Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems

Aidong Zhang[†] Marian Nodine[‡] Bharat Bhargava[†] Omran Bukhres[†]

[†]Department of Computer Science
Purdue University
West Lafayette, IN 47907 USA

[‡]Department of Computer Science
Brown University
Providence, RI 02912 USA

Abstract

Global transaction management requires cooperation from local sites to ensure the consistent and reliable execution of global transactions in a distributed database system. In a heterogeneous distributed database (or multidatabase) environment, various local sites make conflicting assertions of autonomy over the execution of global transactions. A flexible transaction model for the specification of global transactions makes it possible to deal robustly with these conflicting requirements. This paper presents an approach that preserves the *semi-atomicity* (a weaker form of atomicity) of flexible transactions, allowing local sites to autonomously maintain serializability and recoverability. We offer a fundamental characterization of the flexible transaction model and precisely define the semi-atomicity. We investigate the commit dependencies among the subtransactions of a flexible transaction. These dependencies are used to control the commitment order of the subtransactions. We next identify those restrictions that must be placed upon a flexible transaction to ensure the maintenance of its semi-atomicity. As atomicity is a restrictive criterion, semi-atomicity enhances the class of executable global transactions.

1 Introduction

A multidatabase system (MDBS) is a collection of autonomous local databases (LDBSs) and it is viewed as a single unit. There are two types of transactions in an MDBS. A local transaction, which accesses a local database only, is submitted directly to an LDBS. 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 various LDBSs.

SIGMOD 94- 5/94 Minneapolis, Minnesota, USA
© 1994 ACM 0-89791-639-5/94/0005..$3.50

Atomicity in traditional distributed databases can be ensured using well-known commit protocols. However, in an MDBS, the stronger autonomy requirements of the component local databases interfere with the cooperation necessary for these protocols to execute successfully [16]. Of particular concern is the fact that multidatabases cannot assume that all participating local database systems support a visible prepare-to-commit state for global subtransactions, where the ability to commit is guaranteed. In such situations, a local site (LS) that participates in a multidatabase environment may unilaterally abort a global subtransaction without agreement from the global level (termed *a local unilateral abort*). Consequently, the MDBS cannot ensure that subtransactions in multiple local sites will commit consistently with a global commit decision. Even if the local database systems are assumed to support a prepare-to-commit state, the potential blocking and long delays caused by such states severely degrade the performance.

Flexible transaction models proposed for the MDBS environment, such as Flex Transactions [4] and S-transaction [17], increase the failure resiliency of global transactions by allowing alternate 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 as long as either the primary subtransactions or the alternate subtransactions commit. The following example is illustrative:

Example 1 *Consider the following global transaction. A client at bank b_1 wishes to withdraw \$50 from her savings account a_1, and deposit it in her friend's checking account a_2 in bank b_2. If this is not possible, she will deposit the \$50 in her own checking account a_3 in bank b_3. With flexible 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: Deposit \$50 in checking account a_2 in bank b_2;
t_3: Deposit \$50 in checking account a_3 in bank b_3.

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. □

Since the flexible transaction model was proposed, much research has been devoted to its application [1, 8]. 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 formalize the model of flexible transactions and precisely define the semi-atomicity property. 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 [2]. 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 in specifying the structure of flexible transactions. We classify the set of recoverable flexible transactions that can be executed in an error-prone MDBS environment. We combine both forward and backward recovery techniques defined in [10] to eliminate the requirement of local prepare-to-commit states. The proposed 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 [5] 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. 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 [11]. [18, 3] enforce a global decision on the subtransactions by redoing or retrying them as necessary. [14, 9, 13] commit subtransactions locally before a global decision is made and rely on compensation when a global transaction is aborted. [10, 12] 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. 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 presented in Section 6.

2 A Formal Model of Flexible Transactions

Following [4], the definition of flexible transactions takes the form of a high-level applications description. Various applications semantics are captured in the flexible transaction definition, including commit and abort dependencies and the acceptable set of successful subtransactions. 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, we define precisely a flexible transaction model for global transactions.

From a user's point of view, a *transaction* is a sequence of actions performed 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.[1]

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

[1] This is necessary for the concurrency control of global transactions [6].

