A Uniform Approach to Global Concurrency Control and Recovery in Multidatabase Environment

SangKeun Lee
Dept. of Computer Sci. and Eng.
Korea University
lsk@disys.korea.ac.kr

Chong-Sun Hwang
Dept. of Computer Sci. and Eng.
Korea University
hwang@disys.korea.ac.kr

WonGye Lee
Dept. of Computer Science Education
Korea University
lee@comedu.korea.ac.kr

Abstract

In this paper, we provide a uniform approach to global concurrency control and recovery in multidatabase environment. Instead of considering global serializability and global atomicity as two orthogonal concepts, we simply adopt global serializability as the only correctness criterion and require global serializability to be maintained even in a failure-prone multidatabase environment. We first propose *rigid conflict serializability* (R-CSR) as a sufficient condition for the global transaction manager to ensure global serializability in an autonomous, heterogeneous, and failure-free multidatabase environment. Following this, we show that the combination of cascadeless R-CSR of global transactions and a *context-sensitive* and *late redo* recovery leads to the achievement of global serializability in a failure-prone multidatabase environment.

1 Introduction

A *Multidatabase System* (MDBS) [18] is a facility that supports global applications accessing data stored in different databases. It is assumed that access to these databases is controlled by autonomous and heterogeneous *Local Database Systems* (LDBSs). The MDBS permits local transactions and global transactions to coexist. Local transactions are submitted to a single LDBS outside of the MDBS control, while global (sub)transactions interact with both the MDBS and the LDBSs for purposes of local concurrency control and recovery. The *Global Transaction Manger* (GTM) in the MDBS supports the execution of global transactions spanning multiple local databases in the federation. Transaction management mechanisms in MDBSs must ensure *serializability* and *atomicity*, while properly coping with *autonomy* and *heterogeneity* of the participating LDBSs.

CIKM *97 Las Vegas Nevada USA*

Local autonomy is the most fundamental assumption of the MDBS concept and is usually classified into *design* autonomy, *execution* autonomy, and *communication* autonomy [21]. Since autonomy of the participating LDBSs distinguishes MDBSs from traditional distributed database systems, the overriding issue for enforcing global serializability and global atomicity in multidatabase environment has been the preservation of local autonomy.

Local heterogeneity is another important assumption of the MDBS concept. We identify the important heterogeneities to transaction management as dissimilarities in :

- *Concurrency control mechanisms* used by the LDBSs;
- *Commitment protocols* used by the LDBSs; and
- *Recovery mechanisms* used by the LDBSs.

Compared to local autonomy, there has been relatively not much work on the impacts of local heterogeneity on global serializability. Although many approaches [10,11,26] to global serializability have successfully led to the maintenance of global serializability without violation of local autonomy, the impacts of heterogeneity in the participating LDBSs on these approaches have not been analyzed, as we will show in Section 3. Independently of ensuring global serializability, the problem of ensuring global atomicity in an MDBS has also been widely studied in [8,9,15,19,25]. However, global serializability and global atomicity have been considered as two orthogonal concepts, and as a result, the combination of ensuring global serializability and global atomicity has become a difficult and complex work.

In this paper, we simply adopt global serializability as the only correctness criterion, instead of considering global serializability and global atomicity as two orthogonal concepts, and require global serializability to be maintained even in a failure-prone multidatabase environment. We first propose *rigid conflict serializability* (R-CSR) as a sufficient condition for the GTM to ensure global serializability in an autonomous, heterogeneous, and a failure-free multidatabase environment. Subsequently, we derive several conditions for maintaining global serializability in presence of failures. The prin-

ciple is to ensure cascadeless R-CSR of global transactions, which is combined with a *context-sensitive* and *late redo* global recovery scheme.

The organization of this paper is as follows. Section 2 presents the system model, notation, and terminology used throughout the paper. Section 3 characterizes local heterogeneity as *abnormal direct conflicts* among global transactions and analyze their impacts on the previous approaches [7,10,11,26] to global serializability. Section 4 proposes R-CSR as a sufficient condition for the GTM to ensure global serializability in an autonomous, heterogeneous, and failure-free multidatabase environment. In Section 5, we present the ways of ensuring global serializability in a failure-prone multidatabase environment. Conclusion is in Section 6.

2 The System Model, Notation, and Terminology

An MDBS consists of a number of pre-existing LDBSs, located at sites $s_1, s_2, \ldots, s_n$ $(n \geq 2)$, where each $LDBS_i$ $(1 \leq i \leq n)$ is a local database management system. An MDBS supports local transactions and global transactions. Local transactions access data managed by only a single LDBS and are executed under the LDBS. Global transactions are executed under MDBS control. A global transaction consists of a set of global subtransactions, each of which is an ordinary local transaction from the point of view of an LDBS where the subtransaction is executed. We assume that each global transaction generated by the MDBS has at most one subtransaction at each LDBS. We also presumes that concurrency control mechanisms of LDBSs ensure local *serializability* (SR) [5], and each LDBS is responsible for ensuring local *atomicity* [5].

