv_c H^*_i$. Consider an edge $T_i \rightarrow T_j$ in $CCG(H)$. Thus, either there are two conflicting operations $p_i, q_j$ of $T_i, T_j$ (respectively) such that $p_i <_H q_j$, or $c_{i,k} <_H c_{j,k}$ for some $k$. Because $C(H) \equiv_c H^*_i$, $p_i <_H q_j$, i.e. $T_i <_{H^*_i} T_j$. On the other hand, $c_{i,k} <_H c_{j,k}$ also implies $T_i <_{H^*_i} T_j$ because $H$ is inter-committable. Thus, we've shown that if $T_i \rightarrow T_j$ is in $CCG(H)$ then $T_i <_{H^*_i} T_j$. Now suppose there is a cycle in $CCG(H)$ and without loss of generality let that cycle be $T_1 \rightarrow T_2 \rightarrow, ..., \rightarrow T_k \rightarrow T_1$. These edges imply that in $T_1 <_{H^*_i} T_2 <_{H^*_i}, ..., <_{H^*_i} T_1$, i.e. $T_1 <_{H^*_i} T_1$, a contradiction! So no cycle can exist in $CCG(H)$. □

By the definition of global commitment and the above Theorem 3.3 and 3.4, we have,

Theorem 3.5 A global history $H$ is globally committable iff both $DCG(H)$ and $CCG(H)$ are acyclic.

Like SR and RC, GC is also a prefix commit-closed property because a prefix extension-closed property is also a prefix commit-closed. That is, if $H$ is a GC history, then for any prefix $H'$ of $H$, $C(H')$ is also GC. A global history $H$ in a MDBS environment, therefore, is correct if it is SR, RC and GC, i.e. $H \in SR \cap RC \cap GC$.

It is obvious that GC intersects with both SR and RC, but is incomparable to them. In fact, a GC history does not impose any restriction on the relationship between any two local transactions. Thus, there may be a GC history that is not serializable nor recoverable. On the other hand, there are some global histories that are serializable and recoverable (even rigorous),

16
but not globally committable, e.g. the global histories in Example 3.3 and Example 3.4. Figure 2 illustrates the relationship among GC, SR, and RC etc. sets. It can be verified that all regions shown in Figure 2 are non-empty, even though we do not provide the histories required to show this.

4 Reliable Transaction Management

In this section, we will discuss how to produce a globally committable history in an $MDBS$ environment. Recall that a multibase database system is an integration of pre-existing autonomous local database systems. Furthermore, an inter-committable history requires that the commit order between a local transaction and a global subtransaction must be consistent with the order of the conflict operations between them. So, a correct global history depends not only on the global transaction scheduler strategies but also on the local transaction scheduler properties. We first study the local transaction management requirements and then give two sufficient conditions that guarantee an $MDBS$ to produce globally committable histories.
4.1 Local Transaction Management Requirements

In a multidatabase system environment, we assume that the global transaction management has no way to control the execution of local operations. As the consequence of the assumption, we should impose some requirements on local transaction management so that the interleaved execution of a local transaction and a global subtransaction satisfies the requirements of a globally committable history.

For example, a local transaction scheduler that produces a strict and serializable history is not sufficient to ensure that a global history is inter-committable. Consider the following strict and serializable history of a local transaction $L_1$ and two global subtransactions $G_{1,i}$ and $G_{2,i}$ in $LDBS_i$:

\[
S_i: r_{g1,i}(a), w_{g1,i}(a), c_{g1,i}, r_{g2,i}(b), c_{g2,i}, w_{l1}(b), c_{l1}
\]

The serialization order of the transactions in $LDBS_i$ is $G_2 \rightarrow L_1 \rightarrow G_1$, but $G_1$ is committed before $L_1$. Therefore, the scheduler on an $LDBS$ that is strict and serializable is not sufficient to produce a global inter-committable history.

In [BGRS90, GRS91], a notion of analogous execution and serialization order was defined. Transactions in a serializable history $H$ have analogous execution and serialization orders if for any pair of transactions $T_i$ and $T_j$ such that $T_i$ is committed before $T_j$ in $H$, $T_i$ is also serialized before $T_j$ in $H$. It is obvious that if a serializable scheduler in an $LDBS$ produces an analogous execution and serialization orders of transactions, then it also produces a local history in which the commit order between a global subtransaction and another local transaction (or subtransaction) is consistent with the order of the conflict operations between them, but the reverse is not true because we do not restrict the commit operation orders among local transactions in our definition of inter-committable history.