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 T_i.[2] Note that T_i and T_j may not be disjoint. Both precedence and preference relations are irreflexive and transitive. In other words, for each $t_i \in \mathcal{T}$, $\neg(t_i \prec t_i)$; 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 $T_i \rhd T_k$.

The precedence relation defines the correct parallel and sequential execution ordering dependencies among the subtransactions, while the preference relation defines the priority dependencies among alternate 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 precedence ($\prec$) and preference ($\rhd$) relations 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 a precedence relation $\prec_i$ defined on T_i. We then say that $(T_i, \prec_i)$ is a partial order of subtransactions. $(T_i, \prec_i)$ is a **representative partial order**, abbreviated as **$\prec$-rpo**, if the execution of subtransactions in T_i represents the execution of the entire flexible transaction $\mathcal{T}$. Clearly, if $(T_i, \prec_i)$ is a $\prec$-rpo, then there are no subsets T_{i1} and T_{i2} of T_i such that $T_{i1} \rhd T_{i2}$. Since each global transaction has at most one subtransaction at a local site, each $\prec$-rpo of a flexible transaction must have at most one subtransaction at a local site.

The structure of a flexible transaction $\mathcal{T}$ can thus be depicted as a set of $\prec$-rpos $\{(T_i, \prec_i), i = 1, ..., k\}$ of subtransactions, with $\bigcup_{i=1}^{k} T_i = \mathcal{T}$.[3] Note that $\mathcal{T}$ may contain more than one subtransaction at a local site, provided that they are in different $\prec$-rpos. Let $(T, \prec)$ be a $\prec$-rpo of $\mathcal{T}$. A partial order $(T', \prec')$ is a *prefix* of $(T, \prec)$, denoted $(T', \prec') \leq (T, \prec)$, if:

- $T' \subseteq T$;
- 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
- for each $t \in T'$, all predecessors of t in T are in T'.

A partial order $(T', \prec')$ is the prefix of $(T, \prec)$ with respect to $t \in T$, 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 $(T, \prec)$ with respect to $t \in T$, 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.

We now use prefixes and suffixes to show how a flexible transaction can switch from executing one $\prec$-rpo to executing a lower-priority alternative. Intuitively, if $\prec$-rpos $(T_i, \prec_i)$ and $(T_j, \prec_j)$ share some prefix and the subtransactions $t_1, \cdots, t_k$ immediately following that prefix in the execution of $(T_i, \prec_i)$ fail, then $(T_j, \prec_j)$ can continue execution from the point where the shared prefix completed. In this case, the set of $\{t_1, \cdots, t_k\}$ forms a *switching set*, formally defined as follows:

Definition 2 (Switching set) *Let $t_1, \cdots, t_k$ be subtransactions in $\prec$-rpo $(T, \prec)$ of a flexible transaction $\mathcal{T}$, with respective suffixes $(T_i, \prec_i) \geq (T, \prec)(t_i)$ for $i = 1, \cdots, k$. $\{t_1, ..., t_k\}$ forms a* switching set *of $(T, \prec)$ if*

- *there is a $\prec$-rpo $(T', \prec')$ of $\mathcal{T}$ such that $(T - (T_1 \cup ... \cup T_k), \prec'')$ is a prefix of $(T', \prec')$, and*
- *$(T_1 \cup ... \cup T_k) \rhd (T' - (T - (T_1 \cup ... \cup T_k)))$.*

A *switching point* is a subtransaction in a switching set which relates one $\prec$-rpo to another $\prec$-rpo.

Let $p_1 = (T_1, \prec_1)$ and $p_2 = (T_2, \prec_2)$ be two $\prec$-rpos of flexible transaction $\mathcal{T}$. We say that p_1 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 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 $\prec$-rpo $(T, \prec)$ of $\mathcal{T}$, $(T_1 \cup ... \cup T_k)$ where $(T_i, \prec_i) \geq (T, \prec)(t_i)$, for $i = 1, ..., k$, has no two alternatives with the same priority.
- None of the $\prec$-rpos $p_1, ..., p_l$ 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 $\prec$-rpos of a flexible transaction may not be clearly ranked, even if it is unambiguous. The aborting of subtransactions determines which

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

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

alternative $\prec$-rpo will be chosen. In the remainder of this paper, we assume that all flexible transactions are unambiguous.