The MDBS software that executes on top of the pre-existing LDBSs consists of a GTM and a set of servers, one associated with each LDBS. Each global transaction submits its operations to the GTM. For each submitted operation, the GTM determines whether to submit the operation to local sites, or to delay it, or to abort the transaction. If the operation is to be submitted, the GTM selects a local site (or a set of sites) where the operation should be executed. The GTM submits global transaction operations to the LDBSs through the server which acts as the liaison between the GTM and the LDBS. Operations belonging to a global subtransaction are submitted to the LDBS by the server as a single transaction. We assume that each LDBS acknowledges to the server (and, in turn, to the GTM) the execution of operations submitted to it. In particular, we do not consider *failures* or *aborts* throughout Section 3 and 4.

For a transaction T_i, there are four basic operations: $r_i(x), w_i(x), c_i$, and a_i, where c_i and a_i are commit and abort termination operations, and r_i and w_i are read and write operations accessing data item x in an LDBS. A transaction T_i that refers to either a local or global transaction, is a partial order of read, write, commit, abort operations which contain exactly one termination operation that is the last element in the partial order. A local transaction L_k is a transaction that accesses data items at a single site s_k. A global transaction G_i consists of a set of global subtransactions G_{ik} $(1 \leq k \leq n)$, where G_{ik} is a global subtransaction accessing an $LDBS_k$. The local schedule at site s_k, denoted by S_k, is a sequence of local and global transactions operations resulting from their execution at site s_k. We denote $o_i <_{S_k} o_j$ if operation o_i is executed before operation o_j in schedule S_k, and denote $T_i \rightarrow T_j$ if transaction T_i is serialized before transaction T_j. We say that transactions T_i and T_j are in *direct conflict* in schedule S_k, if and only if schedule S_k contains operations $o_i(x)$ followed by operation $o_j(x)$, where $o_i(x)$ or $o_j(x)$ are a write operation and T_i does not abort before $o_j(x)$ is executed. We say that T_i and T_j are in *indirect conflict* in schedule S_k if and only if there is a sequence of transaction $T_1, T_2, \ldots, T_m$ $(m \geq 1)$ such that T_i is in direct conflict with T_1, T_1 is in direct conflict with T_2, ..., and finally, T_m is in direct conflict with T_j.

A set $G = \{G_1, G_2, \ldots, G_m\}$ contains those global transactions that are submitted to the MDBS, and G_k denotes the set of global subtransactions of G at local site s_k. A global schedule S is the combination of all local schedules, and a global subschedule S_G is S restricted to the set G of global transactions in S. A global schedule S is *globally serializable* if and only if there is a total order defined over committed global transactions that is consistent with the serialization order of committed global transactions at each LDBS [6].

3 Abnormal Direct Conflicts and Their Impacts on Global Serializability

It is examined in early work [10,11,26] that the maintenance of global serializability can be reduced to synchronizing the relative serialization orders of global subtransactions of each global transaction at all LDBSs, and local indirect conflict is the major cause of difficulty of achieving global serializability. It is difficult, however, to resolve local indirect conflicts at the global level without violation of local autonomy, because the behavior or even the existence of local transactions is not known to the MDBS [10,11]. The previous discussions on resolving local indirect conflicts indicate that the GTM can determine the serialization order of global subtransactions at each LDBS without violation of local autonomy only by forcing direct conflicting operations between global transactions through *ticket* methods [10,11] or *extra operation* methods [26].

In this section, we will analyze the impacts of local heterogeneity on these methods. We first characterize

local heterogeneity as *abnormal direct conflicts* between global transactions, which may not guarantee that the execution order of direct conflicting transactions is identical to the serialization order at some LDBS.

Definition 1. Two transactions T_i and T_j are in **abnormal direct conflict** in schedule S_k if either of the following two conditions is satisfied:
- T_i and T_j are in direct conflict in schedule S_k such that S_k contains operations $w_i(x)$ followed by $r_j(x)$, and T_i does not commit before $r_j(x)$ is executed;
- T_i and T_j are in direct conflict in schedule S_k such that S_k contains operations $w_i(x)$ followed by $w_j(x)$, and T_j has been submitted to S_k before T_i. □

We now analyze the impacts of abnormal direct conflicts on the previous approaches [7,10,11,26] to global serializability.

3.1 Impacts on Chain-Conflicting Serializability

Chain-conflicting serializability [26] provides a sufficient condition for the GTM to synchronize the relative serialization orders of global subtransactions of each global transaction at all LDBSs without violation of local autonomy. This criterion is based on the property of *chain-conflicting transactions* [26]. In an effort to enfoce it, extra operation method is suggested, where the GTM appends direct conflicting operations and controls the execution order of direct conflicting operations of global transactions identically at every LDBS. This is based on the analogy in the execution order of direct conflicting operations and the serialization order of corresponding global subtransactions at every LDBS. However, the analogy may not be guaranteed in the environment where abnormal direct conflicts exist, as illustrated in the following example.