[BGRS90, GRS91], furthermore, proved that if a transaction management mechanism ensures rigorousness, then it produces conflict serializable histories in which transaction execution and serialization orders are analogous. We conclude that a rigorous local scheduler is sufficient to ensure that every global subtransaction can be committed before or after every other local transaction (or subtransaction) based on their serialization orders in the local site.

Fortunately, most popular local concurrency control mechanisms such as strict 2PL, conservative TO etc. produce rigorous histories [BGRS90, GRS91].
4.2 Maintaining Global Commitment

At this point we are in a position to discuss how the global transaction management system (GTMS) controls the submission of global subtransactions to LDBSs. Once we assume that every local transaction management system LTMS produces a rigorous serializable history, it is obvious from the definition of global commitment that the GTMS only needs to control the commit operation orders of all global transactions to produce a one phase committable history.

A sufficient condition for a scheduler to produce a globally committable history consists of two parts, one for an intra-committable history, and the other for an inter-committable history. An intra-committable history requires that (1) every global transaction \( G \) in the history is primitive, i.e. if some write operation of one committed subtransaction is function-dependent on some read operation of another subtransaction then the latter subtransaction also should be committed, (2) for every global transaction \( G \) there is an intra-serial global transaction \( G'_a \) such that \( C(G) \equiv_d G'_a \) (because the function-dependencies only exist within each global transaction, \( C(H) \equiv_d H'_a \) means \( C(G) \equiv_d G'_a \) for every global transaction \( G \) in \( H \)), and (3) the commit order of two subtransactions should be consistent with that of the function-dependency between the two subtransactions. Conditions (1) and (2) characterize the computation structure that a global transaction must satisfy in an intra-committable history. We call the global transaction satisfying conditions (1) and (2) a simple global transaction.

In an MDBS, the function-dependency relation between two subtransactions can be maintained by the global transaction management system GTMS in the following way: the GTMS collects and maintains the read values of all function-dependency relations from committed transactions and passes them to all dependent subtransactions until all these dependent subtransactions are committed. Note that the GTMS should guarantee that a dependent subtransaction should commit after the subtransaction on which it depends commits. Otherwise, the failure of the site of the subtransaction on which the dependent subtransaction depends, may damage the primitive requirement of a global transaction.

The above observation is stated in the following sufficient condition for an intra-committable scheduler.

**Theorem 4.1 (Sufficient Condition 1)** An MDBS maintains an intra-committable history if the following conditions are satisfied:

1. all local concurrency control mechanisms in the MDBS are rigorous,
2. all global transactions are simple, and
3. a subtransaction can commit iff the subtransactions on which it depends have committed.
Similarly, if we assume that every local concurrency control mechanism in the MDBS produces a rigorous serializable history, a global inter-committable history can be maintained simply by requiring that the commit operation orders of all subtransactions in an LDBS are consistent with those in other LDBSs. That is, if $G_{i,k}$ of $G_i$ commits before $G_{j,k}$ of $G_j$ in $LDBS_k$, then $G_{i,l}$ of $G_i$ also must commit before $G_{j,l}$ of $G_j$ in all other $LDBS_l$. We have the following sufficient condition for an inter-committable scheduler.

**Theorem 4.2 (Sufficient Condition 2)** An MDBS maintains an inter-committable history if the following conditions are satisfied:

1. all local concurrency control mechanisms in the MDBS are rigorous,
2. there is a total order of all global transactions such that, for every two global transactions $G_i$ and $G_j$, if $G_{i,k}$ of $G_i$ commits before $G_{j,k}$ of $G_j$ in $LDBS_k$, then $G_{i,l}$ of $G_i$ also commits before $G_{j,l}$ of $G_j$ in other $LDBS_l$ in which both global transactions appear.

The above two sufficient conditions guarantee that an MDBS maintains a globally committable history without imposing any restriction on the execution of local transactions. The algorithms in [BST90] implement the second sufficient condition to ensure that an MDBS produces an inter-committable history.

