1993

Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems

Aidong Zhang

Marian H. Nodine

Omran Bukhres

Report Number:
93-069

ENSURING RELAXED ATOMICITY FOR FLEXIBLE TRANSACTIONS IN MULTIDATABASE SYSTEMS

Aidong Zhang
Marian H. Nodine
Omran Bukhres

CSD-TR-93-069
November 1993

Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems

Aidong Zhang* Marian H. Nodine* Omran Bukhres*

*Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 USA
{azhang, bukhres}@cs.purdue.edu

*Department of Computer Sciences
Brown University
Providence, RI 02912 USA
mhn@cs.brown.edu

Abstract

Global transaction management and the preservation of local autonomy are conflicting requirements in the design of multidatabase transaction management systems. A flexible transaction model for the specification of global transactions makes it possible to construct robust global transactions while still preserving local autonomy. This paper presents an approach that preserves *semi-atomicity*, a weaker form of atomicity applicable to flexible transactions which span several local database systems that maintain serializability and recoverability. We first offer a fundamental characterization of the flexible transaction model and then precisely define semi-atomicity for flexible transactions. We then investigate the principles underlying the approach to flexible transaction management which ensures this property. Finally, we construct a class of flexible transactions which can be executed in the presence of failures using our proposed commit protocol. The results demonstrate that the flexible transaction model substantially enhances the scope of global transaction management beyond that offered by the traditional global transaction model.

1 Introduction

A multidatabase system (MDBS) is a collection of autonomous local databases (LDBSs) that can be accessed as a single unit. There are two types of transactions in a multidatabase. A local transaction, which accesses a local database only, is submitted directly to a local database system. A global transaction, in contrast, may access several local databases. Such a global transaction is submitted to a global transaction manager (GTM) superimposed upon a set of local autonomous database systems, where it is parsed into a series of global subtransactions to be submitted to the local database systems.

While atomicity in traditional distributed databases can be ensured using well-known protocols [4], multidatabase systems cannot use these protocols directly because of the stronger autonomy requirements of the component local databases [3, 21]. Of particular concern is the fact that multidatabases cannot assume that all participating local databases support a visible prepare-to-commit state for global subtransactions, in which they have not yet been committed but are guaranteed the ability to commit. In such situations, a local database system that participates in a multidatabase environment may unilaterally abort a global subtransaction without agreement from the global level (termed *a local unilateral abort*). As a result, it becomes difficult to ensure that a single logical commit action of the subtransactions of a global transaction will be consistently carried out at multiple local sites. In addition, even when local database systems do provide such support, the potential blocking and long delays caused by prepare-to-commit states would severely degrade local execution autonomy.

A flurry of research activity has been devoted to the problems of enhancing transaction management by using extended transaction models.[1] In particular, some flexible transaction models proposed for the MDBS environment, such as Flex Transactions [8] and S-transaction [22], increase the failure resiliency of global transactions by allowing alternative subtransactions to be executed when a local database fails or a subtransaction aborts. This flexibility allows a global transaction to adhere to a weaker form of atomicity, which we term *semi-atomicity*, while still maintaining its correct execution in the multidatabase. Semi-atomicity allows a global transaction to commit even if some subtransactions abort, provided that their alternative subtransactions commit. The following example is illustrative:

Example 1 *Consider the following global transaction. A user of an Automatic Teller Machine (ATM) wishes to withdraw \$50 from his savings account a_1 in bank b_1 and to obtain the money in cash from the ATM. If the ATM fails to disburse the cash, he will instead transfer the \$50 to his checking account a_2 in bank b_2. With Flex Transactions, this is represented by the following set of subtransactions:*

t_1: *Withdraw \$50 from savings account a_1 in bank b_1.*

t_2: *Disburse \$50 from the ATM.*

t_3: *Transfer \$50 to checking account a_2 in bank b_2.*

In this global transaction, either $\{t_1, t_2\}$ or $\{t_1, t_3\}$ is acceptable, with $\{t_1, t_2\}$ preferred. If t_2 fails, t_3 may replace t_2. The entire global transaction thus may not have to be aborted, even if t_2 fails, but if both t_2 and t_3 fail, then it must abort. □

[1]Many such models have appeared in [6, 7].

Since the flexible transaction model was proposed, much research has been devoted to its application [13, 2, 1, 12]. Most of this work has assumed the availability of visible prepare-to-commit states in local database systems. In such a scenario, the preservation of the semi-atomicity of flexible transactions is relatively straightforward.

1.1 Proposed Research

In this paper, we offer a precise definition of the fundamental model and of the semi-atomicity property of flexible transactions. We present an approach which preserves semi-atomicity in an MDBS environment in which the local database systems are required only to ensure serializability and recoverability [4]. In the proposed formulation, a flexible transaction is defined as a set of subtransactions upon which a set of partial orders is specified. Each partial order provides one alternative for the successful execution of the flexible transaction. This methodology differs from previous approaches in that no specific application semantics are involved. Therefore, a theoretical basis for flexible transaction management can be built. We then classify the set of flexible transactions that can be executed in an error-prone MDBS environment. As the compensation and retry approaches [15] are unified and employed as flexible transaction failure recovery techniques, local prepare-to-commit states are no longer required. We demonstrate that the flexible transaction model substantially enhances the scope of global transaction management beyond that offered by the traditional global transaction model.

1.2 Related Research

In order to handle local unilateral aborts, approaches using forward recovery (redo and retry) and backward recovery (compensation) have been proposed in the literature. These approaches seek to ensure the semantic atomicity [9] of global transactions in MDBSs. When a subtransaction of a global transaction aborts, the GTM may either re-execute it until commitment or undo the effects of the committed subtransactions of the global transaction. The strategies characterizing these approaches can be classified by the relative timing of the commitment of subtransactions in the local databases with respect to the global transaction commit/abort decision [16]. [23, 5] enforce a global decision on the subtransactions by redoing or retrying them as necessary. [19, 14, 18] commit subtransactions locally before a global decision is made and rely on compensation when a global transaction is aborted. [15, 17] combine these two approaches. With the forward approach, all subtransactions must be redoable or retriable, while the backward approach requires that all subtransactions must be compensatable. With the combined approach, only one subtransaction of each global transaction can be neither retriable nor compensatable, and the rest of its subtransactions must be either retriable or compensatable. Consequently, the ability to specify global transactions becomes severely limited when the traditional global transaction model is employed in MDBSs.

1.3 Structure of the Paper

This paper is organized as follows. Section 2 introduces the fundamental flexible transaction model and Section 3 defines the property of semi-atomicity. In Section 4, we define those flexible transactions that can be executed in the error-prone MDBS environment without requiring local prepare-to-commit states. In Section 5, we present the flexible transaction recovery protocol and demonstrate its effectiveness in preserving the semi-atomicity of flexible transactions. Concluding remarks are offered in Section 6.

2 A Formal Model of Flexible Transactions

Following [8, 13], the definition of flexible transactions takes the form of a high-level applications description. Various applications semantics, such as commit dependencies, abort dependencies, and the acceptable set of successful subtransactions, are captured in the flexible transaction definition. Unfortunately, such a semantics-oriented formulation of flexible transactions may not prevent redundancy in the dependency specification, and the structure of flexible transactions cannot generally be effectively depicted. Delineating a generic structure for flexible transactions is thus necessary for the discussion of flexible transaction management. In this section, a fundamental flexible transaction model that specifies global transactions is precisely defined.

From a user's point of view, a transaction is a sequence of actions on data items in a database. In an MDBS environment, a global transaction is a set of subtransactions, where each subtransaction is a transaction accessing the data items at a single local site. We assume that each global transaction has at most one subtransaction at each local site.[2]

The flexible transaction model supports flexible execution control flow by specifying two types of dependencies among the subtransactions of a global transaction: (1) execution ordering dependencies between two subtransactions, and (2) alternative dependencies between two subsets of subtransactions. Below, we shall formally delineate the flexible execution control flow in the flexible transaction model.

Let $\mathcal{T} = \{t_1, t_2, ..., t_n\}$ be a repertoire of subtransactions and $\mathcal{P}(\mathcal{T})$ the collection of all subsets of $\mathcal{T}$. Let $t_i, t_j \in \mathcal{T}$ and $T_i, T_j \in \mathcal{P}(\mathcal{T})$. We assume two types of control flow relations to be defined on the subsets of $\mathcal{T}$ and on $\mathcal{P}(\mathcal{T})$, respectively: (1) **(precedence)** $t_i \prec t_j$ if t_i precedes t_j $(i \neq j)$; and (2) **(preference)** $T_i \rhd T_j$ if T_i is preferred to T_j $(i \neq j)$. If $T_i \rhd T_j$, we also say that T_j is an alternative to (or contingent on) T_i.[3] Note that T_i and T_j may not be disjoint. Both precedence and preference relations are of an irreflexive transitive nature. In other words, for each $t_i \in \mathcal{T}$,

[2]This is necessary for the concurrency control of global transactions [10].