Example 1. Consider an MDBS consisting of two LDBSs, where data item x is in $LDBS_1$, and data item y is in $LDBS_2$. Let G_1 and G_2 be two global transactions defined as follows:

$$G_1 : w_{G1}(x)\ r_{G1}(y) \quad G_2 : r_{G2}(x)\ w_{G2}(y)$$

Let S_1 and S_2 be the global subschedule generated at $LDBS_1$ and $LDBS_2$, respectively:

$$S_1 : w_{G11}(x)\ r_{G21}(x) \quad S_2 : r_{G12}(y)\ w_{G22}(y)$$

Chain-conflicting serializability implies that global serializability is always maintained as long as the execution order of direct conflicting operations of global subtransactions are controlled identically at both LDBSs. Let us assume that an $LDBS_1$ employs an *intentions list* [5] mechanism for local recovery, where the changed state by a transaction is reflected on the database only after the transaction commits. Intentions lists are similar to *shadow page* techniques used for recovery in System R [12] and to the *private workspace* techniques in many optimistic concurrency control algorithms (e.g., [16]). The concurrency control at $LDBS_1$ is possibly constructed such that an issued $r_i(x)$ is always serialized before *active* $w_j(x)$, instead of blocking $r_i(x)$ even though $w_j(x) <_{S1} r_i(x)$. This kind of concurrency control is possible if the concurrency control at $LDBS_1$ uses *recoverability* [4], *invalidation* [3,14], or *preservation* [17] as the basis for determining conflicts, and the serialization order is flexibly determined (e.g., the *history abstraction* model [3,20]). Under this situation, chain-conflicting serializability does not guarantee global serializability anymore. Although the execution orders of abnormal direct conflicting operations of global subtransactions are forced to be identical at both LDBSs such as :

$$w_{G11}(x) <_{S1} r_{G21}(x) \qquad r_{G12}(y) <_{S2} w_{G22}(y)$$

the serialization order of global transactions at $LDBS_1$ can be $G_2 \rightarrow G_1$, while $G_1 \rightarrow G_2$ at $LDBS_2$. □

3.2 Impacts on Sharing Serializability

The fundamental concern in *sharing serializability* [26] is to seek alternative properties of global transactions other than conflicts such that the MDBS can indirectly determine the serialization order of global subtransactions at each LDBS without violation of local autonomy. This criterion on based on the property of *fully-sharing transactions* [26]. Fully-sharing relationship of transactions is defined with respect to all data accessed by those transactions, irrespective of types of operations. It is argued that the execution order of sharing operations of transactions can also determine the serializaion order of the transactions. Example 1, however, illustrates the fact that the execution order of sharing operations of global transactions that is identical to the order of fully sharing property of global subtransactions may not be identical to the serialization order in some LDBS where abnormal direct conflicts exist. Let us consider the following additional example.

Example 2. Consider an MDBS consisting of two LDBSs, where data item x is in $LDBS_1$, and data items y and z are in $LDBS_2$. Let us assume an $LDBS_2$ uses timestamp ordering mechanism for local concurrency control that applies *Thomas' Write Rule* (TWR) [5] with respect to write-write direct conflicts. Let G_1 and G_2 be two global transactions defined as follows:

$G_1 : w_{G1}(x)\ w_{G1}(z)\ c_{G1}$

$G_2 : w_{G2}(x)\ r_{G2}(y)\ w_{G2}(z)\ c_{G2}$

Let S_1 and S_2 be the global subschedule generated at $LDBS_1$ and $LDBS_2$, respectively:

$S_1 : w_{G11}(x)\ c_{G11}\ w_{G21}(x)\ c_{G21}$

$S_2 : r_{G22}(y)\ w_{G12}(z)\ c_{G12}\ w_{G22}(z)\ c_{G22}$

The scenario at $LDBS_2$ is as follows. First, G_{22} has a smaller timestamp than G_{12} since G_{22} has been submitted before G_{12}. Following this, when a TWR write-write synchronizer receives the $w_{G22}(z)$ that has arrived too late insofar as the timestamp ordering rule is concerned, it simply *ignores* (i.e., does not send it to the data manager [5]) but reports its successful completion to the server being responsible for the execution of G_{22}. As a result, although the execution orders of abnormal direct conflicting operations of global subtransactions are forced to be identical at both LDBSs such as:

$$w_{G11}(x) <_{S1} w_{G21}(x) \qquad w_{G12}(z) <_{S2} w_{G22}(z)$$

the serialization order of global transactions at $LDBS_2$ can be $G_2 \rightarrow G_1$, while $G_1 \rightarrow G_2$ at $LDBS_1$. □

