AN OPTIMISTIC COMMIT PROTOCOL FOR DISTRIBUTED TRANSACTION MANAGEMENT

Henry F. Korth, Eliezer Levy, and Abraham Silberschatz

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-90-31 September 1990

An Optimistic Commit Protocol for Distributed Transaction Management*

Henry F. Korth Eliezer Levy 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 combining a novel use of compensating transactions along with an optimistic assumption, we propose a revised 2PC protocol that overcomes these difficulties. In the revised protocol, locks are released when a site votes to commit a transaction, thereby solving the indefinite blocking problem of 2PC. Semantic, rather than standard, atomicity is guaranteed as the effects of a transaction that is finally aborted are semantically undone by a compensating transaction. 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. The results are of particular importance for multidatabase systems.

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]. It guarantees that either all or none of the effects of a multi-site transaction are committed despite site and communication failures. Typically, the 2PC protocol is combined with the strict two-phase locking protocol [BHG87], where locks are held until the end of transactions, as the means for ensuring atomicity and serializability in a distributed database. 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, locks must be held until either a commit or an abort message is received from the coordinator in the second phase of the 2PC protocol. Since the 2PC protocol is a *blocking* protocol [Ske82] the period of time locks are held can be unbounded. Specifically, if the coordinator or a communication link to the coordinator fail in a certain critical time, locks are retained until the failure is repaired. 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 for both the committing transaction, as well as transactions that are waiting for access to the locked data items.

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

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 [Gup89]. One of the distinctive features of a multidatabase is the requirement that the integration does not impair the *autonomy* of the local DBMSs. In particular, it is required that a local DBMS has full control over the execution of local transactions executing on behalf of global activities. However, once 2PC is imposed in a multidatabase environment, a local DBMS becomes a subordinate of an external coordinator, and therefore autonomy is certainly compromised if not sacrificed. A local DBMS can no longer unilaterally determine the fate of those local transactions that are executing as part of global transactions. Moreover, by holding onto locks and other resources for very long periods, global transactions delay local transaction processing, thereby restricting autonomy. Since some of the local DBMSs in the multidatabase may actually prohibit holding locks on data items for such long durations; this renders the execution of multi-site transactions impossible.

It is not possible to have a non-blocking commit protocol that is immune to both site and link failures [BHG87]. Therefore, the 2PC protocol is a compromise, and the problems we have outlined seem to be an inevitable penalty. In this paper, we introduce an alternative to the standard 2PC protocol that alleviates the virtually inevitable blocking and lengthy delays problems. The new protocol is applicable whenever the atomicity problem of a multi-site transaction surfaces. It can benefit distributed database systems in general and multidatabase systems in particular.

The protocol does not guarantee transaction atomicity in the standard sense, but rather a weaker notion referred to as *semantic atomicity* [GM83]. A salient contribution of this paper is the examination of the consequences of the protocol and this weaker atomicity notion in terms of serializability, and correctness in general. We extend the standard serializability theory to account for transaction failures and their recovery by compensating transactions, and specify an appropriate correctness criterion. We also outline an implementation that preserves this criterion. 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 are used in the construction of the protocol and its correctness criterion 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. Some implementation remarks are also presented in this section. We sum up with conclusions in Section 7.

2 The Basic O2PC Protocol

The 2PC protocol is pessimistic in nature. Holding the locks until the end of the second phase, which is the cause of blocking, is necessary only if the transaction at hand is to abort. If such aborts occur rarely, then most of the potentially long-duration waits induced by 2PC are unnecessary. Basing a revised commit protocol on an *optimistic assumption* that in most cases the protocol terminates successfully (i.e., the transaction commits) can dramatically reduce waiting due to data contention, thereby improving the performance of the system. Such an assumption is valid for most practical settings. 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.

We describe such an *optimistic 2PC (O2PC)* protocol that is a slightly modified version of the standard combination of the 2PC protocol and strict 2PL protocol. As in the standard protocol, a multi-site transaction T is associated with a coordinator which initiates the protocol by sending a VOTE-REQ message (also referred to as PREPARE message) to all participating sites. If a site votes to abort T, then as in the standard protocol, an abort message 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, *all locks held by T are released at once, without waiting for the coordinator's final commit or abort message.* In this case, we say that T is *locally-committed* at that particular site.

