An Optimistic Commit Protocol for Distributed Transaction Management*

Eliezer Levy　Henry F. Korth　Abraham Silberschatz

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712 USA

Abstract. A major disadvantage of the two-phase commit (2PC) protocol is the potential unbounded delay that transactions may have to endure if certain failures occur. By using compensating transactions, we obtain a revised 2PC protocol that overcomes these difficulties. In the revised protocol, locks are released as soon as a site votes to commit a transaction, thereby solving the indefinite blocking problem of 2PC. If finally the transaction is to be aborted, then its effects are undone semantically using a compensating transaction. Therefore, Semantic, rather than standard, atomicity is guaranteed. Relaxing standard atomicity interacts in a subtle way with correctness and concurrency control issues. Accordingly, a correctness criterion is proposed that is most appropriate when atomicity is given up for semantic atomicity. The correctness criterion reduces to serializability when no global transactions are aborted, and excludes unacceptable executions when global transactions do fail. We devise a family of practical protocols that ensure this correctness notion. These protocols restrict only global transactions, and do not incur extra messages other than the standard 2PC messages.

1 Introduction

The most common protocol for ensuring atomicity of multi-site transactions in a distributed environment is the two-phase commit (2PC) protocol [Gra78]. Typically, the 2PC protocol is combined with the strict two-phase locking protocol [BHG87], as the means for ensuring the atomicity and the serializability of transactions in a distributed database (e.g., [MLO86]). The implications of this combination on the length of time a transaction may be holding locks on various data items might be severe. At each site, and for each transaction (at least all exclusive) locks must be held until either a commit or an abort message is received from the coordinator of the 2PC protocol. Since the 2PC protocol is a *blocking* protocol [Ske82], the length of time these locks are held can be unbounded. Moreover, even if no failures occur, since the protocol involves three rounds of messages (request for vote, vote and decision) the delay can be intolerable.

The impact of indefinite blocking and long-duration delays is exacerbated in multidatabase systems — a specific type of distributed database system where several *heterogeneous* and *autonomous* database management systems (DBMSs) are integrated to enable the processing of multi-site, or *global*, transactions [hdb90]. Global transactions are processed by decomposing them into local subtransactions that are executed at the different sites. The integrated DBMSs may belong to distinct, and possibly competing business organizations (e.g., competing computerized reservation agencies). Therefore, the preservation of local *autonomy* of the individual DBMSs is crucial. It is undesirable, for example, to use a protocol where a site belonging to a competing organization can harmfully or mistakenly block the local resources; a phenomenon that can occur under the 2PC protocol. Additionally, one of the flavors of local autonomy is defined as the capability of a site to abort any local (sub)transaction at any time before the (sub)transaction terminates [BST90]. Employing the 2PC protocol, a site enters a *prepared* state if it votes to commit a transaction T_i. Once in this state, a site becomes a subordinate of the external coordinator, and it can no longer unilaterally determine the fate of the local subtransaction of T_i.

It is impossible to have a non-blocking commit protocol that is immune to both site and link failures [BHG87]. In this paper, we introduce an alternative to the standard 2PC protocol that alleviates the virtually inevitable blocking and lengthy delays problems yet preserves autonomy. The key concept behind this protocol is the use of compensating transactions [KLS90a].

*This research was partially sponsored by NSF Grants IRI-8805215 and IRI-9003341, and grants from Unisys Roseville Operations and the IBM Corporation.

The new protocol is applicable whenever the problem of atomicity of a multi-site transaction surfaces. It can benefit distributed database systems in general and multidatabase systems in particular.

The protocol guarantees *semantic atomicity* [GM83] rather than the standard all-or-nothing atomicity. A salient contribution of this paper is the examination of the consequences of the protocol and this relaxed atomicity notion in terms of serializability, and correctness in general. The importance of our study of correctness issues is underlined by the growing popularity of advanced transaction models that are based on semantic atomicity [GM83, GMS87, AGMS87, KR88, Reu89, Vei89, GMGK+90], and by the lack of specific correctness criteria in this domain.

The remainder of the paper is organized as follows. Section 2 provides an operational overview of the basic protocol. Several techniques and assumptions that we use are clarified in Section 3. In Section 4, we discuss the need for a new correctness criterion. Section 5 presents the correctness criterion and a sufficient condition for ensuring it. In Section 6, based on this condition, another component of the protocol is presented, whose task is to ensure the correctness criterion.

2 The O2PC Protocol

In the standard 2PC protocol, a multi-site transaction is associated with a coordinator that initiates the protocol by sending a VOTE-REQ message (also referred to as PREPARE message) to all participating sites. Upon receipt of this message, a participating site votes (by sending a VOTE message back to the coordinator) either to commit the particular transaction or to abort it. Based on these votes, the coordinator decides whether to commit or abort the transaction. Only if all the votes are to commit then the transaction is to be committed. Following this, the coordinator transmits its DECISION message to the participating sites.