3.3 Impacts on Rigorous Transaction Scheduling Approaches

It has been argued that in [7] that global serializability is assured in a multidatabase system if each global transaction is *commit-deferred* [7] and every LDBS generates only a *rigorous* [7] schedule. Assuming rigorousness of the participating LDBSs, it has also been argued that *Implicit Ticket Method* (ITM) [10,11] achieves global serializability by controlling the commitment order of global subtransactions. However, *rigorous timestamping ordering* that applies TWR with respect to the write-write abnormal direct conflicts may not guarantee that the relative serialization order of each subtransaction is determined by its commitment order. Consider Example 2 again. Although S_1 and S_2 are rigorous, G_1 and G_2 are commit-deferred, and commitment order is controlled such that $c_{G1} <_S c_{G2}$, the serialization order at $LDBS_2$ can be $G_2 \rightarrow G_1$, while $G_1 \rightarrow G_2$ at $LDBS_1$.

3.4 Impacts on Optimization of Ticket Operations

The *Optimistic Ticket Method* (OTM) [10,11] and the *Conservative Ticket Method* (CTM) [11] force direct conflicts among global transactions by ticket operations. The OTM uses the ticket value as the relative serialization order of a global subtransaction in a local site without violation of local autonomy. The ticket values read by ticket operations at one local site can elegantly determine the relative serialization order of corresponding global subtransactions at a local site even when abnormal direct conflicts exist. A possible optimization [10,11] has also been suggested, saying that there is no need for global transactions to take tickets at a local site if all global transactions conflict directly at the local site. It is not sufficient, however, to observe the order to determine their relative serialization order at the local site, as illustrated in Example 1 and 2.

4 Rigid Conflict Serializability

We have showed in Section 3 that chain-conflicting serializability and sharing serializability may not guarantee global serializability in multidatabase environment mainly due to the existence of abnormal direct conflicts. In this section, we first identify *rigid direct conflicts*. Then, we provide *rigid conflict serializability* (R-CSR), which is used for the GTM to enforce global serializability in an autonomous, heterogeneous, and failure-free MDBS environment.

4.1 Rigid Direct Conflicts

Based on the principles of weaker conflict relations [3,4,13,14,17,24] and a variety of concurrency control and recovery implementations [1,2,5,12,17,23], we identify *rigid direct conflicts* between global transactions, which guarantee that the execution order of involved transactions is *necessarily* identical to the serialization order of them, with tolerance of local heterogeneity as well as without violation of local autonomy.

Definition 2. Two transactions T_i and T_j are in **rigid direct conflict** in schedule S_k if either of the following three conditions is satisfied:

- (Read-Write) T_i and T_j are in direct conflict in schedule S_k such that S_k contains operations $r_i(x)$ followed by $w_j(x)$, i.e., $r_i(x) <_{Sk} w_j(x)$;
- (Write-Read) T_i and T_j are in direct conflict in schedule S_k such that S_k contains operations $w_i(x)$ followed by $r_j(x)$, and T_i commits before r_j is executed, i.e., $w_i(x) <_{Sk} c_i <_{Sk} r_j(x)$;
- (Write-Write) T_i and T_j are in direct conflict in schedule S_k such that S_k contains operations $w_i(x)$ followed by $w_j(x)$, and T_i completes its first operation in S_k before the submission of T_j's first operation. □

The basic rationale of rigid direct conflicts is that no schedule generated by any combination of weaker conflict relations and a variety of concurrency control implementations allows the situation where the execution order of two rigid direct conflicting transactions is different from the serialization order of the two transactions.

4.2 Rigid Conflict Serializability

Rigid conflict serializability (R-CSR), which is modified from chain-conflicting serializability[26], requires that every global transaction accessing more than two LDBSs be in rigid direct conflicts at the accessed LDBSs in order for the GTM to determine the relative serialization order at the LDBSs.

We first define rigid conflict transactions in the following Definition 3.

Definition 3. A set of transactions, $T = \{T_1, T_2, \ldots, T_m\}$, is **rigid conflicting** if there is a total order

$T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_m$ on T such that T_1 is in rigid direct conflict with T_2, T_2 is in rigid direct conflict with T_3,..., and finally, T_{m-1} is in rigid direct conflict with T_m. A set of global transactions, $G = \{G_1, G_2, \ldots, G_m\}$, is rigid conflicting if there is a total order O on G such that for all s_k $(1 \leq k \leq n)$, G_k is rigid conflict in an order consistent with O. □

Definition 4. A gobal subschedule S_G is **rigid conflict serializable** if and only if the set G of committed transaction in S_G is rigid conflicting in a total order O on G, and S_G is serializable in O. □

R-CSR provides a sufficient condition for global serializability, which is used for the GTM to maintain global serializability in an autonomous, heterogeneous, and failure-free multidatabase environment. Note that R-CSR guarantees global serializability only provided that every LDBS generates locally serializable execution. This is shown in the following Theorem 1. We do not provide the proof due to the page limit.