[3]In general, the alternative relationship need not exist only between two individual subtransactions; one subtransaction may be a semantic alternative of several subtransactions.

$\neg(t_i \prec t_i)$; and for each $T_i \in \mathcal{P}(\mathcal{T})$, $\neg(T_i \rhd T_i)$. If $t_i \prec t_j$ and $t_j \prec t_k$, then $t_i \prec t_k$; if $T_i \rhd T_j$ and $T_j \rhd T_k$, then $T_i \rhd T_k$.

Thus, the precedence relation defines the correct parallel and sequential execution ordering dependencies among the subtransactions, while the preference relation defines the priority dependencies among alternative sets of subtransactions for selection in completing the execution of $\mathcal{T}$.

A flexible transaction can be defined as follows:

Definition 1 (Flexible transaction) *A flexible transaction $\mathcal{T}$ is a set of related subtransactions on which the relations of precedence ($\prec$) and preference ($\rhd$) are defined.*

As this basic definition of flexible transactions provides only a vague picture of the structure of flexible transactions, we shall now seek a more precise delineation. Let T_i be a subset of $\mathcal{T}$, with $\prec$ defined on T_i. We then say that $(T_i, \prec)$ is a partial order of subtransactions. $(T_i, \prec)$ is a *singular partial order*, abbreviated as $\prec$-partial order, if the execution of subtransactions in T_i represents the execution of the entire flexible transaction $\mathcal{T}$. To accommodate the assumption that each global transaction has at most one subtransaction at a local site, we require that each $\prec$-partial order of a flexible transaction have at most one subtransaction at a local site. The whole flexible transaction may contain more than one subtransaction at a local site.

The structure of a flexible transaction $\mathcal{T}$ can thus basically be depicted as a set of $\prec$-partial orders $\{(T_i, \prec), i = 1, ..., k\}$ of subtransactions, with $\bigcup_{i=1}^{k} T_i = \mathcal{T}$.[4] Let $(T, \prec)$ be a $\prec$-partial order of $\mathcal{T}$. A partial order $(T', \prec)$ is a prefix of $(T, \prec)$, denoted $(T', \prec) \leq (T, \prec)$, if (1) $T' \subseteq T$; (2) for all $t_1, t_2 \in T', t_1 \prec t_2$ in $(T', \prec)$ if and only if $t_1 \prec t_2$ in $(T, \prec)$; and (3) for each $t \in T'$, all predecessors of t in T are also in T'. A partial order $(T', \prec)$ is the prefix of $(T, \prec)$ with respect to subtransaction t, denoted $(T', \prec) \leq (T, \prec)(t)$, if $(T', \prec)$ is a prefix of $(T, \prec)$ and T' contains only all predecessors of t in T. A partial order $(T', \prec)$ is the suffix of of $(T, \prec)$ with respect to subtransaction t, denoted $(T', \prec) \geq (T, \prec)(t)$, if, for all $t_1, t_2 \in T', t_1 \prec t_2$ in $(T', \prec)$ if and only if $t_1 \prec t_2$ in $(T, \prec)$ and T' contains only t and all successors of t in T. A set of subtransactions $\{t_1, ..., t_k\}$ in $\prec$-partial order $(T, \prec)$ defines a *switching set* of $(T, \prec)$ if there is a $\prec$-partial order $(T', \prec)$ of $\mathcal{T}$ such that $(T \setminus (T_1 \cup ... \cup T_k), \prec)$ is a prefix of $(T', \prec)$ and $(T_1 \cup ... \cup T_k) \rhd (T' \setminus (T \setminus (T_1 \cup ... \cup T_k)))$, where $(T_i, \prec) \geq (T, \prec)(t_i)$, for $i = 1, ..., k$. Here, we use $T \setminus T'$ to denote the subset of T which results from the removal of those subtransactions that belong to T'. Each subtransaction in a switching set is called a *switching point*. Thus, a switching point relates one $\prec$-partial order to another $\prec$-partial order.

Let $p_i = (T_1, \prec)$ and $p_2 = (T_2, \prec)$ be two $\prec$-partial orders of flexible transaction $\mathcal{T}$. We say that p_i has higher priority than p_2 in $\mathcal{T}$, denoted $p_1 \rightarrow p_2$, if there are $T_{1i} \subseteq T_1$ and $T_{2j} \subseteq T_2$ such that $T_{1i} \rhd T_{2j}$. The preference relation defines the preferred order over alternatives. We state

[4]Note that when $k = 1$, a flexible transaction becomes a traditional global transaction.

that two subsets $T_j, T_k \subset \mathcal{T}$ have the same priority if there is a $T_i \subset \mathcal{T}$ such that $T_i \rhd T_j$ and $T_i \rhd T_k$, but $\neg(T_j \rhd T_k)$ and $\neg(T_k \rhd T_j)$. The execution of a flexible transaction $\mathcal{T}$ at any moment must be uniquely determined. We say that a flexible transaction $\mathcal{T}$ is *unambiguous* if the following conditions are satisfied:

- For any switching set $\{t_1, ..., t_k\}$ in a $\prec$-partial order $\mathcal{T}$, $(T_1 \cup ... \cup T_k)$ where $(T_i, \prec) \geq (\mathcal{T}, \prec)(t_i)$, for $i = 1, ..., k$, has no two alternatives with the same priority.
- No any $\prec$-partial orders $p_1, ..., p_l$ of $\mathcal{T}$ are in a priority cycle such that $p_{i_1} \rightarrow ... \rightarrow p_{i_l} \rightarrow p_{i_1}$ for a permutation $i_1, ..., i_l$ of $1, ..., l$.

Note that the set of all $\prec$-partial orders of a flexible transaction may not be clearly ranked, even if it is unambiguous. The aborting of subtransactions determines which alternative $\prec$-partial order will be chosen. In the remainder of this paper, we assume that all flexible transactions are unambiguous.

So far, we have specified a flexible transaction syntactically as a set of alternative $\prec$-partial orders of subtransactions that is determined by the two relations of precedence and preference. The semantics of the precedence relation refers to the execution order of subtransactions. For instance, $t_1 \prec t_2$ may imply that t_2 cannot start before t_1 finishes or that t_2 cannot finish before t_1 finishes. Similarly, the preference relation defines alternative choices and their priority. For instance, $\{t_i\} \rhd \{t_j, t_k\}$ may imply that t_j and t_k must abort when t_i commits or that t_j and t_k should not be executed if t_i commits. In this situation, $\{t_i\}$ is of higher priority than $\{t_j, t_k\}$ to be chosen for execution.

To make the structure of a flexible transaction more visible, we also describe the structure of a flexible transaction $\mathcal{T}$ by an execution dependency graph, denoted $EDG(\mathcal{T})$, which is a directed graph whose nodes are all subtransactions of $\mathcal{T}$ and whose edges are all $t_i \xrightarrow{\rho} t_j$ $(t_i, t_j \in \mathcal{T})$, where ρ is the list of $\prec$-partial orders of $\mathcal{T}$ such that t_i precedes t_j in a $\prec$-partial order in ρ and there is no other subtransaction t_k which follows t_i and precedes t_j in the $\prec$-partial order. If a subtransaction t_i is not ordered with other subtransactions in a $\prec$-partial order, the list of the $\prec$-partial orders in which t_i participates is attached to the node of t_i. The following example is illustrative:

Example 2 *Consider a travel agent information system engaged in arranging a travel schedule for a customer. Assume that a flexible transaction $\mathcal{T}$ has the following subtransactions:*

t_1: *withdraw the plane fare from account a_1;*

t_2: *withdraw the plane fare from account a_2;*

t_3: *reserve and pay for a non-refundable plane ticket;*

t_4: *rent a car from Avis;*

t_5: *book a limo seat to and from the hotel.*

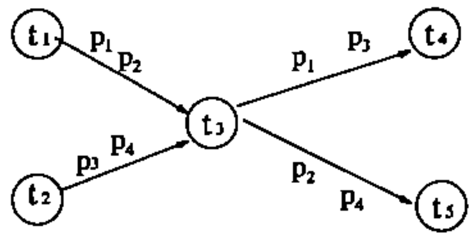


Figure 1: Execution dependency graph of Example 2.