As was mentioned earlier, global serializability is obtained by combining the 2PC protocol with a 2PL discipline. This combined protocol is referred to in [BHG87] as *distributed 2PL* and it works as follows. It is assumed that the coordinator of T_i initiates the 2PC protocol only after it has received acknowledgements for all of T_i's operations. Therefore, when the coordinator initiates the 2PC protocol by sending the VOTE-REQ messages, T_i has surely obtained all the locks it will ever need. Locks are released only after the VOTE-REQ message has been received. Consequently, it is guaranteed that a transaction releases a lock at any site only after it has finished acquiring locks at all sites. Since a two-phased locking discipline is enforced, the distributed 2PL protocol guarantees serializability globally. For the well known reasons of avoiding cascading aborts, and use of state-based recovery, the exclusive (i.e., write) locks are released only after the decision message is received locally. Thus, a *strict* version of 2PL is used. It is possible to release the shared (i.e., read) locks as soon as the VOTE-REQ message is received.

Holding the locks until a DECISION message is received, which is the cause of blocking, is necessary only if the transaction at hand has to be aborted. Our revised protocol is based on the *optimistic assumption* that in most cases the protocol terminates successfully (i.e., the transaction commits) and therefore the locks can be released earlier. This can dramatically reduce waiting due to data contention, thereby improving the performance of the system. Such an assumption is valid in most practical distributed environments. Furthermore, since the commit protocol is initiated only when the transaction at hand has already obtained all its locks and completed all its operations, its failure is very unlikely. The validity of the optimistic assumption is orthogonal to the protocol correctness. However, if the assumption is unfounded, the overhead incurred by the protocol is likely to outweight its benefits.

The *optimistic 2PC (O2PC)* protocol is a slightly modified version of the distributed 2PL protocol. The same message exchange is carried out as in the standard protocol. If a site votes to abort T_i, then as in the standard protocol, an abort vote is sent back to the coordinator, and the locks held by the transaction are released as soon as the transaction is locally undone (rolled-back). However, if a site votes to commit T_i, *all locks held by T_i are released at once, without waiting for the coordinator's final commit or abort message.* In this case, we say that T_i is *locally-committed* at that particular site. Observe that a global 2PL discipline is preserved, even under the early lock release provision of the O2PC protocol.

The uncoordinated local commitment resulting in the early release of locks is the crux of the protocol. On the one hand, the early release of locks solves the problems of blocking and the local commitment keeps the sites autonomous. On the other hand, the uncoordinated commitment of updates may violate the standard all-or-nothing atomicity guarantee of a transaction, if at least one of the sites votes to abort it. A situation may arise where, at some sites T_i is locally committed, whereas at some other sites T_i is aborted. In this case, the effects of T_i must be undone at sites where it is locally-committed. Undoing the effects of a locally-committed T_i is problematic, if not impossible, using standard recovery techniques.

The key to an adequate solution is the notion of *compensating transactions.* Compensating transactions are intended to handle situations where it is required to undo a transaction whose updates have been read by other transactions, without resorting to cascading aborts. The concept of compensation is formally defined in [KLS90a] and the essential details are reviewed in Section 3.2.

We propose to use compensating transactions, in conjunction with the O2PC protocol, as the means for ensuring transaction atomicity despite of the uncoordinated commitment of updates at different sites. After voting to commit T_i, a site still carries on with the second phase of the regular 2PC protocol (despite having released the locks held by T_i). If the site receives a decision message from the coordinator to abort T_i, then it invokes the corresponding compensating transaction. Since it is quite likely that the decision would be to commit T_i, the gain by the early release of locks should outweigh the overhead associated with those cases requiring compensation for T_i.

Instead of the familiar all-or-nothing semantics, the protocol ensures a similar, though weaker, atomicity guarantee referred to as *semantic atomicity*. Semantic atomicity states that when a multi-site transaction is decomposed into a set of single-site subtransactions, either all subtransactions are locally-committed (and then the entire transaction is committed), or all locally-committed subtransactions are compensated-for and all other subtransactions are rolled-back in the standard manner.

We note that not all transactions are compensatable. Transactions involving *real actions* [Gra81] (e.g., firing a missile or dispensing cash) may not be compensatable. The adjustment for transactions involving non-compensatable actions is simply to retain the locks and delay real actions until a commit message is received (as in distributed 2PL) in all sites performing these actions. All other sites running subtransactions on behalf the multi-site transaction can still benefit from the early lock release.

3 Transaction Structure

In order to proceed we must first introduce some assumptions and terminology concerning transaction structure that are used in our exposition.

3.1 Transaction Management

We distinguish between *local* and *global* transactions. A local transaction accesses data at a single site, whereas a global transaction accesses data located at two or more sites. A global transaction T_i that requires access to data located at sites $S_1, S_2, \ldots, S_k$ is submitted for execution as a collection of local subtransactions $T_{i1}, T_{i2}, \ldots, T_{ik}$, where T_{ij} is executed at site S_j. We make a distinction between a local subtransaction that is executed on behalf of a global transaction, and an independent local transaction. Local transactions (i.e., non-subtransactions) follow the strict 2PL protocol.

This abstraction of global transactions as a set of local subtransactions is most appropriate for understanding our protocol and its formal properties. However, several comments concerning some practical issues are in order.

The decomposition of a global transaction into local subtransactions conforms to one of two models. In the first model, all the requests of a global transaction to a particular site constitute the local subtransaction at that site. Each subtransaction can be viewed as an arbitrary collection of reads and writes against the local data. That is, no predefined semantics is associated with a subtransaction. This model is elaborated in [CP87] and is the standard model in distributed databases. Henceforth, this model is referred to as the *generic model*. The generic model is also considered as the general framework in the multidatabases context [BS88, BST90].