Theorem 1. Let S be a global schedule and G be the set of global transactions in S. If S_G is rigid conflict serializable and every S_k at $LDBS_k$ $(1 \leq k \leq n)$ is locally serializable, then S is globally serializable. □

In order to enforce R-CSR, we suggest a *rigid method* where the GTM forces rigid direct conflicts on the *ticket* data item located in each local site when each global subtransaction begins its execution in a *conservative* manner that the relative serialization order of global subtransactions is the same in all participating LDBSs. Determining R-CSR order at the beginning of each global subtransaction has a desirable feature with respect to an effecive global recovery, as will be described in the following section.

5 Rigid Conflict Serializability in Failure-Prone MDBSs

In MDBS environment, a global transaction at a local site can be aborted as a result of normal database management systems operations (such as aborts caused by a local deadlock detection procedure), while the same transaction can be committed at some other local sites. Multidatabase recovery procedures should ensure that the GTM can recover from these aborts and failures alike. In this paper, we consider such situations as *global transaction failures* and any global transaction failure is reported to the GTM. If a global transaction fails before it commits, then any of its local subtransactions must be undone by the appropriate LDBSs. As a consequence, global database consistency is also preserved since the transaction did not make changes in any of its local databases. Thus, the MDBS does not need to use the *undo* operation to restore a multidatabase to a consistent database state. The situation becomes more complicated if a failure occurs during the processing of a *commit* operation of a global transaction. The difficulty in achieving atomicity of global transactions is caused by the fact that many pre-existing LDBSs do not support *prepared states* for implementing *atomic commitment protocols* such as *two-phase commit* (2PC) protocol [5].

In this section, we show how database consistency can be preserved by forcing R-CSR in presence of global transaction aborts and several types of failures. In particular, we suggest a *context-sensitive* and *late redo* global recovery scheme which effectively relaxes the strictness requirement of global transactions imposed by many redo approaches [6,8,9,15,19,25] to global atomicity.

5.1 The Recovery Model and Restrictions

To achieve atomic commitment protocol, an MDBS uses the 2PC protocol where the GTM acts as the coordinator and the associated servers at local sites act as the participants. Since we assume that LDBSs do not support a prepared-to-commit state, the global transaction may be aborted at some local site at any time, even after the server has voted to commit the transaction. In particular, a global subtransaction is said to be *unilaterally aborted* [15] if an LDBS aborts the subtransaction that the GTM has decided to commit. For the sake of simplicity, we also say that the global transaction containing at lease one unilaterally aborted subtransactin is unilaterally aborted. If a global subtransaction is unilaterally aborted, the GTM must take redo recovery actions to ensure its updates are reflected on the database. The GTM achieves this by resubmitting a redo transaction for the unilaterally aborted subtransaction until it is committed at the local site. To construct such a redo transaction, the server must maintain a *server log* in which it logs the updates of global subtransactions. In case of failure of the redo transaction, it is repeatedly resubmitted by the server until it commits.

The previous approaches [8,15,19] have required the following conditions in order to achieve the atomicity of global transactions :

- *Restrictions imposed on the execution of local transactions* : the schedule produced by each participating LDBS is *cascadeless* [1];
- *Restrictions imposed on the data items accessed by transactions* : the data set is partitioned into *globally updateable* and *locally updateable* subsets[2], and a global update subtransaction is disallowed to read any locally updateable data.

[1] This restriction is identified as *M-recoverability* in [19]

[2] *Globally updateable* data items are those that can only be modified by global transactions, and *locally updateable* data items are those that can only be modified by local transactions [8]

We assume that the execution of local transactions and the data items accessed by transactions are restricted as above in the remainder of this paper. Additionally, it is assumed that the GTM forces R-CSR among global transactions by applying the rigid method.

5.2 Scheduling of Global Commit Operations

Under the recovery model and restrictions presented in section 5.1, the task of ensuring global serializability in presence of failures can reduce to preventing the situations where R-CSR between a unilaterally aborted global transaction and other global transactions is violated by executing a redo transaction of the unilaterally aborted global transaction.

Note that R-CSR between any two global transactions G_i and G_j necessarily involves one of three types of rigid direct conflicts at each local site accessed by the both transactions : Read-Write, Write-Read, or Write-Write rigid direct conflict. If R-CSR between G_i and G_j involves Write-Read rigid direct conflict at all local sites accessed by the two transactions, R-CSR can be maintained even if any transaction of the two ones would be unilaterally aborted. That's because Write-Read rigid direct conflict requires read operation to be executed only after write operation successfully commits at global level. In contrast, care must be taken if R-CSR between G_i and G_j involves at least one Read-Write or Write-Write rigid direct conflict at any local site, as illustrated in the following Example 3.