The following $\prec$-partial orders are defined on the above subtransactions:

$$p_1 = \{t_1 \prec t_3, t_3 \prec t_4\}, \qquad p_2 = \{t_1 \prec t_3, t_3 \prec t_5\},$$
$$p_3 = \{t_2 \prec t_3, t_3 \prec t_4\}, \qquad p_4 = \{t_2 \prec t_3, t_3 \prec t_5\},$$

where $\{t_1\}$ and $\{t_4\}$ are the switching sets. With these switching sets, we have $\{t_1, t_3, t_4\} \triangleright \{t_2, t_3, t_4\}$ and $\{t_4\} \triangleright \{t_5\}$. The EDG(T) is shown in Figure 1. Clearly, the set of $\prec$-partial orders in this flexible transaction is unambiguous. Note that $p_1 \to p_2$ and $p_1 \to p_3$, but p_2 and p_3 cannot be ranked in any preferred order. □

In each $\prec$-partial order of subtransactions, the value dependencies among operations in different subtransactions define data flow among the subtransactions. Let flexible transaction G have subtransactions $t_1, t_2, \cdots, t_n$. We say that t_{j_t} is *value dependent* on $t_{j_1}, ..., t_{j_{t-1}}$ $(1 \leq j_1, ..., j_t \leq n)$, denoted $t_{j_1} \to_v t_{j_t}$, $t_{j_2} \to_v t_{j_t}$, ..., $t_{j_{t-1}} \to_v t_{j_t}$, if the execution of one or more operations in t_{j_t} is determined by the values read by $t_{j_1}, ..., t_{j_{t-1}}$.

We say that a database state is *consistent* if it preserves database integrity constraints. These constraints are formulas in predicate calculus that express relationships among data items that a database must satisfy. As defined for traditional transactions, the execution of a flexible transaction as a single unit should map one consistent multidatabase state to another. However, with flexible transactions, this definition of consistency requires that the execution of each $\prec$-partial order of subtransactions must map one consistent multidatabase state to another.

3 Ensuring Semi-atomicity

We now discuss the execution of flexible transactions. We assume that the execution of a subtransaction can be viewed by the system as a sequence of read and write accessing operations followed by either a commit or an abort termination operation. We denote *commit* and *abort* operations as c and a (possibly subscripted).

Traditionally, a transaction must be executed atomically, requiring that either all or none of its actions are completed. In the distributed database environment, this concept of atomicity has been relaxed by the notion of *semantic atomicity* [9]. Semantic atomicity differs from atomicity in that a global transaction is allowed to commit parts of its results at different times. If all subtransactions commit, then the entire global transaction commits; otherwise, the effects of all tentatively committed subtransactions are undone and the entire global transaction is to abort. The concept of semantic atomicity can be further relaxed in the execution of flexible transactions. Since a flexible transaction allows for the specification of multiple $\prec$-partial orders and results in the successful execution of the subtransactions in *one* of those $\prec$-partial orders (termed *committed $\prec$-partial order*), the execution of a flexible transaction can proceed in several different ways. The subtransactions in different $\prec$-partial orders may be attempted simultaneously, as long as any attempted subtransactions which are not in the committed $\prec$-partial order can either be aborted or have their effects undone. The *semi-atomicity* of flexible transactions, which is an extension of semantic atomicity, is defined as follows:

Definition 2 (Semi-atomicity) *A flexible transaction $\mathcal{T}$ is executed with semi-atomicity if one of the following conditions is satisfied:*

- *All its subtransactions in one of the $\prec$-partial orders commit and all other attempted subtransactions not in this $\prec$-partial order are either aborted or have their effects undone;*[5] *or*
- *No partial effects of its subtransactions remain permanent in local databases.*

Such an extension of semantic atomicity allows different $\prec$-partial orders of a flexible transaction to be attempted, possibly concurrently, while ensuring that the effects of either no or exactly one $\prec$-partial order remain permanent in the multidatabase. Thus, even if a flexible transaction commits, some of its subtransactions may abort and the effects of some committed subtransactions may be undone.

As local prepare-to-commit states are not pre-assumed in our multidatabase system, we shall now investigate the preservation of the semi-atomicity of flexible transactions through a unification of the retry and compensation approaches. As pointed out in [15], in contrast to the redo technique [5], the retry technique allows us to relax some of the restrictions on data items that global transactions can read and write.

Each subtransaction is categorized as either *retriable*, *compensatable*, or *pivot*. We say that a subtransaction t_i is *retriable* if it is guaranteed to commit after a finite number of submissions when executed from any consistent database state. The retriability of subtransactions is highly determined by integrity constraints. For instance, a bank account usually has no upper limit, so a deposit is retriable. However, it usually does have a lower limit, so a withdrawal is not retriable.

A subtransaction is *compensatable* if the effects of its execution can be semantically undone

[5]This may cause more than one subtransaction of a flexible transaction to be executed at a local site. However, a local database system may view the subtransactions in different $\prec$-partial orders of a flexible transaction as being from different global transactions.

after commitment by executing a compensating subtransaction at its local site. We assume that a compensating subtransaction ct_i for a subtransaction t_i is retriable. That is, it is guaranteed that any compensation initiated will complete successfully. This requirement, termed ***persistence of compensation***, has been discussed in the literature [9]. ct_i must also be independent of the transactions that execute between t_i and ct_i. This is because local database autonomy requires that arbitrary local transactions be executable between the time t_i is committed and the time ct_i is executed, and these local transactions can both see and overwrite the effects of t_i during that time. For example, consider an MDBS that has account a in LS_1 and account b in LS_2, with the integrity constraints $a \geq 0$ and $b \geq 0$. Suppose a transaction T_1 transfers \$100 from a to b. The withdraw subtransaction t_1 at LS_1 is compensatable, while the deposit subtransaction t_2 at LS_2 is not. The compensation of t_2 may violate the integrity constraint $b \geq 0$ if a local transaction which is executed between t_2 and its compensating subtransaction takes the amount of b. Note that both t_1 and t_2 are compensatable in the traditional distributed database environment, which ensures that the transactions that are executed between t_2 and its compensating subtransaction ct_2 are commutative [11] with ct_2.

Following [15], the compensating subtransactions that are executed for the subtransactions in each $\prec$-partial order can be considered as an independent global transaction. Consequently, the execution of these compensating subtransactions will not violate the assumption that there exists only one subtransaction for each global transaction at a local site.

We say that a subtransaction t_i is a *pivot* subtransaction if it is neither retriable nor compensatable. For example, let us consider a subtransaction which reserves and pays for a non-refundable plane ticket. Clearly, this subtransaction is not compensatable; on the other hand, this subtransaction is also not retriable, since such a ticket might never be available.

Because these properties are dependent on the semantics of the information in the database and by integrity constraints, we assume that the category of the subtransaction is specified by the global transaction definer.

[15] formulates each global transaction as the combination of a set of independent subtransactions, each of which is either compensatable, retriable, or pivot. At most one subtransaction can be pivot. In their commit protocol, the compensatable subtransactions must be committed before the commitment of the pivot subtransaction, which in turn must commit before the commitment of the retriable subtransactions. The global commit/abort decision is determined by the outcome of the pivot subtransaction commit. If it aborts, all of the compensatable subtransactions are compensated for; otherwise the retriable subtransactions are attempted until they commit. With flexible transactions (which require only semi-atomicity), we can extend this protocol to allow the execution of a flexible transaction that does not follow this subtransaction commit order and which permits multiple pivot subtransactions in the flexible transaction. A detailed discussion of these

concepts follows in the next two sections.

4 Constructing Recoverable Flexible Transactions

We shall now present a formulation for flexible transactions which integrates the retry and compensation techniques so as to preserve the semi-atomicity of their execution.

4.1 Commit Dependencies

We first examine the commit dependency relationships existing between any two subtransactions of a flexible transaction that must be obeyed in the commitment of these subtransactions. Let $\mathcal{T} = \{(T_i, \prec), i = 1, ..., k\}$ be a a flexible transaction which has subtransactions $t_1, t_2, ..., t_n$. Let $t_i, t_j \in \mathcal{T}$. We say that t_j is commit dependent on t_i, denoted $t_i \rightarrow_c t_j$, if the commitment of t_i must precede that of t_j to preserve semi-atomicity. Clearly, if $t_i \prec t_j$ in $(T_i, \prec)$ $(1 \leq i \leq k)$, then $t_i \rightarrow_c t_j$. Such commit dependencies, which are determined by the execution control flow among subtransactions, shall be termed *e-commit dependencies.*

The type (compensatable, pivot, or retriable) of subtransactions also determines their commitment order in the preservation of semi-atomicity. To ensure that a $\prec$-partial order can move either forward to its commitment or backward to the removal of any partial effects of its committed subtransactions, the commitment of compensatable subtransactions should always precede that of pivot subtransactions, which in turn should precede the commitment of retriable subtransactions. We term such commit dependencies that are determined by subtransaction type *t-commit dependencies.*