An alternative model is one in which each global transaction is decomposed into a possibly structured collection of local subtransactions, each of which performs a semantically coherent task. The subtransactions are selected from a well-defined repertoire of operations (i.e., subtransactions) forming an interface at each site in the distributed system. This model is referred to as the *restricted model*, hereafter, and is suitable for a federated distributed database environments [fdb87]. The distinction between the two models becomes relevant once compensating transactions are introduced. Our work applies to both models; however, fitting the ideas in each framework is bound to be different, and probably easier in the restricted model as we explain later.

3.2 Compensating Transactions

The *compensating transaction* that is specific to the *forward* transaction T_i is denoted by CT_i. CT_i undoes T_i's effects semantically without causing the cascading aborts of transactions that have read data updated by T_i. The intention is to leave the effects of transactions that read from T_i intact, yet preserve database consistency. Therefore, compensation for T_i does not guarantee the physical undoing of all the direct and indirect effects of T_i. The state of the database after compensation took place may not be identical to the state that would have been reached, had T_i never taken place. Compensation does guarantee, however, that a *consistent* state is established based on *semantic* information. In [KLS90a] we formally characterize the outcome of compensation based on the properties of T_i and on properties of transactions that have read from T_i. By the nature of compensation, CT_i is always serialized to come after the corresponding T_i (though not necessarily immediately after).

It is guaranteed that once compensation is initiated, it completes successfully. This stringent requirement is referred to as *persistence of compensation* and is recognized in [GMS87, GM83, Vei89, GMGK$^+$90, Reu89, KLS90a]. The rationale behind the persistence of compensation requirement is the need to preserve (semantic) atomicity. Initiating a compensating transaction parallels a decision to abort the transaction in the traditional setting — definitely a non-reversable decision. Observe that persistence of compensation implies that there is no need to use a commit protocol to ensure the atomicity of a compensation transaction in a distributed environment.

In the context of the O2PC protocol, compensation is employed as follows. If T_i is a global transaction, CT_i is also a global transaction that consists of $CT_{i1}, CT_{i2}, \ldots, CT_{ik}$ of local *compensating subtransactions*, one for each site where T_i was executed. Each compensating subtransaction is submitted for execution at a site just like any other local transaction, and hence it is subject to the local concurrency control.

Consider a global transaction T_i that is locally-committed at some sites, whereas other sites have voted to abort it. At a site S_j where T_i is locally-committed, CT_{ij} is a special compensation action, since T_i's updates have been exposed. At a site S_k which voted to abort T_i, the local subtransaction T_{ik} is automatically rolled-back using standard recovery techniques (e.g., undo from log). We model undoing a transaction using the standard roll-back recovery, as a special case of a compensating transaction where there are no transactions that have read from the undone transaction [KLS90a]. Thus, a global CT_i is a blend of standard roll-backs at sites having voted to abort T_i, and actual compensating subtransactions at sites having voted to commit T_i.

It should be recognized that in a system conforming to the restricted model it is easier to apply compensation techniques than in the generic model. In the restricted model, since each subtransaction performs a semantically coherent task, supplying a counter-task can be done in advance and should not be that hard (e.g., a DELETE as compensation for an INSERT subtransaction).

With respect to locking, compensating subtransactions are treated as local transactions rather than as subtransactions of global transactions. That is, they also follow strict 2PL locally. The reasons for this important decision are elaborated in the next section. The key point to note is that at each site, the local execution over local transactions, subtransactions, and compensating subtransactions is guaranteed to be serializable.

4 Correctness Issues

The local uncoordinated commitment and the use of compensating transactions in the O2PC protocol pose some interesting questions regarding concurrency control and correctness issues. In contrast to the traditional serializability theory which deals only with committed projections of histories [BHG87], the theoretic and modeling tools (e.g., serialization graphs) used here, must account for failed transactions and their corresponding compensating transactions. One might be tempted to impose serializability over all transactions, including compensated-for and compensating transactions, as the correctness criterion. Compensating transactions, however, possess several special properties that render this extended serializability notion both unattainable and inappropriate:

- Persistence of compensation means that a compensating transaction has a simplified atomicity notion — it can only commit. Consequently, there is no need to use a commit protocol for compensating transactions in a distributed environment. Avoiding the use of 2PC to terminate global compensating transactions is critical, since there is no chance to couple locking decisions with the commit protocol messages, as it is done in the distributed 2PL and O2PC protocols.

- A second problem regarding the scheduling of compensating transactions stems from the fact that in a site S_k that votes to abort a transaction T_i, the standard roll-back of T_{ik} is considered as a compensating subtransaction; that is, as CT_{ik}. Such roll-backs are automatic and uncoordinated with the initiation and termination of other compensating subtransactions of the same transaction. This is especially true in a multidatabase environment, where the preservation of local autonomy dictates that no constraint can be placed on the automatic local roll-back. Therefore, serializability of histories with compensating transaction may be again jeopardized since the global scheduling of compensating transactions is not regulated and coordinated. For example, rolling-back a subtransaction T_{ij} as part of CT_i and releasing locks once the roll-back is complete, violates the 2PL rule for CT_i as a whole.