Example 3. Consider the global subschedule S_1 and S_2 generated at $LDBS_1$ and $LDBS_2$ respectively, where data item x, y, and z in $LDBS_1$ are globally updateable and a data item v in $LDBS_2$ are also globally updateable (ua is used to denote a unilateral abort step, L_1 is a local transaction at $LDBS_1$, and T_3 is a redo transaction for G_1).

$S_1 : r_{G11}(t_1)\ r_{G11}(x)\ w_{G11}(y)\ w_{G21}(t_1)\ r_{G21}(x)\ w_{G21}(z)$
$ua_{G11}\ c_{G21}\ r_{L1}(z)\ r_{L1}(y)\ c_{L1}\ w_{T3}(y)\ c_{T3}$
$S_2 : w_{G12}(t_2)\ w_{G12}(v)\ c_{G12}\ r_{G22}(t_2)\ w_{G22}(v)\ c_{G22}$

Although the GTM keeps R-CSR between two global transactions G_1 and G_2 in the order $G_1 \rightarrow G_2$ at both sites through Read-Write and Write-Read rigid direct conflicts, the unilateral abortion of G_{11} and its redo transaction T_3 cause the serialization order to be $G_2 \rightarrow G_1$ at S_1, thus resulting in a globally nonserializable schedule. □

The above situation demonstrates that the GTM should use some criterion on scheduling global commit operations in order to ensure global serializability in presence of failures. We present below a crieterion on scheduling of global commit operations.

- *Criterion on Global Commitment :* The GTM schedules the global commit operations of any two global transactions, including redo transactions, such that the commitment order is consistent with the R-CSR order for any site at which both transactions were executing.

The above criterion on scheduling of global commit operations enables the GTM to enforce R-CSR by *aborting* all other global subtransactions succeeding the unilaterally aborted one in the R-CSR order before performing a redo transaction. After the abort step, the GTM applies the rigid method to each redo transaction.

Example 3. (Revisited) Consider again the subschedule S_1 where now the GTM follows the criterion on scheduling global commit operations, aborts G_{21}, and applies the rigid method to T_3.

$S_1 : r_{G11}(t_1)\ r_{G11}(x)\ w_{G11}(y)\ w_{G21}(t_1)\ r_{G21}(x)\ w_{G21}(z)$
$ua_{G11}\ a_{G21}\ r_{T3}(t_1)\ w_{T3}(y)\ c_{T3}\ r_{L1}(z)\ r_{L1}(y)\ c_{L1}$
$S_2 : w_{G12}(t_2)\ w_{G12}(v)\ c_{G12}\ r_{G22}(t_2)\ w_{G22}(v)\ a_{G22}$

which maintains global database consistency. □

Although many previous approaches [6,8,15,19,25] to redo techniques have required *strictness* [5] of schedules at global level in order to preserve global database consistency, the strictness requirement is relaxed at the expense of aborting concurrent global transactions.

The basic advantage of aborting concurrent global transactions with the criterion on global commitment is that it relaxes the strictness imposed on the global execution of global transactions. Its main disadvantage is that global transactions suffer frequently from *global restarts* caused by the abortion step. In the following section we suggest a novel recovery that addresses the issue.

5.3 A Context-Sensitive and Late Redo Recovery

In this section we suggest a novel global recovery, called a *context-sensitive* and *late redo* recovery. The suggested recovery scheme can, to some extent, abbreviate the abortion step necessary to maintain global serializability in presence of failures.

A global recovery for a unilaterally aborted $w_i(x)$ is performed in a *context-sensitive* manner such that :

- *Context-Sensitive Recovery Rule (in case of unilateral aborts) :* A global redo for $w_i(x)$ is a *null* operation, denoted by $\lambda_i(x)$, if there exists some write operation $w_j(x)$ satisfying $w_i(x) <_{SG} w_j(x)$. Otherwise, it is $w_i(x)$.

The semantics of the context-sensitive redo recovery for $w_i(x)$ is based on the resulting schedule stored in a server log up to the time a unilateral abort happens. Note that the context-sensitive recovery rule utilizes both the local atomicity property in all the participating LDBSs and the R-CSR order determined at the beginning of each global subtransaction.

If a redo transaction of a unilaterally aborted global subtransaction is composed of a set of *null* operations

only, the GTM can safely assume that it is successfully completed, and if not, the GTM aborts all the concurrent global subtransactions and performs a redo transaction for a unilaterally aborted one. To demonstate the effects of the context-sensitive redo recovery, consider the following schedule S_1 (data item x and y are globally updateable) :

$S_1 : r_{G11}(t_1)\ w_{G11}(x)\ w_{G21}(t_1)\ r_{G21}(y)\ w_{G21}(x)\ ua_{G11}$ $\boxed{\lambda_{G11}(x)}\ c_{G21}$

which maintains global database consistency without performing the step to abort G_{21}.