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 voted to abort it. A situation may arise where, at some sites T is locally committed, whereas at some other sites T is aborted. In this case, the effects of T must be undone at sites where it is locally-committed. Undoing the effects of a locally-committed T is problematic, if not impossible, using standard recovery techniques. If a transaction which read T's updates has already committed, this execution is deemed non-recoverable [BHG87]. Even if the transactions that read from T are still active, the only way to undo T's effects is via *cascading aborts*; all transactions that have read from T must be also aborted. Cascading aborts is an undesirable phenomenon since it can result in uncontrollably many transactions being forced to abort because some other transaction happened to abort.

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 [KLS90] 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 to ensure transaction atomicity despite of the uncoordinated commitment of updates at different sites. After voting to commit T, a site still carries on with the second phase of the regular 2PC protocol (despite having released locks held by T). If the site receives a decision message from the coordinator to abort T, then it invokes the corresponding compensating transaction. Since it is more likely that the decision would be to commit T, the gain by the early release of locks should outweigh the overhead associated with those cases requiring compensation for T. The message transfer in the O2PC protocol between the coordinator and a participating site is depicted pictorially in Figure 1.

Instead of the familiar all-or-nothing semantics, the protocol ensures a similar, though weaker, atomicity guarantee referred to as *semantic atomicity*. Decomposing a multi-site transaction into a set of single-site transactions, semantic atomicity states that either all subtransactions are locally-committed (and then the entire transaction is committed), or all locally-committed subtransactions are compensated-for. We acknowledge 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 standard 2PC) 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.

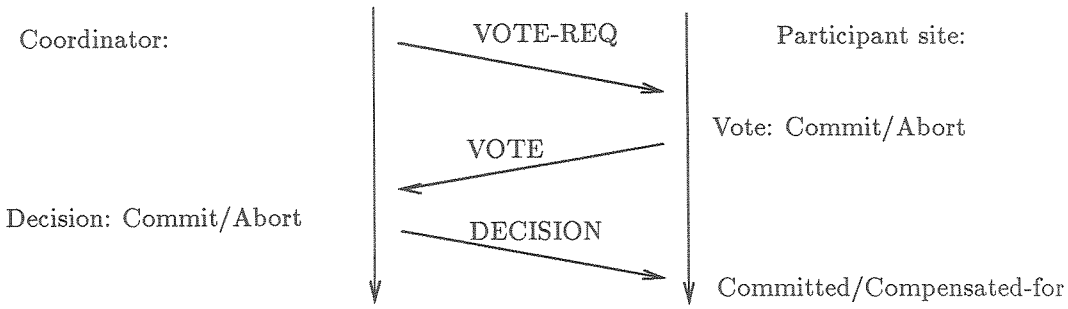

Figure 1: A schematic view of the O2PC protocol

3 Preliminaries

In order to discuss the correctness of our protocol we must first introduce some assumptions and terminology that are used in our exposition. These issues concern the underlying distributed transaction management architecture, compensating transactions, and the exact locking discipline of the O2PC protocol, and are presented in this order.

3.1 Transaction Management Architecture

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 *sub*transaction that is executed on behalf of a global transaction, and an independent local transaction.

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.

Each global transaction is associated with a global transaction manager whose functions are to spawn local subtransactions, monitor their execution and terminate them by playing the role of the 2PC coordinator. It is irrelevant to our discussion whether a single transaction manager in the system supervises all global transactions, or whether there are several such managers, each responsible for some global transactions.

The decomposition of a global transaction into local subtransactions conforms to one of two models [Joh90]. 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

We denote the *compensating transaction* that is specific to the *forward* transaction T_i by CT_i. CT_i undoes T_i's effects semantically without causing the cascading aborts of transactions that have read from 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 [KLS90] 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).

Compensating transactions cannot voluntarily abort, nor are they subject to a system-initiated abort. Also, their completion is guaranteed despite system crashes by either resuming them from a save-point, or retrying them. Finally, a compensating transaction must be designed to avoid a logical error leading to abort. Consequently, 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, KLS90]. The rationale behind persistence of compensation is preserving semantic atomicity. Initiating a compensating transaction parallels a decision to abort the transaction in the traditional setting — definitely a non-reversable decision. The persistence of compensation requirement implies that design of a compensating transaction is a complex and application-dependent task. The fact that CT_i always executes *after* T_i must be used to alleviate this difficulty. Essentially, the log records of T_i should contain enough information for CT_i to execute properly. This information should include input parameters for T_i's operations for example. 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. 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 [KLS90]. 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.