For those subtransactions which are retriable, we also observe that value dependencies must be considered in determining their commitment order. For example, assume that a value written by a subtransaction t_i at local site LS_{l_1} is dependent on a value read by a retriable subtransaction t_j at local site LS_{l_2}. If t_i commits and t_j aborts, then t_j should be retried. However, local transactions may be executed after t_j is aborted but before it is retried at LS_{l_2}, which may result in inconsistencies between the data read from the original execution of t_j and from its retrial. To ensure that the retrial of t_j does not result in any database inconsistency, when a subtransaction t_i is value dependent on t_j, the commitment of t_j must precede that of t_i. Thus, if the retrial of t_j leads to a result which is different from that of its original execution, t_i that has read the data from the original execution of t_j may be aborted and re-executed. Consequently, each retriable subtransaction remains retriable without resulting in any database inconsistency as long as all other subtransactions that are value dependent upon it have not committed. We designate such commit dependencies that are determined by the value dependencies among subtransactions as *v-commit dependencies.*

These three types of commit dependencies are closely related. Whenever a subtransaction t_2 is e-commit dependent, t-commit dependent, or v-commit dependent on t_1, we must have $t_1 \rightarrow_c t_2$. A set of subtransactions $t_1, ..., t_k$ is in a *commit dependency cycle* if there is a permutation $i_1, ..., i_k$ of $1, ..., k$ such that $t_{i_1} \rightarrow_c t_{i_2} \rightarrow_c ... \rightarrow_c t_{i_k} \rightarrow_c t_{i_1}$. The existence of a commit dependency cycle in a flexible transaction may cause its commitment to result in a deadlock situation, in which each subtransaction has to wait for another subtransaction to commit before its commitment. In addition, the possession of two or more pivot subtransactions in a $\prec$-partial order may render it difficult to determine a commit order among them which ensures that the $\prec$-partial order can move either forward to the commitment of its subtransactions or backward to the removal of any partial effects of the committed subtransactions. In the remainder of this section, we examine those restrictions that need to be placed on flexible transactions.

4.2 Well-formed Flexible Transactions

As e-commit dependencies must be obeyed in the commitment of subtransactions, the commitment of any pivot or retriable subtransaction t in T will cause the effects of subtransactions in T' where $(T', \prec) \leq (T, \prec)(t)$ to no longer be undoable. We define a pivot subtransaction as a *critical subtransaction* if all subtransactions in T' where $(T', \prec) \leq (T, \prec)(t)$ are compensatable.[6] $(T_i, \prec)$ may have several critical subtransactions which are not ordered with each other by the $\prec$ relation. One critical subtransaction acts as the *critical point* of the $\prec$-partial order; its commit or abort determines whether the $\prec$-partial order can either go forward to its commitment or must move backward to the removal of any partial effects of its committed subtransactions. The *critical point* of $(T_i, \prec)$ is determined through the following rules:

- If $t \in T_i$ is the only critical subtransaction, then t is the critical point; otherwise
- If there is a critical subtransaction $t \in T_i$ which is not a switching point, then choose t as the critical point; otherwise
- A critical subtransaction is randomly chosen as the critical point.

Any compensatable subtransactions which precede a critical subtransaction and are not ordered in $\prec$ relation with the critical point can commit before it, and any retriable subtransactions which are not ordered with the critical point can commit after it. However, the abort of any pivot subtransaction which is not the critical point and of any compensatable subtransaction which follows a pivot or retriable subtransaction in $(T_i, \prec)$ may hamper $(T_i, \prec)$ in either moving forward to its commitment or backward to the removal of any partial effects of its committed subtransactions. Such problematic subtransactions are termed ***abnormal subtransactions.*** More precisely, a

[6]If $(T_i, \prec)$ has no such pivot subtransaction, then a dummy null pivot subtransaction, which is not ordered in $\prec$ relation with any subtransaction of T_i, is created.

subtransaction t in T_i is abnormal if one of the following conditions is satisfied:

- t is a compensatable or pivot subtransaction and there is a pivot or retriable subtransaction t_1 in T_i such that $t_1 \prec t$; or
- t is a pivot subtransaction but is not the critical point.

Otherwise, t is a *normal* subtransaction. Note that only a compensatable or pivot subtransaction may be an abnormal subtransaction.

Following the discussion in [15], no abnormal subtransactions can be permitted in a traditional global transaction. Otherwise, the semantic atomicity of the global transaction may not be preserved. As a result, each global transaction can only have one pivot subtransaction. This may be too restrictive for some applications, especially those complex global applications in an MDBS environment which involve many local sites.

The use of flexible transactions can extend the traditional global transaction model to permit the presence of abnormal subtransactions. In Example 2, it is obvious that t_1 and t_2 are compensatable, t_3 is pivot, and t_4 is compensatable. If we assume that a limo is available in the required time period, t_5 is retriable. Note that t_4 is an abnormal subtransaction in both p_1 and p_3. If t_1 and t_3 have already committed and then t_4 aborts, the partial effects of p_1 cannot be undone. However, the execution of t_4 can be replaced by the execution of t_5. As t_5 is retriable, $\mathcal{T}$ can be committed. Thus, abnormal subtransactions can be permitted in flexible transactions. However, if any abnormal subtransaction t_1 aborts, appropriate actions must ensue to continue the execution of the flexible transaction.

Let $\mathcal{T}$ be a flexible transaction and $(T_i, \prec)$ be a $\prec$-partial order of $\mathcal{T}$ which has an abnormal subtransaction t. Let an *immediate predecessor* of a subtransaction t denote a subtransaction t_1 such that $t_1 \prec t$ and no other subtransaction t_2 exists such that $t_1 \prec t_2 \prec t$. We say that an abnormal subtransaction t in T_i is a *blocking point* if one of the following conditions is satisfied:

- all predecessors of t are normal. In this case, t must be a pivot subtransaction; or
- all immediate predecessors of t are not compensatable; or
- if t has a immediate predecessor t_1 which is compensatable, t_1 has a successor which is not ordered by $\prec$ relation with t and not compensatable.[7]

An abnormal subtransaction t that is a blocking point in $\prec$-partial order $(T_i, \prec)$ identifies a place at which the abortion or compensation of t must result in the replacement of the execution of $(T_i, \prec)$ with an alternative $\prec$-partial order. The following example is illustrative:

[7]This is because, if the non-compensatable successor commits, then it is no longer possible to back up to the point where t_1 can be undone.

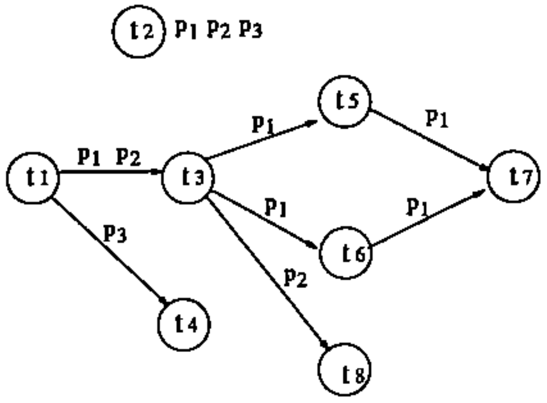


Figure 2: Execution dependency graph of Example 3.

Example 3 *Assume that a flexible transaction $\mathcal{T}$ is defined in Figure 2. Let t_1, t_5, t_6, and t_7 be compensatable subtransactions, t_2 and t_3 be pivot subtransactions, and t_4 and t_8 be retriable subtransactions. There are three $\prec$-partial orders in $\mathcal{T}$:*

$p_1 = (\{t_1, t_2, t_3, t_5, t_6, t_7\}, \prec)$,

$p_2 = (\{t_1, t_2, t_3, t_8\}, \prec)$,

$p_3 = (\{t_1, t_2, t_4\}, \prec)$,

where $\{t_5, t_6, t_7\} \rhd \{t_8\}$ and $\{t_3, t_5, t_6, t_7, t_8\} \rhd \{t_4\}$. t_2 and t_3 are the critical subtransactions of both p_1 and p_2. t_2 is the critical subtransaction for p_3. By the critical point determination rules, t_2 is the critical point for all three partial orders, because there is alternative if the pivot subtransaction t_3 fails. Thus, t_3, t_5, t_6, t_7 are abnormal subtransactions and among them, t_3, t_5, and t_6 are the blocking points. □

We define *well-formed* flexible transactions as follows:

Definition 3 (Well-formed flexible transaction) *A flexible transaction $\mathcal{T}$ is well-formed if every abnormal subtransaction which is a blocking point is a switching point such that, in its switching set, all other switching points are abnormal subtransactions and the successors of each switching point which are not ordered by $\prec$ relation with the successors of another switching point are compensatable.*

Thus, in a well-formed flexible transaction, for any $\prec$-partial order $(T_i, \prec)$ which contains an abnormal subtransaction t, there is at least one alternative $\prec$-partial order $(T_j, \prec)$ which shares a prefix with $(T_i, \prec)$ such that the aborting of t will lead the execution of $\mathcal{T}$ from $(T_i, \prec)$ to $(T_j, \prec)$ without resulting in any database inconsistency.