- We observe that at least in the restricted model the executions of the compensating subtransactions are semantically independent. That is, there should be no value dependencies [DE89] among the different subtransactions. A compensating transaction in the restricted model can be viewed as a *set* of semantically unrelated subtransactions. This argument and the previous points give the impression that the execution of a compensating subtransactions is somewhat independent from the execution of its sibling compensating subtransactions. This observation is underlined once it is realized that compensation, as a recovery activity, is an *after the fact* activity. That is, the forward transaction has executed, and compensation is carried out based on its effects. In support of our observation, we cite [Vei89, map89] where a large-scale, commercial application that is predicated on this independence of compensating subtransactions, is described.

The reasons given above suggest that compensating subtransactions should release their locks once they complete their local processing, regardless of the execution of their sibling compensating subtransactions. As a result, serializability may be lost. That is, a global serialization graph may contain cycles with compensating transactions. The independence of the subtransactions of a compensating transaction implies that it need not see a globally consistent state as a global transaction. Therefore, cycles whose only global transactions are compensating transactions do not introduce inconsistencies in the database, and should be thus allowed.

Another important consideration in designing an alternative correctness criterion is the following requirement. A transaction either reads a database state affected by T_i (and not by CT_i), or it reads a state reflecting the compensatory actions of CT_i. However, a transaction should never read both uncompensated-for updates of T_i as well as data items already updated by CT_i. This important constraint is referred to as *atomicity of compensation* in [KLS90a] and is elaborated in [Lev90].

Our intention is to propose a revised correctness criterion that takes into account the special properties of compensating transactions and guarantees atomicity of compensation. In the next section, we formally present our correctness criterion.

Before we proceed, we note that the of loss of serializability would not be worrisome if sagas [GMS87], or their generalization — multi-transactions [GMGK+90, KR88, Reu89] are used. Then the O2PC scheme can be employed as it was presented so far, without any further adjustments. The rest of the paper, however, addresses the problems of loss of serializability and correctness.

5 Theoretical Results

Our correctness criterion is stated in terms of serialization graphs (SGs) that are a slightly extended version of the standard SGs [BHG87]. For brevity, we omit the underlying concept of complete histories that is identical to the definition given in [BHG87].

Let $\mathcal{T}$ be a set of global transactions $\{T_1, T_2, \ldots, T_n\}$, and let $\mathcal{CT}$ be the set of the corresponding compensating transactions $\{CT_1, CT_2, \ldots, CT_n\}$. Also, let $\mathcal{L}$ be a set of local transactions $\{L_1, L_2, \ldots, L_m\}$. The *local serialization graph* for a complete local history H over $\mathcal{T}, \mathcal{CT}$ and $\mathcal{L}$ is a directed graph SG(H)=(V, E). The set of nodes V consists of transactions in $\mathcal{T} \cup \mathcal{CT}$ and the committed transactions in $\mathcal{L}$. The set of edges E consists of all $A_i \rightarrow B_j$, $A_i, B_j \in \mathcal{T} \cup \mathcal{CT} \cup \mathcal{L}$, such that one of A_i's operations precedes and conflicts with one of B_j's operations in H.

A *global SG* is an SG that corresponds to a history at more than one site. The SG of site a is denoted SG_a. Given a set of local SGs, each represented as $SG_a = (V_a, E_a)$, the corresponding global SG is defined as $SG_{global} = (\cup\, V_a, \cup\, E_a)$.

Intuitively, a history H is *correct* if the global SG(H) is acyclic, except for cycles that consist of at least one compensating transaction and (potentially) local transactions.

The main result of this section is the derivation of a sufficient condition for obtaining the correctness criterion and atomicity of compensation. The strategy in obtaining the main result is summarized as follows:

- We identify the types of cycles that are *not* allowed in global SGs, namely *regular cycles*, and state the correctness criterion formally (Lemma 1).
- We show that if a regular cycle exists in the global SG then certain conditions, called the *cycle conditions*, are implied (Lemma 2).
- We introduce properties of SGs, called *stratification properties* whose negation is implied by the cycle conditions (Lemma 3).
- We conclude in Theorem 1 that by ensuring the stratification properties, regular cycles are avoided.
- Theorem 2 identifies the type of compensating transactions for which atomicity of compensation is guaranteed by preventing regular cycles.

To present our results, we must first establish some notation. We use capital letters at the beginning of the alphabet (e.g., A, B, C) to denote either compensating or regular global transactions. Also, for a particular history H, the notation $A \rightharpoondown B$ is used to denote that there is a directed path (of arbitrary length) between the two transaction nodes in $SG(H)$.

We define *local* and *global paths* to be paths (entirely) within a local SG and global SG, respectively. When specifying a local path, the local SG it belongs to, is also specified.