The execution of a compensating transaction requires access to the log and other information stored on stable storage, thus further increasing the cost associated with this type of transaction. For these reasons, we limit the usage of compensating transactions in our context, for relatively rare pessimistic cases of failures of global transactions.

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).

3.3 Locking Discipline

The locking policy described in this section is implicitly assumed throughout the paper. The O2PC protocol is based on a combination of a strict 2PL and 2PC protocols that is referred to in [BHG87] as *distributed 2PL*. This combination (as it is, without our proposed changes) guarantees serializability globally.

The locking discipline of global transactions in O2PC is based on the following:

- No additional locks can be requested by a subtransaction once a VOTE-REQ message for the global transaction has been sent by the transaction's 2PC coordinator.
- A lock may be released only after the VOTE-REQ message has been accepted.

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. Hence, as locks are released locally only after the VOTE-REQ message has been received, it is guaranteed that a transaction releases a lock at any site only after it has finished acquiring locks at all sites. Observe that this 2PL property is characteristic of the original distributed 2PL, as well as the O2PC protocols, even under the early lock release provision of the latter.

At each site locks for data items residing at that site are granted by the local lock manager to the transactions and subtransactions executing locally. As was mentioned earlier, locks held by subtransactions are released as soon as the site votes to commit the corresponding global transaction, or after the local roll-back is completed if the site votes to abort. Local transactions (i.e., non-subtransactions) follow the strict 2PL protocol. Compensating subtransactions are treated as local transactions rather than as subtransactions of global transactions with respect to locking; that is, they also follow strict 2PL locally. The reasons for this important decision are elaborated in the next section. The bottom line is that at each site, the local execution over local transactions, subtransactions, and compensating subtransactions is serializable.

4 Correctness, Transaction Failures and Compensations

The local uncoordinated commitment and the use of compensating transactions in the O2PC protocol interact in a subtle manner with concurrency control and correctness issues. A prerequisite for studying this interaction is the acknowledgement that the theoretic and modeling tools (e.g., serialization graphs) must account for failed transactions and their corresponding compensating transactions. By contrast, the traditional serializability theory deals only with committed projections of histories [BHG87]. We adopt this extended approach in our

exposition. Having made this adjustment, one might be tempted to impose serializability over all transactions, including compensated-for and compensating transactions, as the correctness criterion. There are, however, several compelling reasons why this is not a good choice. Essentially, compensating transactions possess several special properties that render such adjusted 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. Each local compensating subtransaction can terminate independently. 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. Hence, coordinating the scheduling of compensating subtransactions can be done only by extra messages, thereby defeating the purpose of the O2PC in the first place.

- 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, i.e., 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 for multidatabases, where autonomy prevents constraining the automatic local roll-back. Therefore, serializability of a compensating transaction may be again jeopardized since the scheduling of the compensating transaction as a whole 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, violate 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. It is envisioned that in practice the forward subtransactions would leave traces (in the form of informative log records, for example) for their corresponding compensations. Then, compensation would be driven by these independent traces, rather then by a coordinated effort (even in the generic model). 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.

For these reasons it is impossible, and sometimes unnecessary, to impose a global locking discipline on a global compensating transaction. Compensating subtransactions are going to release their locks once they complete their local processing, regardless of the execution of their sibling compensating subtransactions. As a result, serializability of global compensating transactions may be lost. That is, cycles with compensating transactions may occur in the global serialization graphs.

The of loss of serializability would not be relevant if serializability is abandoned in the first place as the correctness criterion for global transactions. That is, 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. Also, another simple way of taking advantage of the O2PC idea without tackling correctness issues is to allow *only local transactions* to benefit from the fact that global transactions release their locks early. That is, a global transaction releases its locks and becomes locally-committed only for the purposes of letting local transactions proceed; other global transactions are still delayed. This simple version of the O2PC protocol reduces the length of time local transactions are delayed due to global transactions. However, if the O2PC is to be used in the most general manner, to the advantage of both global and local transactions, and if serializability is of concern, then the problem of loss of serializability that was raised above has to be addressed. The rest of the paper is devoted to examining this issue.