Observation 1 *If a flexible transaction is well-formed, it has at least one $\prec$-partial order such that it has no abnormal subtransaction and no other $\prec$-partial order has lower priority than it.*

In Example 2, the flexible transaction $\mathcal{T}$ has two $\prec$-partial orders p_2 and p_4 that contain no abnormal subtransaction. The only blocking point t_4 constitutes a switching set. Thus, $\mathcal{T}$ is well-formed. In Example 3, the flexible transaction $\mathcal{T}$ has one $\prec$-partial order p_3 that contains no abnormal subtransaction. Since the blocking point t_3 constitutes the switching set and the blocking points t_5 and t_8 also constitute the switching set, $\mathcal{T}$ is also well-formed.

4.3 Maintaining the Recoverability of Flexible Transactions

We now discuss the preservation of semi-atomicity. Following [4], we define a *schedule over a set of transactions* as a partial order of the operations of those transactions which orders all conflicting operations and which respects the order of operations specified by the transactions. A global schedule S is a schedule over both local and flexible transactions which are executed in an MDBS. We denote $o_1 <_S o_2$ if operation o_1 is executed before operation o_2 in global schedule S.

Clearly, to preserve semi-atomicity, the commit dependencies of flexible transactions must be correctly preserved in global schedules. We formulate below the concept of a *commit dependency graph* that is defined on each well-formed flexible transaction, effectively incorporating all effects of e-commit dependencies, t-commit dependencies, and v-commit dependencies.

Definition 4 (Commit dependency graph of a flexible transaction) *A commit dependency graph of a well-formed flexible transaction $\mathcal{T} = \{(T_i, \prec), i = 1, ..., k\}$, denoted CDG($\mathcal{T}$), is a directed graph whose nodes are all subtransactions of $\mathcal{T}$ and whose edges are all $t_1 \rightarrow t_2$ ($t_1, t_2 \in T_i$ for $1 \leq i \leq k$ and $t_1 \neq t_2$) such that:*

- *(e-commit dependency) $t_1 \prec t_2$;*
- *(v-commit dependency) $t_1 \rightarrow_v t_2$ and t_1 is retriable;*
- *(t-commit dependency) t_1 is compensatable and normal and t_2 is the critical point; or*
- *(t-commit dependency) t_1 is the critical point and t_2 is either pivot or retriable.*

We define the concept of *commit dependency preserving* on global schedules as follows:

Definition 5 (Commit dependency preserving) *Let $\mathcal{G}$ be a set of well-formed flexible transactions. A global schedule S is commit dependency preserving if, for any two subtransactions t_1 and t_2 of $\mathcal{T}$ in S such that $t_1 \rightarrow t_2$ in CDG($\mathcal{T}$), $c_{t_2} \in S$ implies $c_{t_1} <_S c_{t_2}$.*

Lemma 1 *Let $\mathcal{G}$ be a set of flexible transactions that participate in global schedule S. If, for each $\mathcal{T}$ in $\mathcal{G}$, CDG($\mathcal{T}$) is acyclic, then S is commit dependency preserving.*

Proof: Since, for each $\mathcal{T}$ in $\mathcal{G}$, CDG($T_i, \prec$) is acyclic for all $(T_i, \prec)$ in $\mathcal{T}$, for any $(T_i, \prec)$ in $\mathcal{T}$, CDG($T_i, \prec$) may be topologically sorted. Without loss of generality, let $t_1, ..., t_m$ be the nodes of

CDG($T_i, \prec$) and $j_1, ..., j_m$ be a permutation of 1,2,...,m such that $t_{j_1}, t_{j_2}, ..., t_{j_m}$ is a topological sort of CDG($T_i, \prec$). This order ensures that the commitment orders of these subtransactions in global schedule S conform to the definition of commit dependency preserving. To illustrate this, let t_l and t_k be subtransactions in T_i such that $t_k \rightarrow_c t_l$. By the definition of CDG($T_i, \prec$), $t_k \rightarrow t_l$ is an edge in CDG($T_i, \prec$). Thus, t_k must appear before t_l in the topological sort $t_{j_1}, t_{j_2}, ..., t_{j_m}$. If the commitment order of all subtransactions in T_i follows the order of $t_{j_1}, t_{j_2}, ..., t_{j_m}$ in global schedule S, then the commitment of t_k precedes that of t_l in S. Hence, S is commit dependency preserving. □

A well-formed flexible transaction in a commit dependency preserving global schedule may still not be semi-atomic. For example, if two alternative retriable subtransactions commit simultaneously, it will be impossible to undo the effects of one of those subtransactions. Since the effects of both will remain, the execution of that flexible transaction cannot be semi-atomic. We define the concept of *F-recoverability* on global schedules as follows:

Definition 6 (F-recoverability) *Let $\mathcal{G}$ be a set of well-formed flexible transactions. A global schedule S is F-recoverable if, for each flexible transaction $\mathcal{T}$ in $\mathcal{G}$, no two pivot or retriable subtransactions t_1 and t_2 which participate in different $\prec$-partial orders commit in S simultaneously.*

Theorem 1 *Let $\mathcal{T}$ be a well-formed flexible transaction and CDG($\mathcal{T}$) be acyclic. If global schedule S is commit dependency preserving and F-recoverable, then the semi-atomicity of $\mathcal{T}$ is preservable.*

Proof: Without loss of generality, we assume that each $\prec$-partial order of $\mathcal{T}$ contains at least one pivot subtransaction. Following Lemma 1, we can assume that S is commit dependency preserving. The proof proceeds by induction on a number n of $\prec$-partial orders of $\mathcal{T} = \{(T_i, \prec), \text{i=1,...,n}\}$:

Basic step (n=1). $\mathcal{T}$ is a traditional global transaction. If every subtransaction of T_1 commits, then the atomicity of $(T_1, \prec)$ is obviously preserved. Suppose $t \in T_1$ is aborted. Assume that t is either compensatable or pivot. Since S is commit dependency preserving, by the definition of $\rightarrow_c$, all committed subtransactions of T_1 in S must be compensatable. Hence, the committed partial effects of T_1 can be undone by the execution of the corresponding compensating subtransactions. Now assume that t is retriable. Again, since S is commit dependency preserving, by the definition of $\rightarrow_c$, any subtransactions of T_1 which are value dependent on t must not have committed in S. Let $t' \in T_1$, which is value dependent on t, also be executed but not yet committed in S. t' can be aborted and resubmitted for execution if the retrial of t would result in inconsistency in the execution of t'. As our model assumes that value dependencies and $\prec$ are the only relationships in effect among the subtransactions of each $\prec$-partial order, t can thus be retried without creating any multidatabase inconsistencies. Hence, the semi-atomicity of $\mathcal{T}$ is preservable.

Induction. Suppose for $n = k(\geq 1)$, the semi-atomicity of $\mathcal{T}$ is preservable. Consider $n = k + 1$. Let $(T_i, \prec)$ be a $\prec$-partial order of $\mathcal{T}$. Consider the following situations:

(1) If every subtransaction of T_i commits, then, since S is F-recoverable, no pivot or retriable subtransaction in another $\prec$-partial order of $\mathcal{T}$ has committed. Thus, the effects of all committed subtransactions of $\mathcal{T}$ which are not in T_i can be undone by the execution of the corresponding compensating subtransactions. Consequently, the semi-atomicity of $\mathcal{T}$ is preservable.

(2) Suppose now that $t \in T_i$ is aborted and t is either compensatable and normal or a critical point. Since S is commit dependency preserving, by the definition of $\rightarrow_c$, all committed subtransactions of T_i in S must be compensatable. Any of the partial effects of T_i can be undone. The problem then is reduced to preserving the semi-atomicity of less than $k+1$ $\prec$-partial orders of $\mathcal{T}$. Thus, by the induction hypothesis, the semi-atomicity of $\mathcal{T}$ is preservable.

(3) Suppose now that $t \in T_i$ is aborted and t is retriable. Then, at least one pivot subtransaction of $(T_i, \prec)$ has committed. By the F-recoverability of S, we know that no pivot or retriable subtransaction in another $\prec$-partial order of $\mathcal{T}$ has committed. As with the proof in the basic step, since S is commit dependency preserving, t can be retried until commitment. The situation then resolves to either (1) or (4).