So far, we have syntactically specified a flexible transaction as a set of alternate $\prec$-rpos of subtransactions that is determined by the 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 may describe the structure of flexible transaction $\mathcal{T}$ by an execution dependency graph, denoted $EDG(\mathcal{T})$. This is a directed graph whose nodes are all subtransactions of $\mathcal{T}$ labeled with the $\prec$-rpos that execute them and whose edges are all $t_i \rightarrow t_j$ $(t_i, t_j \in \mathcal{T})$, where t_i precedes t_j in $\mathcal{T}$ and there is no other subtransaction t_k which follows t_i and precedes t_j in that $\prec$-rpo. The following example is illustrative:

Example 2 *Consider a travel agent information system 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.

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

$p_1 = (\{t_1, t_3, t_4\}, \prec_1)$, $p_2 = (\{t_1, t_3, t_5\}, \prec_2)$,
$p_3 = (\{t_2, t_3, t_4\}, \prec_3)$, $p_4 = (\{t_2, t_3, t_5\}, \prec_4)$,

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

In each $\prec$-rpo of subtransactions, the value dependencies among operations in different subtransactions define data flow among the subtransactions. Let $(T, \prec)$ be a $\prec$-rpo and T 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 $t_{j_1} \rightarrow_v t_{j_t}$, $t_{j_2} \rightarrow_v t_{j_t}$, ..., $t_{j_{t-1}} \rightarrow_v t_{j_t}$, if the execution of one or more operations in t_{j_t} is determined by the values read by $t_{j_1}, ..., t_{j_{t-1}}$.

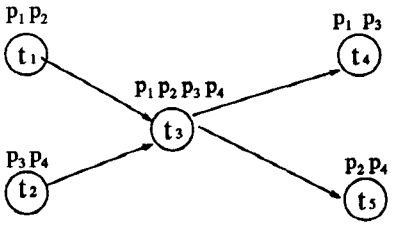


Figure 1: Execution dependency graph of Example 2.

We say that a database state is *consistent* if it preserves database integrity constraints. As defined for traditional transactions, the execution of a flexible transaction as a single unit should map one consistent multidatabase state to another. However, for flexible transactions, this definition of consistency requires that the execution of subtransactions in each $\prec$-rpo must map one consistent multidatabase state to another.

3 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 a termination operation (either a commit or an abort). 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 be completed. In the MDBS environment, this concept of atomicity has been relaxed to that of *semantic atomicity* [5]. 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 aborted. 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 $\prec$-rpos and results in the successful execution and commitment of the subtransactions in *one* of those $\prec$-rpos (which is termed *committed* $\prec$-rpo), the execution of a flexible transaction can proceed in several different ways. The subtransactions in different $\prec$-rpos may be attempted simultaneously, as long as any attempted subtransactions not in the committed $\prec$-rpo 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 3 (Semi-atomicity) *The execution of a flexible transaction T preserves the property of semi-atomicity if one of the following conditions is satisfied:*

- *All its subtransactions in one $\prec$-rpo commit and all attempted subtransactions not in the committed $\prec$-rpo are either aborted or have their effects undone.*
- *No partial effects of its subtransactions remain permanent in local databases.*

Such an extension of semantic atomicity allows different $\prec$-rpos of a flexible transaction to be attempted, possibly concurrently, while ensuring that the effects of either zero or exactly one $\prec$-rpo 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. We say that the execution of a $\prec$-rpo $(T, \prec)$ can *terminate* if its execution can move either forward to the commitment of its subtransactions or backward to the removal of any partial effects of the committed subtransactions. Obviously, if the execution of a flexible transaction is semi-atomic, then each $\prec$-rpo of the flexible transaction must terminate.

As local prepare-to-commit states are not presumed in the multidatabase system, we now investigate the preservation of the semi-atomicity of flexible transactions through a unification of the retry and compensation approaches. As pointed out in [10], in contrast to the redo technique [3], 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 implicit or explicit integrity constraints. For instance, a bank account usually has no upper limit, so a deposit action is retriable. However, it usually does have a lower limit, so a withdrawal action is not retriable.