The semantics of the context-sensitive recovery, however, may violate the global database consistency if an active $w_j(x)$ succeeding $w_i(x)$ is *nonunilaterally* aborted at the local site after the context-sensitive recovery has been performed. Consider the following schedule S_1 :

$S_1 : r_{G11}(t_1)\ w_{G11}(x)\ w_{G21}(t_1)\ r_{G21}(y)\ w_{G21}(x)\ ua_{G11}$ $\boxed{\lambda_{G11}(x)}\ a_{G21}$

which violates global database consistency since the value of x written by *committed* G_{11} is not reflected on the database. This shows that the semantics of the context-sensitive redo recovery alone is not sufficient to maintain global database consistency. To maintain global database consistency a global *late redo* recovery for a unilaterally aborted $w_i(x)$ should be employed together with the context-sensitive one in such a manner that :

- *Late Redo Recovery Rule (in case of nonunilateral aborts) :* A global late redo for $w_i(x)$ is $w_i(x)$ if there does not exist some active $w_j(x)$ satisfying $w_i(x) <_{SG} w_j(x)$ when a global subtransaction containing $w_k(x)$ is nonunilaterally aborted. Otherwise, it is $\lambda_i(x)$.

Consider again the above schedule S_1 (T_3 is a redo transaction for G_1) where now the late redo recovery rule is applied:

$S_1 : r_{G11}(t_1)\ w_{G11}(x)\ w_{G21}(t_1)\ r_{G21}(y)\ w_{G21}(x)\ ua_{G11}$ $\boxed{\lambda_{G11}(x)}\ a_{G21}\ \boxed{r_{T3}(t_1)\ w_{T3}(x)\ c_{T3}}$

which maintains global database consistency, possibly by aborting all the concurrent global subtransactions at S_1.

5.4 Enforcing Rigid Conflict Serializability

Although the strictness requirement imposed on the global execution of global transactions in the literature is effectively relaxed through the criterion on global commitment and the context-sensitive and late redo recovery, it is necessary that the global execution of global transactions be *cascadeless* if no additional mechanism is employed at global level. Consider the following globally noncascadeless schedule S_1 where a data item x is globally updateable :

$S_1 : r_{G11}(t_1)\ w_{G11}(x)\ ua_{G11}\ w_{G21}(t_1)\ r_{G21}(x)\ w_{G21}(x)$ $\boxed{\lambda_{G11}(x)}\ c_{G21}$

which violates the global database consistency since the value of x written by *committed* G_{11} is not reflected on G_{21}. Note that $r_{G21}(x)$ can be executed at cascadeless $LDBS_1$ since G_{11} is aborted from the view of $LDBS_1$, and G_{11} is assumed to be successfully completed from the GTM viewpoint by the context-sensitive recovery rule. This undesirable situation can be avoided if the GTM controls the global exection of global transactions such that it generates only cascadeless schedule at global level.

The following Theorem 2 shows global database consistency can be preserved in presence of failures. We do not provide the proof due to the page limit.

Theorem 2. A global schedule S is globally serializable in presence of failures if the following conditions are satisfied :

1. The GTM controls the execution of global transactins, including redo transactions, such that it generates cascadeless and rigid conflict serializable schedules by the rigid method;
2. The GTM schedules global commit operations according to the criterion on global commitment;
3. A global redo procedure follows the context-sensitive and late redo recovery rules;
4. Each S_k at $LDBS_k$ $(1 \leq k \leq n)$ is locally cascadeless and serializable;
5. Each $LDBS_k$ $(1 \leq k \leq n)$ is responsible for ensuring local atomicity;
6. The data set is partitioned into globally updateable and locally updateable subsets, and a global update subtransaction is disallowed to read any locally updateable data. □

6 Conclusion

We have provided a uniform approach to global concurrency control and recovery in multidatabase environment. For the end, we have simply adopted global serializability as the only correctness criterion and have required global serializability to be maintained in a failure-prone environment.

With respect to global serializability in a failure-free multidatabase environment, we have identified abnormal direct conflicts among global transactions and have shown that *rigid conflict serializability* proposed in this paper notably resolves abnormal direct conflicts. With respect to global serializability in a failure-prone multidatabase environment, we have presented a criterion on global commitment and a context-sensitive and late redo recovery scheme. In particular, the context-sensitve and late redo recovery relaxes the strictness restriction imposed on global execution of global transactions in an effective manner.