(4) Suppose now that $t \in T_i$ is aborted and t is abnormal. Since $\mathcal{T}$ is well-formed, by Definition 3, the execution of $(T_i, \prec)$ can be changed to another $\prec$-partial order $(T_j, \prec)$ without resulting in any database inconsistency. The problem then is reduced to preserving the semi-atomicity of less than $k+1$ $\prec$-partial orders of $\mathcal{T}$. Thus, by the induction hypothesis, the semi-atomicity of $\mathcal{T}$ is preservable. □

We say that a flexible transaction is *recoverable* if its semi-atomicity is guaranteed to be preserved. Based upon Theorem 1, if a flexible transaction is well-formed and its commit dependency graph is acyclic, then it is recoverable.

Because abnormal subtransactions are permitted, the concept of a well-formed flexible transaction greatly extends the scope of global transactions that can be specified in the MDBS environment beyond that of the basic global transaction model proposed in [15]. This was demonstrated in Example 2, where a compensatable transaction (t_4: reserve a rental car) could be attempted after the pivot (t_3: reserve and pay for a non-refundable plane ticket) because of the presence of an alternative retriable transaction (t_5: book the limo).

5 The Flexible Transaction Commit Protocol

In this section, we present a system model and a commit protocol for flexible transactions that is based upon Theorem 1. It maintains semi-atomicity during the commit process through a combination of retry and compensation techniques.

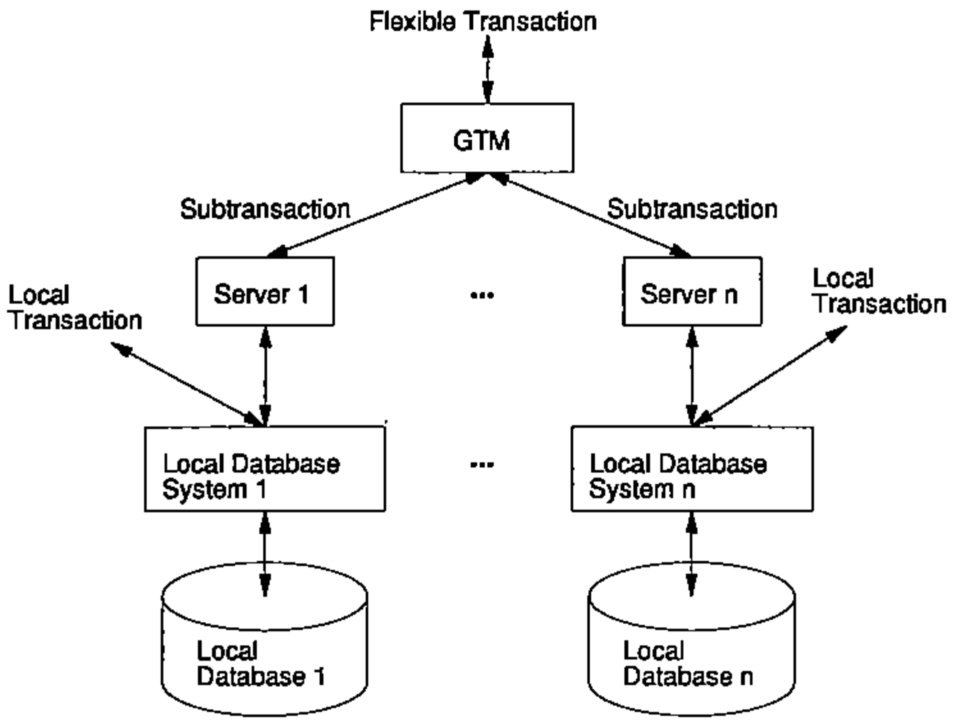


Figure 3: The multidatabase system model.

5.1 System Model

The system model employed for this protocol is shown in Figure 3. We assume that the Global Transaction Manager (GTM) submits flexible transaction operations to the local databases through servers that are associated with each local local database system. This model assumes that, at any time, only one $\prec$-partial order of each flexible transaction is executing.[8] The local databases also may execute their own local transactions independently.

In the flexible transaction definition, we assume that a specification mechanism is provided to allow users to identify to the system the type (compensatable, pivot, or retriable) of each subtransaction. The compensating subtransactions are submitted together with the corresponding compensatable subtransactions. The GTM ensures that the flexible transaction is well-formed by finding all abnormal subtransactions which are blocking points and ensuring that each is also a switching point such that all other switching points in the same switching set are abnormal subtransactions and the successors of each switching point which are not ordered by $\prec$ relation with the successors of another switching point are compensatable. Also, a commit dependency graph is built for each flexible transaction and is checked for cycles. If the flexible transaction is

[8]A multiple-threads approach which deals with multiple $\prec$-partial orders simultaneously can be extended from this basic approach. Such an approach should not commit more than one pivot or retriable subtransaction simultaneously in different $\prec$-partial orders of a flexible transaction or F-recoverability may be violated.

not well-formed or if its commit dependency graph is cyclic, it is rejected.

As each subtransaction begins to execute, the type of the subtransaction is sent to the local database server with the *begin* command. The operations belonging to the subtransaction are submitted to an individual local database by its server as a part of a single subtransaction. The completion of each submitted operation, as well as the begin and commit operations, are individually acknowledged by the local database server to the GTM.

5.2 The Commit Protocol

In the proposed protocol, commitment of flexible transactions is approached in a *dynamic* manner. Each subtransaction is permitted to commit locally as soon as possible after it has finished executing. Such an approach is mandated by the possible presence of value dependencies among subtransactions. If $t_1 \rightarrow_v t_2$ is a value dependency between two subtransactions t_1 and t_2, then the value t_2 reads cannot be guaranteed until t_1 commits. The abort of t_1 may force the abort of t_2, resulting in a cascading abort. To avoid such cascading aborts, we restrict that if $t_1 \rightarrow_v t_2$ and t_1 is retriable, then t_2 is not submitted until t_1 commits. In contrast, the commit protocol defined in [15] is *static*, in that it processes the commit operations of a global transaction following the completed execution of all its subtransactions.

We assume that the GTM maintains state information for each subtransaction in the commit dependency graph of its flexible transaction. This state is *inactive* if the subtransaction has not started its execution, is *active* if the subtransaction has started but not completed its execution,[9] is *to_be_committed* if the subtransaction has completed its execution, is *committed* if the subtransaction has committed, is *aborted* if the subtransaction has aborted, and is *committed-reversed* if both the subtransaction and its compensating subtransaction have committed. Figure 4 shows the state transition diagram for a subtransaction. In this figure, *op* is an operation submitted by the GTM, and *ack* is an acknowledgement from a local database server.

The execution and dynamic commitment of a flexible transaction is coordinated via message exchanges. The message exchanges that occur during the commitment of a single subtransaction t are shown in Figure 5. In this figure, items in italics are messages that are passed. Other items indicate code that is executed when the message is received. Time proceeds from top to bottom.

When the last operation in a subtransaction is acknowledged, the GTM updates t's subtransaction state to "to_be_committed." The GTM can thus commit the subtransaction as soon as is consistent with the maintenance of semi-atomicity. Once all the subtransactions that precede t in the commit dependency graph have committed, the GTM sends a *commit(t)* message to t's server, which then tries to commit t in the local database. This commit is either successful (case (a) in

[9]That is, it has not completed its read and write operations.

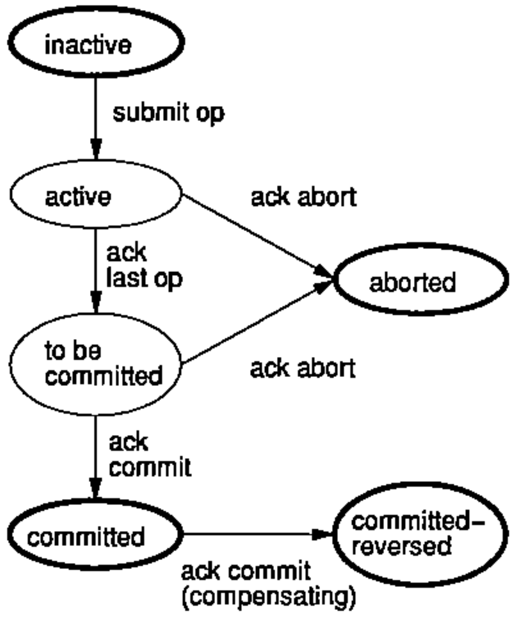


Figure 4: State transitions for a subtransaction in a flexible transaction.