A subtransaction is *compensatable* if the effects of its execution can be semantically undone after commitment by executing a compensating subtransaction at its local site. We assume that a compensating subtransaction ct_i for a subtransaction t_i will commit successfully if persistently retried.[4] ct_i must also be independent of the transactions that execute between t_i and ct_i. 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 must be able to 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 withdrawal 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 with ct_2 [7].

The set of compensating subtransactions that are executed for the subtransactions in each $\prec$-rpo 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.

A subtransaction t_i is a *pivot* subtransaction if it is neither retriable nor compensatable. For example, consider a subtransaction which reserves and pays for a non-refundable plane ticket. Clearly, this subtransaction is not compensatable. This subtransaction is also not retriable, since such a ticket might never be available.

[10] 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 this 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, we 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 section.

4 Constructing Recoverable Flexible Transactions

In this section, we present a formulation for flexible transactions that integrates the retry and compensation techniques to preserve the semi-atomicity of their execution.

[4] This requirement, termed *persistence of compensation*, has been discussed in the literature [5].

4.1 Commit Dependencies

We first examine the commit dependency relationships between any two subtransactions of a flexible transaction that must be obeyed in the commitment of these subtransactions. Let $\mathcal{T} = \{(T_i, \prec_i), i = 1, ..., k\}$ be a flexible transaction with 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 $(T_i, \prec_i)$ $(1 \leq i \leq k)$, then $t_i \rightarrow_c t_j$. These dependencies, which are determined by the execution control flow among subtransactions, are 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 the execution of a $\prec$-rpo can terminate, the commitment of compensatable subtransactions should always precede that of pivot subtransactions, which in turn should precede the commitment of retriable subtransactions. These dependencies are termed *t-commit dependencies.*

For those subtransactions which are retriable, value dependencies must be considered in determining a commitment order. For example, assume that a value written by subtransaction t_i at local site LS_1 is dependent on a value read by retriable subtransaction t_j at local site LS_2. If t_i commits and t_j aborts, then t_j should be retried. However, local transactions may be executed after the abort of t_j but before it is retried at LS_2. This 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 can 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. Such dependencies are termed *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 cyclic commit situation, where 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 $\prec$-rpo may render it difficult to determine a commit order among them which ensures that the execution of the $\prec$-rpo can terminate. In the remainder of this section, we examine those restrictions that need to be placed on flexible transactions to ensure semi-atomicity.

4.2 Well-formed Flexible Transactions

Let $(T, \prec)$ be a $\prec$-rpo in a flexible transaction. 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 its predecessors in T', where $(T', \prec') \leq (T, \prec)(t)$, to be no longer undoable. Among the pivot subtransactions[5] of $(T, \prec)$ for which all predecessors are compensatable, we identify one as the *critical point* of the $\prec$-rpo. The commitment of this subtransaction requires that the flexible transaction must commit, and the aborting of this subtransaction requires that either $(T, \prec)$ be switched to an alternative $\prec$-rpo or any partial effects of $(T, \prec)$ must be undone. Let $T' = \{t_1, ..., t_k\}$ be the set of all pivot subtransactions of $(T, \prec)$ for which all predecessors are compensatable. The *critical point* of $(T, \prec)$ is determined through the following rules:

- if t is the only pivot subtransaction in T', then t is the critical point; otherwise
- if there is a $t \in T'$ which has been chosen as the critical point for a higher-priority $\prec$-rpo, then choose t as the critical point; otherwise
- if there is a $t \in T'$ which is not a switching point, then choose t as the critical point; otherwise
- a subtransaction in T' is randomly chosen as the critical point.