We illustrate the problem of loss of serializability in the following example. Consider a global transaction T_1 composed of two subtransactions T_{11} and T_{12} executing at sites S_1 and S_2, respectively. Suppose that the O2PC is employed and that S_1 voted to commit T_1, whereas S_2 voted to abort T_1. In site S_1, updates of T_1 are exposed since locks have been released, whereas in site S_2, the updates of T_1 were automatically undone. A transaction T_2 (that is serialized before CT_{11}) sees an inconsistent state by accessing data items in both sites. A cycle $CT_1 \rightarrow T_2 \rightarrow CT_1$ is formed in the global serialization graph, and hence global serializability is not preserved. This scenario is depicted in Figure 2(a), where SG1 and SG2 denote the serialization graphs of sites S_1 and S_2, respectively.

Preventing scenarios like the one shown above by globally coordinating and synchronizing the execution of global compensating transactions contradicts their special properties, and is bound to introduce the very same problems the O2PC was originally set to solve. Therefore, a revised correctness criterion (rather than serializability) is called for. In the quest for an appropriate alternative let us consider the above example again. Recalling that a compensating transaction is actually a set of semi-independent subtransactions, we note that a potential for inconsistency arises due to T_2's participation in the cycle, rather then the fact CT_1 is part of this cycle. The independence of the subtransactions of CT_1 implies that CT_1 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.

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 [KLS90] and is elaborated in [Lev90]. Referring to the execution depicted in Figure 2(a), if T_2 reads data items written by T_1 in S_1, and also reads data items written by CT_1 in S_2, atomicity of compensation is violated.

Our intention is to propose a revised correctness criterion that takes into account the special properties of compensating transactions and guarantees atomicity of compensation. It is natural to base the revised criterion on the requirement that in the absence of transactions failures, serializability is preserved. In the next section, we formally present our correctness criterion.

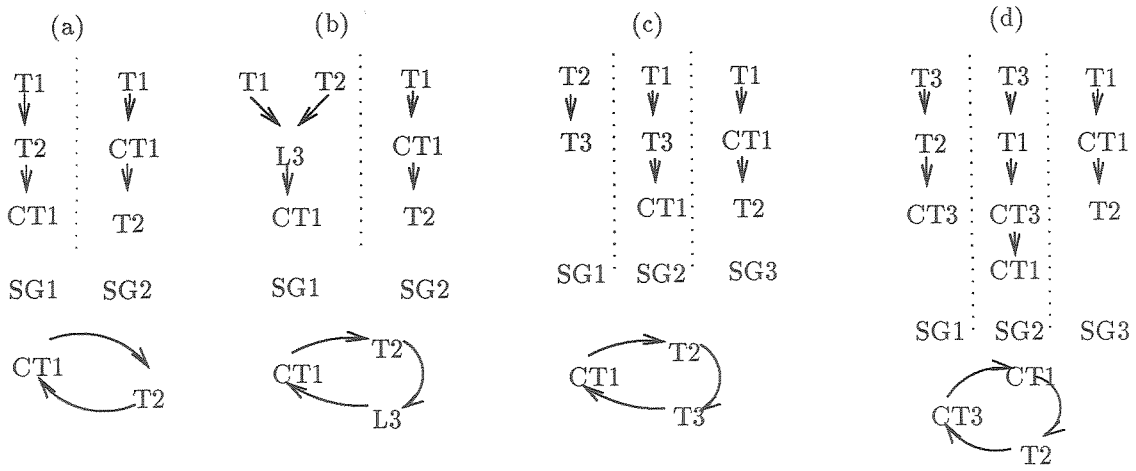

Figure 2: Regular Cycles

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].

The *local serialization graph* for a complete local history H over global transactions $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$, the corresponding compensating transactions $\mathcal{CT} = \{CT_1, CT_1, \ldots, CT_n\}$, and local transactions $\mathcal{L} = \{L_1, L_2, \ldots, L_m\}$ 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.

Observe that only aborted local transactions (which are irrelevant for our purposes) are not represented in the local SG.

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 local transactions.

The main result of this section is a theorem that prescribes a sufficient condition for obtaining the correctness criterion and atomicity of compensation. Proofs are given in the Appendix. 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.

5.1 Paths and Cycles

Capital letters at the beginning of the alphabet, e.g., A, B, C denote either compensating or regular global transactions. Given a particular history H, the notation $A \rightarrow B$ is used to denote that there is a directed path (of arbitrary length) between the two transaction nodes in $SG(H)$.