Figure 5) or unsuccessful (case (b) or (c)). If it is successful, the final state of the subtransaction in the GTM is *committed.* Otherwise, if is retriable, the subtransaction is retried by the local database server until it eventually commits, and the final state of the subtransaction becomes *committed.* Otherwise, it is set to *aborted.*

When an $ack\_abort(t)$ is received, the GTM seeks an alternative $\prec$-partial order of the flexible transaction. Such an alternative may be obtained directly, or one may be constructed by finding alternatives of more remote predecessors and trying from an early point. In this paper, we assume that the next alternative attempted by the GTM requires as little backtracking as possible and that the alternatives must be tried in preference order. The following algorithm describes the backtracking method:

Algorithm 1 (Backtracking Algorithm) *Input: t, the aborted subtransaction.*

1. *If t is a switching point, find the switching set that includes t and that has the fewest committed successors. Otherwise, find the closest predecessor t_1 in the $\prec$-partial order that is a switching point and then find the switching set that includes t_1 that has the fewest committed successors.*
2. *If a switching set is found, abort all subtransactions in the switching set as well as all their successors that are in the active state and compensates for all subtransactions in the switching set and their successors that are in the committed state. Attempt the next untried alternative in preference order.*

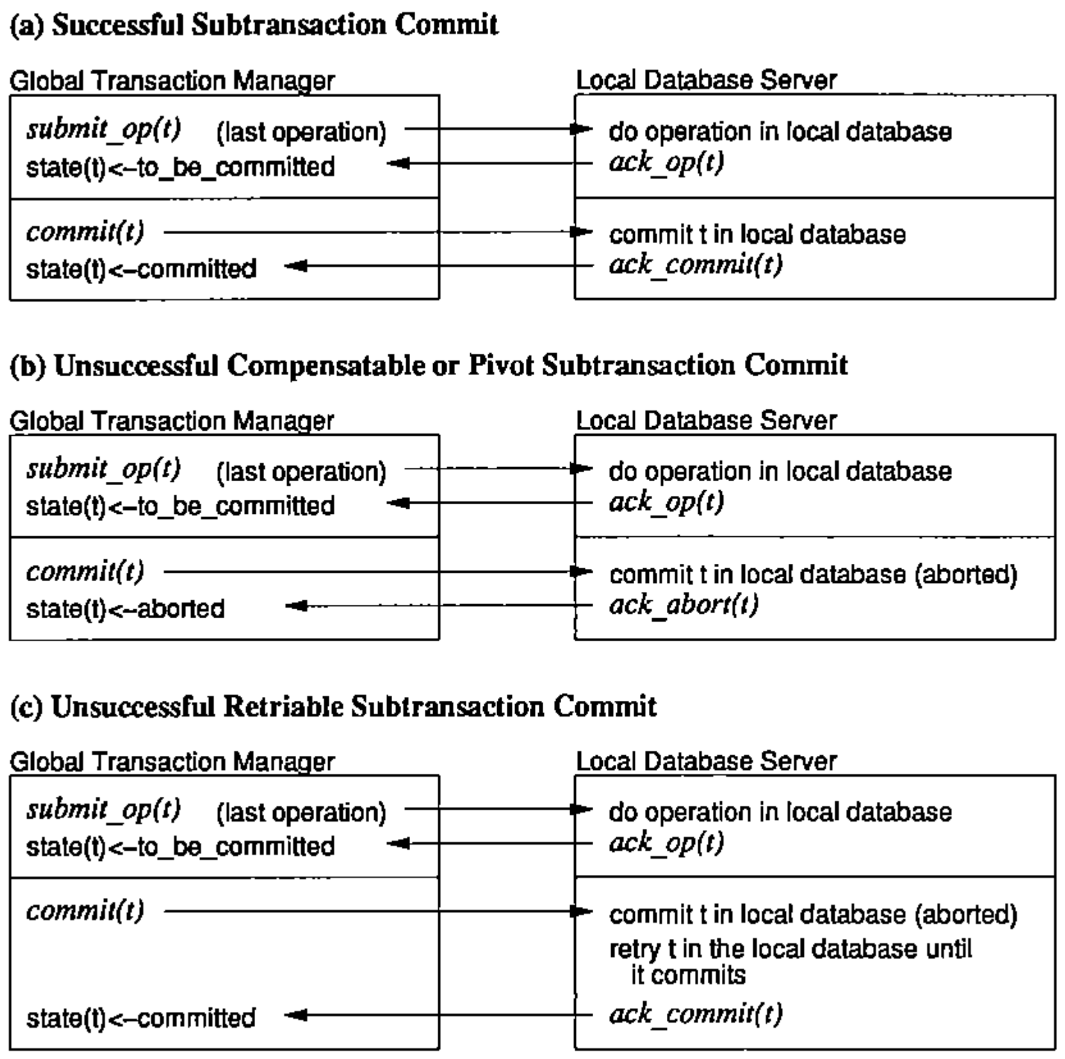
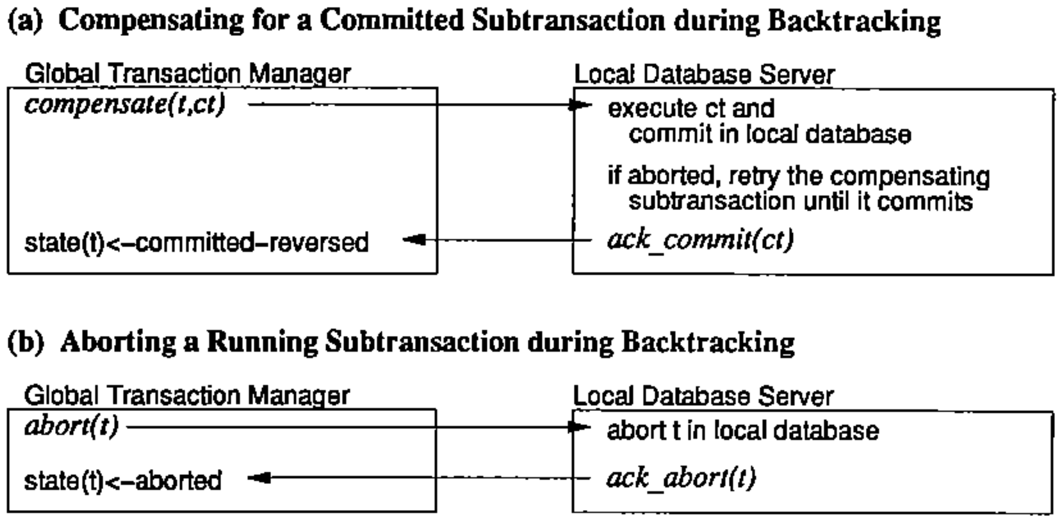

(a) Successful Subtransaction Commit
Global Transaction Manager
Local Database Server
submit_op(t) (last operation)
do operation in local database
state(t)<--to_be_committed
ack_op(t)
commit(t)
commit t in local database
state(t)<--committed
ack_commit(t)
(b) Unsuccessful Compensatable or Pivot Subtransaction Commit
Global Transaction Manager
Local Database Server
submit_op(t) (last operation)
do operation in local database
state(t)<--to_be_committed
ack_op(t)
commit(t)
commit t in local database (aborted)
state(t)<--aborted
ack_abort(t)
(c) Unsuccessful Retriable Subtransaction Commit
Global Transaction Manager
Local Database Server
submit_op(t) (last operation)
do operation in local database
state(t)<--to_be_committed
ack_op(t)
commit(t)
commit t in local database (aborted)
retry t in the local database until it commits
state(t)<--committed
ack_commit(t)

Figure 5: Message passing sequences for committed and aborted subtransactions. (a) Commit succeeds in the local database. (b) Commit fails in the local database, subtransaction is not retriable. (c) First commit fails in the local database, subtransaction is retriable.



(a) Compensating for a Committed Subtransaction during Backtracking
Global Transaction Manager
compensate(t,ct)
state(t)<–committed–reversed
Local Database Server
execute ct and commit in local database
if aborted, retry the compensating subtransaction until it commits
ack_commit(ct)
(b) Aborting a Running Subtransaction during Backtracking
Global Transaction Manager
abort(t)
state(t)<–aborted
Local Database Server
abort t in local database
ack_abort(t)

Figure 6: Message passing sequences used during backtracking: (a) when compensation is required, (b) when aborting is possible.

3. *If no switching set is found, abort the flexible transaction by aborting all subtransactions in the $\prec$-partial order that are in the active state and compensates for all subtransactions in the $\prec$-partial order that are in the committed state.*

When a subtransaction must be compensated for, the compensating subtransaction may need to be retried until it actually succeeds. When the compensating subtransaction commits, the state of the original subtransaction in the GTM changes to committed-reversed. Thus, the execution of the compensating subtransaction *ct* of subtransaction *t* is as shown in Figure 6.

When all subtransactions in the currently executing $\prec$-partial order of the flexible transaction enter the committed state, the flexible transaction commits. All its subtransactions which are not in the committed $\prec$-partial order should be in either inactive, aborted, or committed-reversed states. Otherwise, there is no committed $\prec$-partial order, the flexible transaction aborts. In this case, all its subtransactions should be in either inactive, aborted, or committed-reversed state.