Any compensatable subtransactions that do not follow pivot or retriable subtransactions can commit before the critical point, and any retriable subtransactions which are not ordered with the critical point can commit after the critical point. However, once the critical point commits, the aborting of any pivot subtransaction which is not the critical point and of any compensatable subtransaction which follows a pivot or retriable subtransaction in $(T, \prec)$ may hamper $(T, \prec)$ in termination. Such problematic subtransactions are termed *abnormal subtransactions.* More precisely:

Definition 4 (Abnormal subtransaction) *Let $(T, \prec)$ be a $\prec$-rpo in a flexible transaction. A subtransaction t in T is* abnormal *if one of the following conditions is satisfied:*

- *t is compensatable and there is a pivot or retriable subtransaction t_1 in T such that $t_1 \prec t$; or*

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

- *t is a pivot subtransaction but is not the critical point.*

Note that only a compensatable or pivot subtransaction may be an abnormal subtransaction.

Clearly, 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, and no compensatable subtransaction can commit after any pivot or retriable subtransactions. 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 permits 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, T can be committed. Thus, abnormal subtransactions can be permitted in flexible transactions. However, if any abnormal subtransaction aborts, appropriate actions must ensue to continue the execution of the flexible transaction.

Let T be a flexible transaction and $(T_i, \prec_i)$ be a $\prec$-rpo of 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 $t_1 \prec t_2 \prec t$.

Intuitively, we define an abnormal subtransaction t to be a *blocking point* in $(T_i, \prec_i)$ if it identifies a place at which the aborting or compensation of t must result in the switching of the execution of $(T_i, \prec_i)$ to an alternative $\prec$-rpo. Formally:

Definition 5 (Blocking point) *Let $(T_i, \prec_i)$ be a $\prec$-rpo in a flexible transaction. An abnormal subtransaction t in T_i is a* blocking point *if it has no predecessors $t_1, ..., t_k$ that have the following properties:*

- *$t_1, ..., t_k$ are members of a switching set SW, and*
- *all members of SW and their successors that are not in T_i', where $(T_i', \prec_i') \geq (T_i, \prec_i)(t)$, are compensatable, and*
- *all subtransactions in the alternative to the suffixes of $(T_i, \prec_i)$ with respect to members in SW are either retriable or abnormal.*

The following example is illustrative:

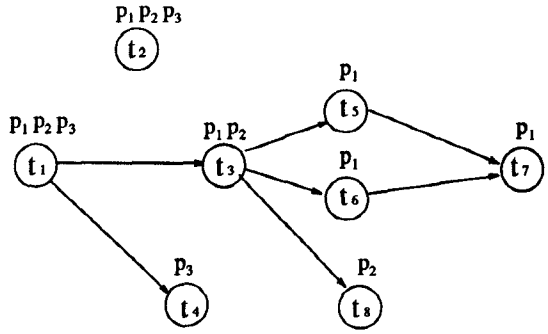


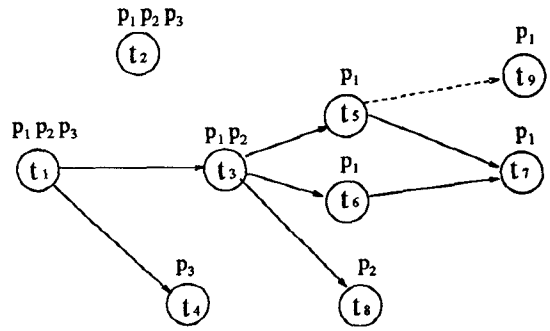
Figure 2: Execution dependency graph of Example 3.

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

$p_1 = (\{t_1, t_2, t_3, t_5, t_6, t_7\}, \prec_1)$,
$p_2 = (\{t_1, t_2, t_3, t_8\}, \prec_2)$,
$p_3 = (\{t_1, t_2, t_4\}, \prec_3)$,

where $\{t_5, t_6, t_7\} \rhd \{t_8\}$, $\{t_3, t_8\} \rhd \{t_4\}$, and $\{t_3, t_5, t_6, t_7\} \rhd \{t_4\}$. By the critical point determination rules, t_2 is the critical point for all three $\prec$-rpos, because there is an alternative if pivot subtransaction t_3 fails. Thus, t_3, t_5, t_6, t_7 are abnormal subtransactions; among them, t_3, t_5, and t_6 are the blocking points. Clearly, $\{t_3\}$ and $\{t_5, t_6\}$ are the switching sets of p_1, and $\{t_3\}$ is the switching set of p_2. □

Note that, in Example 3, t_7 is not a blocking point. When t_7 aborts, the execution of p_1 can switch to p_2 by compensating the effects of t_5 and t_6. Now, let us consider a variation of Figure 2 in which another pivot subtransaction t_9 is the successor of t_5, as shown in the following figure:



In this situation, t_7 becomes a blocking point. If t_9 commits but t_7 aborts, it is then impossible for T to either switch from p_1 to another $\prec$-rpo or to have the partial effects of p_1 undone.