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

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. Observe that it is irrelevant in this case whether C is a local transaction or a local subtransaction, since it is omitted in the representation.

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). This representation is also 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. Consequently, the global path $CT_1 \rightarrow CT_3$ does not include T_2. $\diamond$

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 2 demonstrates several regular cycles, by presenting the corresponding segments of the local SGs.

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

The proof of Lemma 1 is based on the fact that regular transactions follow the 2PL rule.

Our correctness criterion states that a *history H is correct if, and only if, $SG(H)$ contains no regular cycles.*

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.*

$C1 \equiv (\exists T_i, T_j : i \neq j : (\exists a : CT_i \rightarrow T_j \text{ in } SG_a) \wedge (\exists b : T_j \text{ in } SG_b : T_j \rightarrow CT_i \vee nopath(b, T_j, CT_i)))$

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.*

$C2 \equiv (\exists T_i, T_j : i \neq j : (\exists a : T_j \rightarrow CT_i \text{ in } SG_a) \wedge (\exists b : T_j \text{ in } SG_b : CT_i \rightarrow T_j \vee nopath(b, T_j, CT_i)))$

Figure 3, schematically depicts the minimal representation of a regular cycle $A_0 \rightarrow A_1 \rightarrow \ldots \rightarrow A_n \rightarrow A_0$. Observe that each A_i is not related to A_{i+1} in SG_{i+1} according to the lemma.

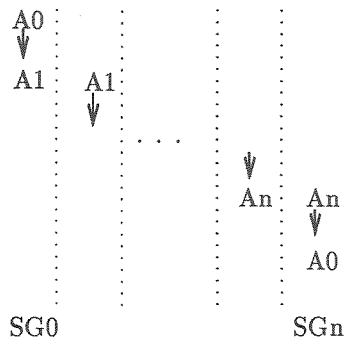

Figure 3: A generic regular cycle

5.2 Preventing Regular Cycles

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_j follows another T_i in the SG before the latter transaction is globally committed or fully compensated-for (the meaning of this vague statement is to be made precise soon). See Figure 2(a), for instance, with T_1 and T_2 as T_i and T_j, respectively. Such pairs of transactions are identified in the following definition:

T_i is *active with respect to* T_j if, and only if, there exist an SG_a where the following conditions are met:

- Both transactions appear and it is not the case that $T_j \rightarrow T_i$ in SG_a.
- There is a path (in either direction) in SG_a between CT_i and T_j.

Next, we introduce a family of characterizations of global SGs, called *stratification properties*. These properties 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.

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 $T_j \rightarrow CT_i$ without having T_i on that path.

Using these predicates we introduce seven stratification properties:

S6. $(\forall T_i,\ T_j : T_i$ is active wrt $T_j : A1)$

S7. ($\forall T_i$, $T_j : T_i$ is active wrt $T_j : A2$)

S3. ($\forall T_i$, $T_j : T_i$ is active wrt $T_j : A3$)

S5. ($\forall T_i$, $T_j : T_i$ is active wrt $T_j : A4$)

S4. ($\forall T_i$, $T_j : T_i$ is active wrt $T_j : A1 \vee A2$)

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

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

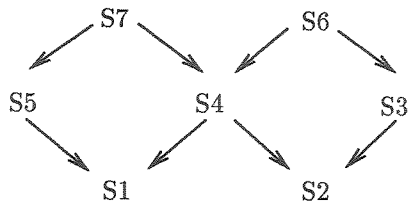

Figure 4: The lattice of $S_1 \ldots S_7$

Lemma 3. *The relationships among $S1 \ldots S7$ are given by the lattice of Figure 4, where each arrow represents a logical implication in the same direction.*

The proof of Lemma 3 is a straight forward exercise in predicate logic.

Lemma 4. *The following assertions hold:*

- $C1 \Rightarrow \neg S1$
- $C2 \Rightarrow \neg S2$

Proof.

- Consider the path $CT_i \rightarrow T_j$ in SG_a whose existence is guaranteed by the first conjunct of C1. Because of this path, T_i is active with respect to T_j. By the second conjunct of C1, there exist an SG_b where either $T_j \rightarrow CT_i$, or there is no path between the two. In both cases, the negation of $A1(i, j)$ is implied. Considering the path $CT_i \rightarrow T_j$ in SG_a again, we observe that the negation of $A4(i, j)$ holds. Therefore, we have demonstrated that for T_i and T_j, where the former is active with respect to the latter, both $\neg A1$ and $\neg A4$ hold.
- By a symmetric argument the second part of the lemma follows.