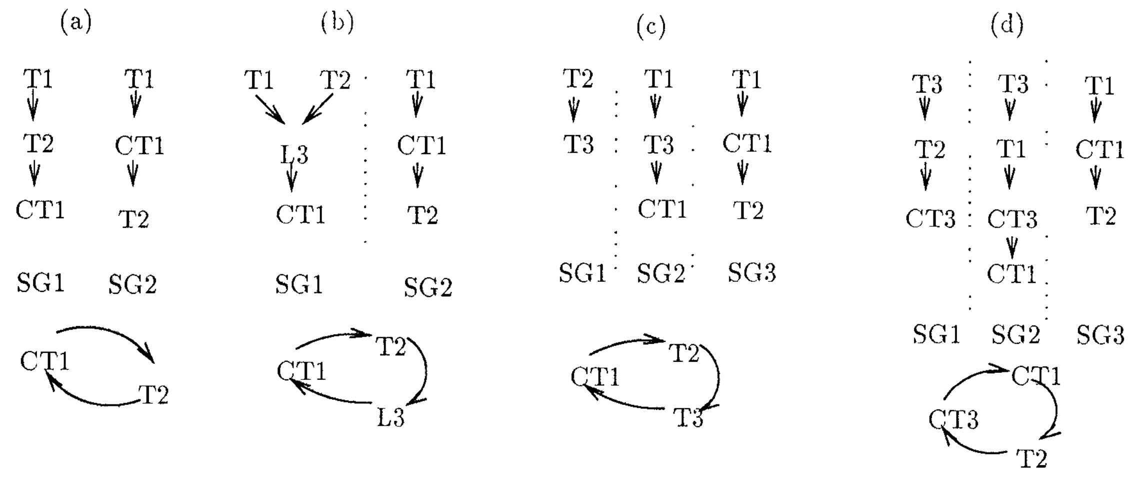

Figure 1: Regular Cycles

When considering global paths it is useful to segment such paths into local paths, and represent each such local path by its end points. For example, consider the paths $A \rightarrow B$ in SG_1, and $B \rightarrow C \rightarrow D$ in SG_2. The global path $A \rightarrow D$ is represented by the local paths $A \rightarrow B$ in SG_1 and $B \rightarrow D$ in SG_2.

Thus, a *representation* for a given global path lists the local paths constituting the global path in order. This representation is not necessarily unique. A *minimal representation* for a given global path is the path representation with the minimal number of local segments (paths). Again, this representation is not necessarily unique. Accordingly, when we say that a global path *includes A*, we mean that A appears on one of the minimal representations of this path.

Example 1. Consider the following local paths:

$CT_1 \rightarrow T_2$ in SG_1
$CT_1 \rightarrow T_2 \rightarrow CT_3$ in SG_2
$CT_3 \rightarrow CT_1$ in SG_3

Consider the global path $CT_1 \rightarrow CT_3$. It has two representations:

1. $CT_1 \rightarrow T_2$ in SG_1; $T_2 \rightarrow CT_3$ in SG_2
2. $CT_1 \rightarrow CT_3$ in SG_2

The latter being the minimal representation. The global path $CT_1 \rightarrow CT_3$ does not include T_2. ◇

A *regular* cycle is a global cyclic path in a global SG that includes at least one regular global transaction. Observe that there are no regular cycles in Example 1. Figure 1 demonstrates several regular cycles, by presenting the corresponding segments of the local SGs.

Lemma 1. *Any regular cycle includes at least one compensating transaction.* □

Our correctness criterion states that a *history H is correct if, and only if, SG(H) contains no regular cycles and no local cycles.* Since we assume that local histories are serializable, and hence there are no local cycles, we focus on preventing regular cycles to ensure correctness.

Lemma 2. *If there exists a regular cycle in a global SG, then the following* cycle conditions *hold:*

C1. *There exist distinct global transactions T_i and T_j such that $CT_i \rightarrow T_j$ at some SG_a, and at some other SG_b where T_j appears, either $T_j \rightarrow CT_i$, or there is no local path between T_i and T_j in SG_b.*

C2. *There exist distinct global transactions T_i and T_j such that $T_j \rightarrow CT_i$ at some SG_a, without having T_i on that path, and at some other SG_b where T_j appears, either $CT_i \rightarrow T_j$, or there is no local path between T_i and T_j in SG_b.* □

For the purpose of avoiding regular cycles we need to identify the pairs of transactions that can cause the formation of such cycles. Intuitively, regular cycles may be formed when T_2 follows another T_1 in the SG before the latter transaction is globally committed or fully compensated-for. (Consider Figure 1(a) for example). Such pairs of transactions are identified as follows:

We say that T_i is *active with respect to* T_j if, and only if, there exist an SG_a where both transactions appear and $T_j \rightarrow T_i$ is not in SG_a, but there is a path (in either direction) in SG_a between CT_i and T_j.

Next, we introduce two properties of global SGs that are used to 'stratify' the global SG, thereby preventing regular cycles. Each property is presented as a formal assertion. We first introduce four predicates that depend on the transaction identifiers i and j:

A1. At any SG_a where T_j appears, $T_i \rightarrow CT_i \rightarrow T_j$.

A2. At any SG_a where T_j appears, $T_j \rightarrow CT_i$ without having T_i on that path.

A3. At any SG_a where both T_j and T_i appear, if there is a path between T_j and either T_i or CT_i, then the path $T_i \rightarrow CT_i \rightarrow T_j$ is in SG_a.