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

Definition 6 (Well-formed flexible transaction) *A flexible transaction $\mathcal{T}$ is* well-formed *if, for each abnormal subtransaction t that is a blocking point in a $\prec$-rpo $(T_i, \prec_i)$ in $\mathcal{T}$, the following conditions are satisfied:*

- *t is a member of some switching set SW such that the other members in SW are compensatable, and*
- *if a successor of a member in SW is not ordered by $\prec_i$ relation with another member in SW, then it is compensatable, and*
- *all subtransactions in the alternative to the suffixes of $(T_i, \prec_i)$ with respect to members in SW are either retriable or abnormal.*

Thus, in a well-formed flexible transaction, for any $\prec$-rpo $(T_i, \prec_i)$ which contains an abnormal subtransaction t, there is at least one alternate $\prec$-rpo $(T_j, \prec_j)$ which shares a prefix with $(T_i, \prec_i)$ such that the aborting of t will lead the execution of $\mathcal{T}$ from $(T_i, \prec_i)$ to $(T_j, \prec_j)$ without resulting in any database inconsistency. Note that, following Definition 6, a switching set can have at most one blocking point that is a pivot subtransaction.

Observation 1 *If a flexible transaction is well-formed and contains a finite number of subtransactions, it has at least one $\prec$-rpo p that has no abnormal subtransactions, and no other $\prec$-rpo has lower priority than p.*

In Example 2, the flexible transaction $\mathcal{T}$ has two $\prec$-rpos p_2 and p_4 that contain no abnormal subtransactions. 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 $\prec$-rpo p_3 that contains no abnormal subtransactions. Since the blocking point t_3 constitutes the switching set and the blocking points t_5 and t_6 also constitute the switching set, $\mathcal{T}$ is also well-formed.

4.3 Semi-atomicity for Recoverability of Flexible Transactions

We now discuss the preservation of semi-atomicity. Following [2], 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 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 7 (Commit dependency graph) *A commit dependency graph of a well-formed flexible transaction $\mathcal{T} = \{(T_i, \prec_i), i = 1, ..., k\}$, 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) $t_1 \prec_i t_2$ $(1 \leq i \leq k)$; or*
- *(v-commit dependency) $t_1 \rightarrow_v t_2$ and t_1 is retriable; or*
- *(t-commit dependency) t_1 is compensatable but not abnormal 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 8 (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: Omitted due to space constraints. □

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 9 (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 which participate in different $\prec$-rpos 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: The proof can proceed by induction on a number n of $\prec$-rpos of $\mathcal{T} = \{(T_i, \prec_i)$, i=1,...,n$\}$. The detailed proof is omitted due to space constraints. □

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 extends the scope of global transactions that can be specified in the MDBS environment beyond that of the basic global transaction model. 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

We now present a commit protocol for flexible transactions that is based upon Theorem 1 and that maintains semi-atomicity as defined in Definition 3.

5.1 System Model

The system model employed for this protocol is shown in Figure 3. We assume that the GTM submits flexible transaction operations to the local databases through servers that are associated with each LDBS. This model assumes that, at any time, only one $\prec$-rpo of each flexible transaction is executing.[6] The local databases may execute their own local transactions independently.

In the flexible transaction definition, we assume that each flexible transaction contains a finite number of subtransactions and 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 flexible transactions are well-formed. Also, a commit dependency graph is built for each flexible transaction and is checked for cycles. If a flexible transaction is 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.