5 Conclusions

We have extended the definition of a conventional transaction to model the execution of a global (or multi-site) transaction in the context of a multidatabase system. The extension includes two aspects of operations in a global transaction. First, the extension divides a global transaction into a set of atomic subtransactions. A subtransaction accesses only the data items in a local database. The extension makes it simple and effective to model the executions of transactions in the multidatabase environment in which some pre-existing local DBMSs may not support the prepared state of the 2PC protocol. Second, the extension refines the definition of read and write operations within a global transaction such that the function-dependency relation between the read operation in a subtransaction and the write operation in another subtransaction is explicitly specified in the definition of a global transaction. The refinement makes it possible to commit a global transaction without using the 2PC protocol even in the presence of failures in the context of multidatabase systems.
Note that in the conventional definition of transactions the function-dependency relation between read and write operations within a transaction is not explicitly defined so the values written must be considered as uninterpreted functions of all data values read at the previous read operations of the same transaction and all the read and write operations of a transaction form an arbitrary partial order [Pap79, BHG87]. The effect of the function-dependency relation on the concurrency control of multidatabase systems in the absence of failures was discussed in [DEK90b].

However, a globally committable history $H$ requires that all global transactions in $H$ are simple, i.e. the function-dependency relations do not result in a cycle among subtransactions in a global transaction. For example, let a global transaction $G_1$ consist of two subtransactions $G_{1,1}$ and $G_{1,2}$ in site $S_1$ and $S_2$, respectively:

$$G_{1,1}: r_{1,1}(a), w_{1,1}(a), c_{1,1};$$
$$G_{1,2}: r_{1,2}(b), w_{1,2}(b), c_{1,2};$$

If there exists a function-dependency relation between $r_{1,1}(a)$ and $w_{1,2}(b)$ and a function-dependency relation between $r_{1,2}(b)$ and $w_{1,1}(a)$, i.e. $r_{1,1}(a) <_G w_{1,2}(b)$ and $r_{1,2}(b) <_G w_{1,1}(a)$, then a cycle is formed by the two function-dependency relations between $G_{1,1}$ and $G_{1,2}$. That is, the read and write operations of a global transaction are not allowed to form an arbitrary partial order if a globally committable history is required. As far as the structure of computation is concerned, the computation ability of the global transactions in a globally committable history is not so strong as that of conventional multi-site transactions in which all the read and write operations of a transaction form an arbitrary partial order. Fortunately, in most database applications, we believe, the databases in different local sites can be hierarchically integrated and the related global transactions have a natural hierarchical structure based on the integrated database.

The another assumption on global transactions is that the failure atomicity of a global transaction is restricted to be "if one subtransaction commits, then all other subtransactions will eventually commit". The implication of the restriction is that every global transaction must be designed to be globally committable if different subtransactions will not use different values of a data item. Writing a globally committable transaction needs the semantic knowledge of the global transaction itself (by compriion, writing a compensating transaction may need the semantic knowledge of both global and local transactions). So the assumption does not impose any restriction on the local autonomy requirements.

We have formulated the serializability and recoverability notions in the context of our extended definition of transactions and defined a new correctness notion called \textit{global commitment} (GC) for reliable transaction processing in a multidatabase system. The GC set intersects the SR set and the RC set, but is incomparable to them. Global Commitment (GC) is not only a prefix commit-
closed property like SR and RC etc., but also a prefix extension-closed property. That is, if a history $H$ is a GC history, then every global transaction remains to become globally-committed after local transactions are arbitrarily appended to $C(H')$ of any prefix $H'$ of a history $H$. The local extensibility of a execution history ensures that every global transaction in the history can be globally-committed, even in the presence of failures in a MDBS. We finally have given two sufficient conditions for a multidatabase system to schedule a globally committable history. A scheduler that produces a globally committable history can directly be implemented based on the two sufficient conditions to guarantee global transactions to be globally-committed.

In its present form, this paper assumes that there are no replicated data items in different local sites. It is interesting to notice that replicated data may change the function-dependency relations among subtransactions of a global transaction. The extension to our current global commitment theory by incorporating data replications into our database model, we believe, would be a significant contribution.

Acknowledgements

The authors would like to acknowledge the contributions made by Jim Mullen. He has contributed immensely to the content and presentation of this paper.

References


24