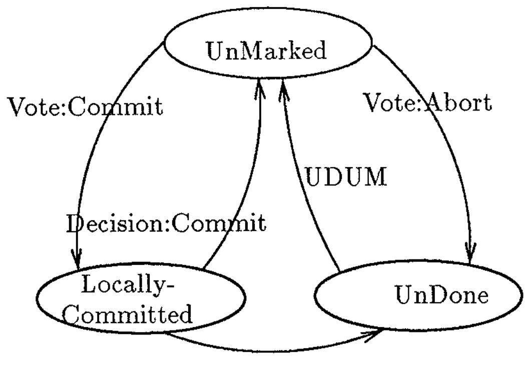

Figure 2: Transitions in the marking of a site with respect to a transaction

A4. At any SG_a where both T_j and T_i appear, if there is a path between T_j and CT_i in SG_a, it must be the path $T_j \longrightarrow CT_i$ without having T_i on that path.

Using these predicates we introduce two *stratification properties*:

S1. $(\forall T_i, T_j : T_i$ is active wrt $T_j : A1 \vee A4)$

S2. $(\forall T_i, T_j : T_i$ is active wrt $T_j : A2 \vee A3)$

Lemma 3. *$C1 \Rightarrow \neg S1$, and $C2 \Rightarrow \neg S2$.* □

Theorem 1. *If either one of the stratification properties S1 or S2 hold then there are no regular cycles in the global SG.* □

Since we assume local serializability, Theorem 1 gives a sufficient condition for ensuring correctness.

Theorem 2. *If a history H is correct, and if CT_i writes at least all data items written by T_i, then there is no case where a transaction T_j reads from both T_i and CT_i in H (i.e., atomicity of compensation is preserved).* □

In [Lev90], we elaborate on other variants of atomicity of compensation and ways to ensure them.

6 Ensuring Correctness

In this section, we present two protocols that ensure our correctness criterion when the O2PC protocol is employed. As such, the protocols actually complement the O2PC protocol. The protocols prevent regular cycles in the global SG by implementing the stratification properties. We strive for protocols whose execution requires no messages other than the standard 2PC messages.

6.1 Marking Sites

The basic building block for implementing protocols that are based on the stratification properties is a simple marking of sites. With respect to a specific global transaction T_i, a site is either *unmarked*, or *marked*. Then, a site is marked *locally-committed* with respect to T_i, or marked *undone* with respect to T_i. Initially, a site is unmarked with respect to a transaction T_i. A site is made locally-committed with respect to T_i once it votes to commit T_i in response to a VOTE-REQ message. On the other hand, if the site votes to abort T_i, the site is made undone with respect to T_i. A site ceases to be locally-committed with respect to T_i and becomes unmarked with respect to that transaction whenever the site receives the decision message from the 2PC coordinator to commit T_i. If the decision is to abort T_i, then the site becomes undone with respect to T_i. At some point, a site ceases being undone with respect to an aborted transaction and becomes unmarked with respect to that transaction. We postpone the discussion concerning this transition to Section 6.2. It is important to note that all these transitions in the marking are triggered either by local events, or by messages that are already part of the 2PC protocol. Figure 2 summarizes the transitions in the markings.

Using this marking scheme, we devise protocols that ensure that the stratification properties are satisfied. Intuitively, the protocol should prevent situations where a global transaction accesses a site that is locally-committed with respect to another transaction, as well as a site that is undone with respect to that other transaction, since such a situation can result in a regular cycle. Protocols P1 and P2 correspond to the stratification properties S1 and S2, respectively. Each of the two protocols can be summarized by a rule that restricts the sites a global transaction T_j may access:

P1. Let T_j execute at a site that is marked with respect to a T_i. Then for each such T_i, either one of the following conditions hold:

- all sites in which T_j executes are undone with respect to T_i.
- all sites in which T_j executes are either locally-committed or unmarked with respect to T_i.

P2. Let T_j execute at a site that is marked with respect to a T_i. Then for each such T_i, either one of the following conditions hold:

- all sites in which T_j executes are locally-committed with respect to T_i.
- all sites in which T_j executes are either undone or unmarked with respect to T_i.

In the context of a multidatabase environment, it is very important to notice that P1 and P2 do not impose any restrictions on local transactions. Only global transactions are subject to the restrictions posed in the protocols. Therefore, the autonomy of local database systems is not affected by these protocols.

6.2 Protocol P1

In this section, we outline a possible implementation of protocol P1, and argue about the correctness of the protocol. Since P2 is in some sense dual to P1 we do not discuss P2 here.

The main challenge in devising an implementation for P1 is the timing of the transition from undone to unmarked with respect to T_i. Making this transition too early can cause the formation of regular cycles. Recall that P1 allows a transaction T_j to access data at sites that are locally-committed with respect to T_i as well as access data at sites that are unmarked with respect to T_i. Therefore, T_j may access a site that is locally-committed with respect to T_i and a site that was undone with respect to T_i and was prematurely unmarked. As far as correctness goes, the precondition for this problematic transition is formulated as follows. A site S_k that is undone with respect to T_i can be unmarked with respect to T_i, if:

> *UDUM0 (undone to unmarked).* All T_j that have accessed sites that are locally-committed with respect to T_i cannot possibly access S_k.

It seems that extra messages are needed for the detection of this condition. However, the fact that global transactions obey the 2PL rule can be used to implicitly deduce UDUM0. Namely, we observe that the condition in UDUM0 is implied by the following:

> *UDUM1.* For each site in which T_i executes, there is a transaction that has also executed at that site, while that site was undone with respect to T_i.