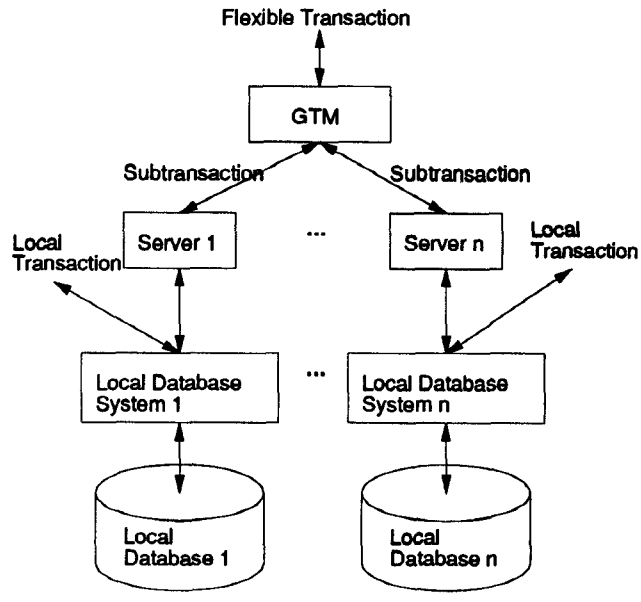


Figure 3: The multidatabase system model.

5.2 The Commit Protocol

In our optimistic 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 execution. This is needed to deal with 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 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.

We assume that the GTM maintains state information for each subtransaction in the commit dependency graph of its flexible transaction. This state may be one of the following cases:

- **inactive**: if the subtransaction has not started its execution.
- **active**: if the subtransaction has started but not completed its execution.[7]
- **to_be_committed**: if the subtransaction has completed its execution.
- **committed**: if the subtransaction has committed.
- **aborted**: if the subtransaction has aborted.
- **committed-reversed**: if both the subtransaction and its compensating subtransaction have committed.

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

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

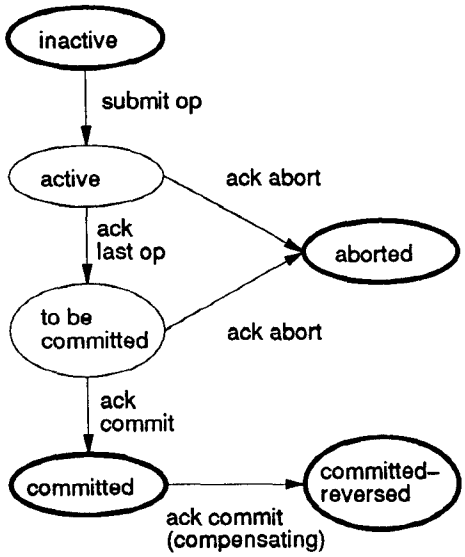


Figure 4: State transitions for a subtransaction in a flexible transaction.

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 it 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 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 it 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 $\prec$-rpo of the flexible transaction. Such an alternative may be obtained directly, or one may be constructed by finding alternatives of more remote pre-

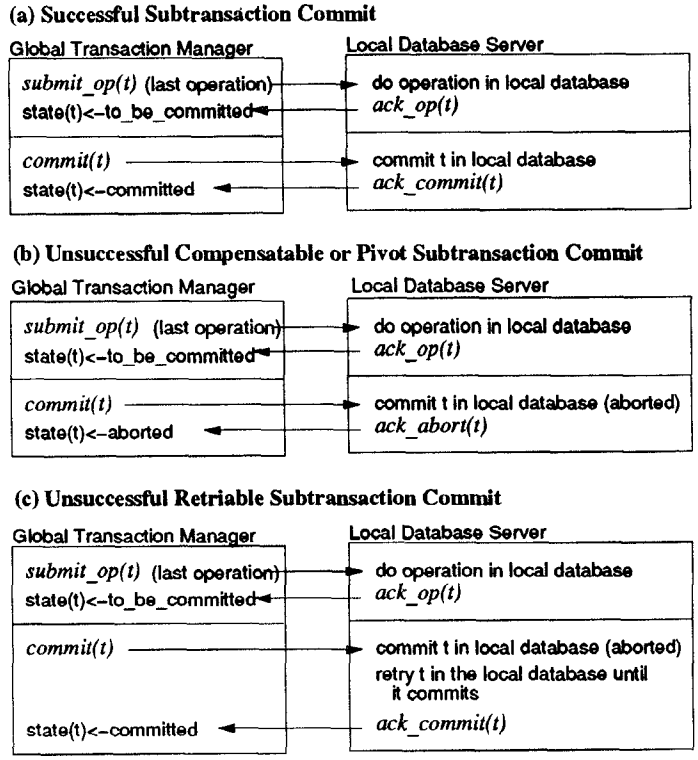


Figure 5: Message passing sequences for committed and aborted subtransactions.