□

Theorem 1. *If any of the stratification properties Si ($1 \leq i \leq 7$) hold then there are no regular cycles in the global SG.*

Proof. From Lemma 4 we have that $(C1 \wedge C2) \Rightarrow \neg Si \ \ (1 \leq i \leq 7)$. By transitivity of implication applied on Lemma 2 and the last implication, we have the counter-positive form of the theorem. Namely, if there is a regular cycle in the global SG, then $\neg Si$. □

As a consequence of Theorem 1, we can develop a family protocols ensuring that the global SG has no regular cycles in Section 6.

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).*

Proof. The definition of 'reads-from' is from [BHG87]. We prove the counter-positive form of the theorem. Assuming that there is T_j that reads from both T_i and CT_i in H, we show that SG(H) has a regular cycle. Because of the given read-from relations the edges $T_i \rightarrow T_j$ and $CT_i \rightarrow T_j$ are in SG(H). Let T_j read a data item x from T_i. By assumption, CT_i writes x, too. Since T_j reads x from T_i, then the write of CT_i of x must follow the read of T_j of x. Therefore, there is an edge $T_j \rightarrow CT_i$ in SG(H) and the cycle $T_j \rightarrow CT_i \rightarrow T_j$ is formed. Since local histories are serializable, this must be a global, and hence a regular cycle. In [Lev90], we elaborate on other variants of atomicity of compensation and ways to ensure them. □

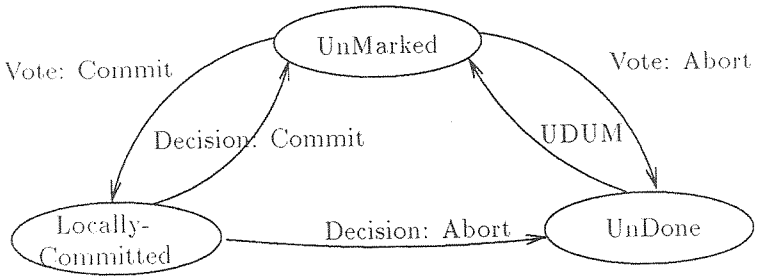

Figure 4: Transitions in the marking of a site with respect to a transaction

6 Protocols Satisfying The Correctness Criterion

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 4 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

- all sites in which T_j executes are undone with respect to T_i, *or*
- 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

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

There is a certain similarity between these protocols and the altruistic locking protocol [SGMA89]. In our case, however, an aborted global transaction creates two *wakes* (see [SGMA89]): an undone wake and a locally-committed wake. Similarly to the way altruistic locking restricts entering and leaving a wake, P1 and P2 restrict accessing both wakes.

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.

We introduce data structures for maintaining the markings. For each site, S_k, the protocol maintains the set *sitemarks.k* defined as follows:

$(T_i, LC) \in$ *sitemarks.k* *iff* S_k is locally committed with respect to T_i

$(T_i, UD) \in$ *sitemarks.k* *iff* S_k is undone with respect to 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. Some possible optimizations are discussed in Section 6.3.

In both protocols P1 and P2, each time a subtransaction is invoked at a new site by the global transaction manager a check whether the markings of the new site comply with the markings of the sites the global transaction has already had subtransactions in, is performed. For each global transaction, T_j, the protocol maintains a set of markings, *transmarks.j* defined as follows:

$(T_i, mark) \in$ *transmarks.j* *iff* $(T_i, mark) \in$ *sitemarks.k* and T_{jk} 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. The logic of this function is described shortly. The pseudo-code segment R4 models the compatibility check and the corresponding actions.

R4. The first action of T_{jk} at S_k:

if *compatible*(*transmarks.j*, *sitesitemarks.k*) **then**
{*transmarks.j* $\leftarrow$ *transmarks.k* $\cup$ *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 locally committed with respect to T_i, and attempts to spawn a subtransaction at a site that is undone with respect to T_i).