Once a site S_k makes a transition in its markings as specified by UDUM1, there can be no T_j that accesses a site that was locally-committed with respect to T_i and is about to access S_k. This argument is formalized in the following lemma.

Lemma 4. *UDUM1 implies UDUM0.* □

For the implementation of P1, the marking of sites locally-committed with respect to transactions is actually redundant, since the protocol allows transactions to access both sites that are locally-committed and unmarked with respect to another transaction. Hence, we can simplify matters and avoid the locally-committed marking altogether. We introduce data structures for maintaining the markings.

For each site, S_k, the protocol maintains a set, called *sitemarks.k*, of transaction IDs:

$$T_i \in \textit{sitemarks.k} \equiv S_k \text{ is undone wrt } T_i$$

These *marking sets* are updated to reflect the transitions described above, and are read by global transactions in order to ascertain whether execution at a particular site complies with the relevant protocol. The fact that a site is unmarked with respect to a transaction is deduced implicitly from the lack of any marking in the corresponding marking set. In order to preserve the semantics of the sets as defined above, concurrent accesses to the sets must be controlled. One option is to designate special entities for storing these sets in the underlying local databases. As part of the database, the sets are accessed by transactions subject to the 2PL rule. Other alternatives are explored in [KLS90b].

Each time a subtransaction is invoked at a new site a check is performed to determine whether the markings of the new site comply with the markings of the sites the global transaction has already had subtransactions in. For each global transaction, T_j, the protocol maintains a set, called *transmarks.j*, of transaction IDs:

$$T_i \in \textit{transmarks.j} \equiv$$
$$T_i \in \textit{sitemarks.k} \text{ and } T_{jk} \text{ was already invoked}$$

The set *transmarks.j* accumulates the markings of sites where T_j already has subtransactions. This set is used for the check which is performed by the function *compatible*(*transmarks*, *sitemarks*). This function returns *true* if the two sets are compatible with each other according to the protocol rules and false otherwise. Since only one type of marks is used, the compatibility check is simply:

```
compatible(transmarks, sitemarks)
    return ($\forall x : x \in$ transmarks : $x \in$ sitemarks)
```

The pseudo-code segment R1 below models the compatibility check and the corresponding actions. R1 should be executed as the first action of T_{jk} at S_k:

```
R1. if compatible(transmarks.j, sitesitemarks.k) then
        {transmarks.j ← transmarks.k ∪ sitemarks.k
         start the actions of $T_{jk}$ }
    else reject $T_{jk}$
```

In case the request to spawn the subtransactions is rejected it can be retried later, unless the incompatibility is such that only aborting the corresponding global transaction can resolve the situation (e.g., T_j is executed at a site that is unmarked with respect to T_i, and attempts to spawn a subtransaction at a site that is undone with respect to T_i).

Next we describe how the transitions in the markings are implemented. Implementing UDUM1 may be cheaper in terms of messages. However, it requires augmenting the data structures. Keeping track of the set of execution sites for each transaction is necessary. Also, it must be possible to determine at what site a marking $T_i \in$ *transmarks.j* was added to the *transmarks.j* set. For brevity, we do not present here the necessary augmented data structures. We note, however, that managing these structures does not incur any extra messages. The following pseudo-code segments summarize the implementation of P1:

```
R2. The last operation of $CT_{ik}$:
    $sitemarks.k \leftarrow sitemarks.k \cup \{T_i\}$

R3. Whenever UDUM1 is detected:
    $sitemarks.k \leftarrow sitemarks.k - \{T_i\}$
```

R3 is executed as part of the transaction that enabled the transition; that is, the transaction whose access to S_k made UDUM1 detectable at that site.

Lemma 5. *If T_j accesses (reads or writes) a data item at site S_k while the site is undone with respect T_i, then $CT_i \rightarrow T_j$ in SG_k.* □

Lemma 6. *Let T_i be an aborted transaction that has executed at site S_k. Suppose that $T_j \rightarrow T_i$ is not in SG_k. If T_j accesses a data item at S_k while the site is unmarked with respect T_i, then either $T_j \rightarrow CT_i$ at all sites where both T_j and T_i appear, or $CT_i \rightarrow T_j$ at all sites where both T_j and T_i appear.* □

Given the above two lemmata, we can establish that protocol P1 ensures that the stratification property S1 is indeed met.

Several comments concerning the protocols and their implementation are in order.

- Considering the proposed implementation for P1, we note that the marking sets induce extra conflicts among otherwise non-conflicting pairs of transactions only if one of the transactions aborts. Thus, again, under the optimistic assumption, performance is not offset by this overhead .

- Deadlocks may arise due to contention to the local marking sets. For example, a transactions that read-locks *sitemarks.k* in order to perform the compatibility check, may be blocked while attempting to access a regular data item x that is locked by CT_{ik}. The compensating transaction, on the other hand, may be blocked too, holding a lock on x and attempting to access *sitemarks.k*. One simple way to avoid this deadlock problem is to perform all the accesses to the marking sets as the last access of subtransactions. The only problem with this simple remedy is the compatibility check (R1). Checking for compatibility late results in wasted efforts in case the check fails. An acceptable compromise would be to perform the check first and then unlock *sitemarks.k*. In case the check succeeds and the subtransaction is completed, the check is validated again as the last action of the subtransaction.