decessors and trying from an earlier 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 has the fewest committed successors. Otherwise, find the closest predecessor t_1 in the $\prec$-rpo that is a switching point and then find the switching set inclusive of 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. Compensate for all subtransactions in the switching set and their successors that are in the committed state. Attempt the next untried alternative in preference order.*
3. *If no switching set is found, abort the flexible transaction by aborting all subtransactions in the $\prec$-rpo that are in the active state. Compensate for all subtransactions in the $\prec$-rpo that are in the committed state.*

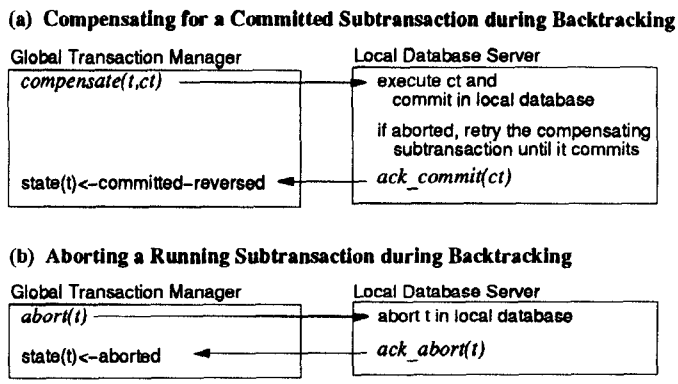


Figure 6: Message passing sequences used during backtracking: (a) when compensation is required; (b) when aborting is possible.

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

5.3 Discussion

The flexible transaction commit protocol preserves the semi-atomicity of 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: Omitted due to space constraints. □

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 a 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 made to the *aborted* state.

In the absence of failures, the proposed protocol requires $2n$ messages that need to be exchanged to reach a commit decision and at least 6 rounds as compared to either $3n$ messages and and 3 rounds needed by the 2PC protocol [2] or $2n$ messages and exactly 6 rounds needed by the GTM commit protocol proposed in [10], where n is the number of local sites.

Both the GTM commit protocol described in [10] and the flexible transaction commit protocol outlined here prevent the severe blocking of local transactions that may be caused by the 2PC protocol. The flexible transaction commit protocol permits each subtransaction to commit dynamically without waiting for other subtransactions of the same flexible transaction to complete their execution. Thus, it generates even less blocking of local transactions than does the GTM commit protocol developed in [10]. 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 performance for the more restricted class of input global transactions to which it applies.

6 Conclusions

Global transaction management in an error-prone MDBS environment has been recognized as an yet unresolved issue in those instances in which the component local database systems do not support prepare-to-commit states [15]. 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. 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 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 proposed theory and protocol are applicable to the traditional distributed database environment. Using flexible transactions, blocking that may be caused

by the 2PC protocol can be prevented. 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 [19].

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*, pages 65–76, Vancouver, Canada, 1992.

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

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

[4] A. K. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A Multidatabase Transaction Model for InterBase. In *Proceedings of the 16th VLDB*, pages 507–581, Brisbane, Australia, 1990.

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

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

[7] H. Korth, E. Levy, and A. Silberschatz. A Formal Approach to Recovery by Compensating Transactions. In *Proceedings of the 16th VLDB*, Brisbane, Australia, 1990.

[8] 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. 1992.

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

[10] 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.

[11] 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.

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

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

[14] W. Perrizo, J. Rajkumar, and P. Ram. HYDRO: a heterogeneous distributed database system. In *Proceedings of ACM SIGMOD*, pages 32–39, Denver, Colorado, May 1991.

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

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

[17] 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.

[18] 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.

[19] A. Zhang and B. Bhargava. Scheduling with Compensation in Multidatabase Systems. Technical Report CSD-TR-93-063, Purdue University, October 1993.