References

[1] Agrawal, D., and Abbadi, A. "Locks with Constrained Sharing," *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, 1990

[2] Alonso, G., Vingralek, R., Agrawal, D., Breitbart, Y., Abbadi, A., Schek, H., and Weikum, G. "A Unified Approach to Concurrency Control and Transaction Recovery," *Information Systems*, Vol.19, No.1, 1994

[3] Anastassopoulos, P., and Jean Dollimore. "A Unified Approach to Distributed Concurrency Control," In *Distributed Computing Systems*. Thomas L., Casavant, and Mukesh Singhal, IEEE Computer Society Press, Califonia, 1994

[4] Badrinath, B., and Ramamritham, K. "Semantics-Based Concurrency Control : Beyond Commutativity," *ACM Transactions on Database Systems*, Vol.17, No.1, 1992

[5] Bernstein, P. A., Hadzilacos, V., and Goodman, N. *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Reading, Mass., 1987

[6] Breitbart, Y., Garcia-Molina, H., and Silberschatz, A. "Overview of Multidatabase Transaction Managemnent," *VLDB Journal*, Vol.1, No.2, 1992

[7] Breitbart, Y., Georgakopoulos, D., Rusinkiewicz, M., and Silberschatz, A. "On Rigorous Transaction Scheduling," *IEEE Transactions on Software Engineering*, Vol.17, No.9, 1991

[8] Breitbart, Y., Silberschatz, A., and Thompson, G. "Transaction Management Issues in a Failure-Prone Multidatabase System Environment," *VLDB Journal*, Vol.1, No.1, 1992

[9] Georgakopoulos, D. "Multidatabase Recoverability and Recovery," *Proceedings of the 1st International Workshop on Interoperability Multidatabase Systems*, 1991

[10] Georgakopoulos, D., Rusinkiewicz, M., and Sheth, A. "On Serializability of Multidatabase Transactions through Forced Local Conflicts," *Proceedings of the 7th International Conference on Data Engineering*, 1991

[11] Georgakopoulos, D., Rusinkiewicz, M., and Sheth, A. "Using Tickets to Enforce the Serializability of Multidatabase Transactions," *IEEE Transactions on Knowledge and Data Engineering*, Vol.6, No.1, 1994

[12] Gray, J., Lorie, R., Putzulo, A., and Traiger, J. "The Recovery Manager of the System R Database Manager," *ACM Computing Surveys*, Vol.13, No.2, 1981

[13] Guerni, M., Ferrie, J., and Pons, J. "Concurrency Control and Recovery for Typed Objects using a New Commutativity Relation," *Deductive and Object-Oriented Databases*. Lecture Notes in Computer Science 1013. Ling, T. W. et al., 1995

[14] Herlihy, M. "Apologizing versus Asking Permission: Optimistic Concurrency Control for Abstract Data Types," *ACM Transactions on Database Systems*, Vol.15, No.1, 1990

[15] Kang, I., and Keefe, T. "Supporting Reliable and Atomic Transaction Management in Multidatabase Systems," *Proceedings of the 13th Internaional Conference on Distributed Computing Systems*, 1993

[16] Kung, H., and Robinson, J. "On Optimistic Methods for Concurrency Control," *ACM Transactions on Database Systems*, Vol.6, No.2, 1981

[17] Lee, S., Jung, S, and Hwang, C. "A New Conflict Relation for Concurrency Control and Recovery in Object-Based Databases," *Proceedings of the 5th International Conference on Information and Knowledge Management*, 1996

[18] Litwin, W. "From Database Systems to Multidatabase Systems: Why and How," *British National Conference on Databases*, Cambridge Press, 1988

[19] Mehrotra, S., Rastogi, R., Breitbart, Y., Korth, H., and Silberschatz, A. "Ensuring Transaction Atomicity in Multidatabase Systems," *Proceedings of the 11th ACM Symposium on Principles of Database Systems*, 1992

[20] Ng, T. "Using Histories to Implement Atomic Objects," *ACM Transactions on Computer Systems*, Vol.17, No.4, 1989

[21] Veijalainen, J. *Transaction Concepts in Autonomous Databases Environments*, R. Oldenbourg Verlag, 1990

[22] Veijalainen, J., and Wolski, A. "Prepared and Commit Certification for Decentralized Transaction Management in Rigorous Heterogeneous Multidatabases," *Proceedings of the 8th International Conference on Data Engineering*, 1992

[23] Vingralek, R., Ye, H., Breitbart, Y., and Schek, H. "Unified Transaction Model for Semantically Rich Operations," *Proceedings of the 5th International Conference on Database Theory*, 1995

[24] Weihl, W. "Commutativity-Based Concurrency Control for Abstract Data Types," *IEEE Transactions on Computer Systems*, Vol.37, No.12, 1988

[25] Wolski, A., and Veijalainen, J. "2PC Agent Method: Achieving Serializability in Presence of Failures in a Heterogeneous Multidatabase," *Proceedings of the PARBASE-90 Conference*, 1990

[26] Zhang, A., and Elmagarmid, A. "A Theory of Global Concurrency Control in Multidatabase Systems," *VLDB Journal*, Vol.2, No.3, 1993