5.3 Discussion

The flexible transaction commit protocol preserves the semi-atomicity of the flexible transactions in a multidatabase. This is shown as follows:

Theorem 2 *The flexible transaction commit protocol ensures that the semi-atomicity of flexible transactions is preserved, provided that the flexible transactions are well-formed and have acyclic commit dependency graphs.*

Proof: Let $\mathcal{T} = \{(T_i, \prec), i = 1, ..., k\}$ be a well-formed flexible transaction and $CDG(\mathcal{T})$ is acyclic. Let $(T_i, \prec)$ of $\mathcal{T}$ be currently in execution. Because the GTM issues commits for the subtransactions of T_i in the order that is defined in $CDG(\mathcal{T})$, the global schedule is commit dependency preserving. When the GTM receives an $ack\_abort(t)$, there are two cases to be considered. If t is either compensatable and normal or a critical point, then any partial effects of $(T_i, \prec)$ can be undone. Based upon the backtracking algorithm, the GTM either changes the execution of $\mathcal{T}$ to an alternative $\prec$-partial order $(T_j, \prec)$ by properly undoing some of the partial effects of $(T_i, \prec)$ or if no alternative exists, it undoes all partial effects of $(T_i, \prec)$ and aborts the entire flexible transaction. If t is abnormal, then, since $\mathcal{T}$ is well-formed, the backtracking algorithm automatically changes the execution of $\mathcal{T}$ to an alternative $\prec$-partial order $(T_j, \prec)$. In either case, the backtracking algorithm guarantees that no partial effects of $(T_i, \prec)$ remain permanent. When the GTM receives commit acknowledgement for all subtransactions of T_i, then $\mathcal{T}$ is determined to be committed. At this moment, no partial effects of any other $\prec$-partial orders remain permanent in the multidatabase.[10] Hence, the execution of $\mathcal{T}$ satisfies Definition 2. □

This commit protocol also is effective with local databases that accept only transactions that are executed and committed as a unit. However, in such instances, the submission and execution of the subtransaction on the local database must occur at the time it is expected to commit. Thus, the *inactive*, *active*, and *to_be_committed* states are all collapsed into a single *to_be_committed* state, and the transition to the *committed* state occurs when the commit acknowledgment for the entire subtransaction is received. If the transaction aborts, then the transition is to the *aborted* state.

Both the GTM commit protocol described in [15] and the flexible transaction commit protocol outlined here prevent the severe blocking of local transactions that may be caused by the 2PC protocol. As the flexible transaction commit protocol permits each subtransaction to commit dynamically without waiting for the other subtransactions of the same flexible transaction to complete their execution, it generates even less blocking of local transactions than does the GTM commit protocol developed in [15]. However, to prevent cascading aborts, value dependencies may cause the execution of some subtransactions to be delayed by retriable subtransactions upon which they are value dependent, a complication not present with the GTM commit protocol. As a result, the GTM commit protocol may achieve better execution performance for the more restricted class of input global transactions with which it deals.

6 Conclusions

Global transaction management in an error-prone MDBS environment has been recognized as a substantial and as yet unresolved issue in those instances in which the component local database

[10]Note that F-recoverability is also guaranteed in the global schedule.

systems do not support prepare-to-commit states [20]. We have advanced a theory which facilitates the preservation of semi-atomicity, a weaker formulation of flexible transaction atomicity which is appropriate to an MDBS environment in which local database systems are required to maintain only serializability and recoverability. This theory includes definitions of a fundamental transaction model and of the semi-atomicity property of flexible transactions and the classification of those flexible transactions that can be executed in the presence of failures.

The preservation of the weaker property of semi-atomicity renders flexible transactions more resilient to failures than are traditional global transactions. This property is preserved through a combination of the compensation and retry approaches. Local prepare-to-commit states are thus not required. The construction of recoverable flexible transactions that are executable in the error-prone MDBS environment demonstrates that the flexible transaction model indeed enhances the scope of global transaction management beyond that offered by the traditional global transaction model. The design of the proposed protocol is currently being investigated as part of the InterBase project at Purdue University.

It is worthy of note that the proposed theory and protocol can be applied as well to the traditional distributed database environment. Using flexible transactions, the retry and compensation techniques can prevent the severe blocking that may be caused by the 2PC protocol. Compensating subtransactions may be subject to fewer restrictions in such an environment.

This discussion has addressed solely the issues relevant to the consistency and reliability of a single flexible transaction. A complete exploration of the concurrency control of flexible transactions must examine the effect of compensation on the concurrent execution of such transactions. The results of these investigations are presented elsewhere.

Acknowledgements

We would like to acknowledge Rachel Ramadhyani for her input regarding the presentation of this paper.

References

[1] M. Ansari, L. Ness, M. Rusinkiewicz, and A. Sheth. Using Flexible Transactions to Support Multi-System Telecommunication Applications. In *Proceedings of the 18th VLDB conference*, Vancouver, Canada, Aug. 1992.

[2] M. Ansari, M. Rusinkiewicz, L. Ness, and A. Sheth. Executing Multidatabase Transactions. In *Proceedings of the 25th Hawaii International conference on System Sciences*, Hawaii, Jan. 1992.

[3] K. Barker and M. Özsu. Reliable transaction execution in multidatabase systems. In *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems*, pages 344–347, Kobe, Japan, Apr. 1991.

[4] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Databases Systems*. Addison-Wesley Publishing Co., 1987.

[5] Y. Breitbart, A. Silberschatz, and G. Thompson. Reliable Transaction Management in a Multidatabase System. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 215–224, May 1990.

[6] U. Dayal, M. Hsu, and R. Ladin. A transactional model for long-running activities. *17th VLDB Proceedings*, pages 113–122, 1991.

[7] A. K. Elmagarmid and et al. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers, 1992.

[8] A. K. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A Multidatabase Transaction Model for InterBase. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 507–581, Brisbane, Australia, Aug. 1990.

[9] H. Garcia-Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM Trans. Database Syst.*, 8(2):186–213, June 1983.

[10] V. Gligor and R. Popescu-Zeletin. Transaction Management in Distributed Heterogeneous Database Management Systems. *Information Systems*, 11(4):287–297, 1986.

[11] H. Korth, E. Levy, and A. Silberschatz. A Formal Approach to Recovery by Compensating Transactions. In *Proceedings of the 16th International Conference on Very Large Data Bases*, Brisbane, Australia, Aug. 1990.

[12] E. Kühn, F. Puntigam, and A. Elmagarmid. An execution model for distributed database transactions and its implementation in VPL. In A. Pirotte, C. Delobel, and G. Gottlob, editors, *Lecture Notes in Computer Science, Advances in Database Technology — EDBT '92*, pages 483–498. Springer-Verlag, 1992. Proceedings of the 3rd International Conference on Extending Database Technology, Vienna, Austria, March, 1992.

[13] Y. Leu, A. Elmagarmid, and N. Boudriga. Specification and execution of transactions for advanced database applications. *Information Systems*, 17(2), 1992.

[14] E. Levy, H. Korth, and A. Silberschatz. An Optimistic Commit Protocol for Distributed Transaction Management. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Denver, Colorado, May 1991.

[15] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz. A transaction model for multidatabase systems. In *Proceedings of International Conference on Distributed Computing Systems*, June 1992.

[16] P. Muth and T. Rakow. Atomic commitment for integrated database systems. In *Proceedings of the 7th Intl. Conf. on Data Engineering*, pages 296–304, Kobe, Japan, Apr. 1991.

[17] M. H. Nodine. *Interactions: Multidatabase Support for Planning Applications*. PhD thesis, Brown University, April 1993.

[18] M. H. Nodine and S. B. Zdonik. Automating compensation in a multidatabase. In *Proceedings of the 27th Hawaii International Conference on System Sciences*, 1994.

[19] W. Perrizo, J. Rajkumar, and P. Ram. HYDRO: a heterogeneous distributed database system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 32–39, Denver, Colorado, USA, May 1991.

[20] A. Silberschatz, M. Stonebraker, and J. Ullman. Database systems: Achievements and opportunities. *Communication of ACM*, 34(10):110–120, 1991.

[21] N. Soparkar, H. F. Korth, and A. Siberschatz. Failure-resilient transaction management in multidatabases. *IEEE Computer*, 24(12):28–36, December 1991.

[22] J. Veijalainen, F. Eliassen, and B. Holtkamp. The S-transaction Model. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992.

[23] A. Wolski and J. Veijalainen. 2PC Agent method: Achieving serializability in presence of failures in a heterogeneous multidatabase. In *Proceedings of PARBASE-90*, Miami Beach, Florida, 1990.