- In addition to protocols P1 and P2 there are a variety of other protocols resulting from other stratification properties. For instance, a very simple protocol is one that requires that for each transaction T_j, all sites in which T_j executes are undone with respect to the same transactions, and are locally-committed with respect to no transaction. There is a trade-off between the protocol's simplicity and the degree of concurrency it allows. Further details on the other protocols can be found in [KLS90b].

7 Conclusion

The use of the 2PC protocol to ensure the atomicity of transactions in a distributed environment creates severe, yet inevitable difficulties. The O2PC protocol presented in this paper avoids these difficulties by trading standard atomicity for semantic atomicity. As a result of the relaxed atomicity notion, serializability may be lost. We propose a correctness criterion that reduces to serializability if no global transactions are aborted, and deviates from serializability only to the extent dictated and allowed by the special characteristics of compensating transactions.

The O2PC was augmented by P1 to preserve this criterion. A distinctive feature of the O2PC/P1 combination is that it makes no changes to the message transfer pattern or the structure of the standard 2PC protocol. Thus, the O2PC scheme is compatible with the standardization efforts of the 2PC protocol, currently underway.

References

[AGMS87] R. Alonso, H. Garcia-Molina, and K. Salem. Concurrency control and recovery for global procedures in federated database systems. *Data Engineering*, 10(3):5–11, September 1987.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, MA, 1987.

[BS88] Y. Breitbart and A. Silberschatz. Multidatabase update issues. In *Proceed-*

ings of ACM-SIGMOD 1988 International Conference on Management of Data, Chicago, pages 135–141, 1988.

[BST90] Y. Breitbart, A. Silberschatz, and G. R. Thompson. Reliable transaction management in a multidatabase system. In *Proceedings of ACM-SIGMOD 1990 International Conference on Management of Data, Atlantic City, New Jersey*, pages 215–224, 1990.

[CP87] S. Ceri and G. Pelagatti. *Distributed Database Systems, Principles and Systems.* McGraw-Hill, New York, 1987.

[DE89] W. Du and A. K. Elmagarmid. Quasi serializability: a correctness criterion for global concurrency control in InterBase. In *Proceedings of the Fifteenth International Conference on Very Large Databases, Amsterdam*, pages 347–355, 1989.

[fdb87] Special issue on federated databases systems. *Data Engineering*, 10(3), September 1987.

[GM83] H. Garcia-Molina. Using semantic knowledge for transaction processing in a distributed database. *ACM Transactions on Database Systems*, 8(2):186–213, June 1983.

[GMGK+90] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem. Coordinating multi-transaction activities. Technical Report UMIACS-TR-90-24, University of Maryland Institute for Advanced Computer Studies, February 1990.

[GMS87] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of ACM-SIGMOD 1987 International Conference on Management of Data, San Francisco*, pages 249–259, 1987.

[Gra78] J. N. Gray. Notes on database operating systems. In *Lecture Notes in Computer Science, Operating Systems: An Advanced Course*, volume 60, pages 393–481. Springer-Verlag, Berlin, 1978.

[Gra81] J. N. Gray. The transaction concept: Virtues and limitations. In *Proceedings of the Seventh International Conference on Very Large Databases, Cannes*, pages 144–154, 1981.

[hdb90] Special issue on heterogeneous databases. *ACM Computing Surveys*, 22(3), September 1990.

[KLS90a] H. F. Korth, E. Levy, and A. Silberschatz. A formal approach to recovery by compensating transactions. In *Proceedings of the Sixteenth International Conference on Very Large Databases, Brisbane*, pages 95–106, August 1990.

[KLS90b] H. F. Korth, E. Levy, and A. Silberschatz. An optimistic two-phase commit protocol. Technical Report TR-90-31, The University of Texas at Austin, Computer Sciences Department, 1990.

[KR88] J. Klein and A. Reuter. Migrating transactions. In *Future Trends in Distributed Computer Systems in the '90s, Hong Kong*, 1988.

[Lev90] E. Levy. A theory of relaxed atomicity. Submitted for publication, November 1990.

[map89] Multidatabase services on ISO/OSI networks for transactional accounting. Technical Report MAP761B, SWIFT, INRIA, GMD/FOKUS, University of Dortmund, 1989. Final Report, Edited by S.W.I.F.T. Society for Worldwide Interbank Financial Telecommunications s.c. 81 avenue Ernest Solvay, B-1310 La Hulpe, Belgium.

[MLO86] C. Mohan, B. Lindsay, and R. Obermarck. Transaction management in the R* distributed database management system. *ACM Transactions on Database Systems*, 11(4):378–396, December 1986.

[Reu89] A. Reuter. ConTracts: A means for extending control beyond transcation boundaries. Presentation at 3rd Workshop on High Performance Transaction Systems, Pacific Grove, CA, September 1989.

[Ske82] D. Skeen. Non-blocking commit protocols. In *Proceedings of ACM-SIGMOD 1982 International Conference on Management of Data, Orlando*, pages 133–147, 1982.

[Vei89] J. Veijalainen. *Transaction Concepts in Autonomous Database Environments.* R. Oldenbourg Verlag, Munich, 1989.