6.2 Implementing P2

For the implementation of P2 the marking of sites undone with respect to transactions is actually redundant, since the protocol allows transactions to access both sites that are undone and unmarked with respect to another transaction. Hence, we can simplify matters and avoid the undone marking altogether. The following pseudo-code segments summarize the implementation of P2:

R1. After S_k responds to the VOTE-REQ message sent for T_i:

if S_k votes to commit T_i **then** $sitemarks.k \leftarrow sitemarks.k \cup \{(T_i, LC)\}$

R2. The last operation of CT_{ik}:

$sitemarks.k \leftarrow sitemarks.k - \{(T_i, LC)\}$

R3. After receiving a DECISION message for T_i:

if DECISION is COMMIT **then** $sitemarks.k \leftarrow sitemarks.k - \{(T_i, LC)\}$

Observe that R3 is required only to discard the LC mark and reclaim its space. It has no consequence regarding regular cycles, since T_i commits.

Since P2 requires marks of only one type, the compatibility check is simply:

$compatible(transmarks, sitemarks)$

return $(\forall x : x \in transmarks : x \in sitemarks)$

Reasoning that P2 is a correct implementation of the stratification property S2 follows from the next two lemmata.

Lemma 5. *If T_j accesses a site S_k while it is locally committed with respect T_i, then if T_i finally aborts $T_j \rightarrow CT_i$ at SG_k.*

Proof. Since the accesse of T_{jk} and CT_{ik} to *sitemarks.k* conflict, they must be ordered. Since T_j accesses S_k while it is locally committed with respect to T_i, and since by R2, CT_{ik} removes the mark (T_i, LC), it must be that $T_j \rightarrow CT_i$ at SG_k. □

Lemma 6. *If T_j accesses a site S_k while it is unmarked with respect T_i, T_i has executed at S_k not preceded by T_j, and T_i finally aborts, then $CT_i \rightarrow T_j$ at SG_k.*

Proof. Similarly to the previous proof, since T_i, CT_i and T_j all conflict when accessing *sitemarks.k* and since $T_i \rightarrow CT_i$ by definition, there are two possible orders among the three transactions: $T_i \rightarrow T_j \rightarrow CT_i$ at SG_k or $T_i \rightarrow CT_i \rightarrow T_j$ at SG_k. Had the first path been a valid one, then by R1 S_k would have been marked locally committed with respect to T_i, and T_j could not have accessed S_k while it is unmarked with respect to T_i. □

To complete the proof of correctness of the implementation all we need to observe is that the compatibility check (R4) enforces the rule form of P2.

6.3 Implementing P1

Similarly to P2, P1 can be implemented using only one type of marks; undone marks in this case.

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 sites that are locally-committed with respect to T_i as well as 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.

Once UDUM0 holds, the undone to unmarked with respect to T_i transition can be safely made. Implementing UDUM0 directly incurs extra messages, however.

One way to alleviate this difficulty is to employ P1 under the provision that undone markings are never discarded. That is, once a site is marked undone with respect to T_i, it remains so. This alternative might not be acceptable as it implies ever growing marking sets for sites, incurring both storage overhead and extra time in performing the checks needed to enforce P1. Even in light of the optimistic assumption that aborts — and therefore undone marks — are rare, we must provide a rule for discarding undone marks for reasons of efficiency of the protocol. By novel use of the fact that global transactions obey the 2PL rule, the knowledge needed to detect UDUM0 can be implicitly deduced rather than explicitly disseminated and gathered by extra messages. 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 7. *UDUM1 implies UDUM0.*

Proof. Let T_j be a global transaction that has accessed a site that was locally-committed with respect to T_i. UDUM1 guarantees that at each site S_m, where T_i executed, there is at least one global transaction, T_{l_m}, that follows T_j at S_m. This order results from the observation that T_j accessed S_m while it was locally-committed with respect to T_i, whereas T_{l_m} accessed that site while it was undone with respect to T_i. Hence, had it been possible for T_j to access S_k, it would have been a violation of the 2PL rule (and Lemma 1), as T_j would have followed a T_{l_m} at S_k. □

Next we describe how the transitions in the markings are implemented for P1. 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 $(T_i, UD) \in transmarks.j$ 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:

R1. The last operation of CT_{ik}:

$$sitemarks.k \leftarrow sitemarks.k \cup \{(T_i, UD)\}$$

R3. Whenever UDUM1 is detected:

$$sitemarks.k \leftarrow sitemarks.k - \{(T_i, UD)\}$$

Regarding R1, recall that if a S_k votes to abort T_i, then the standard undo actions taken locally are considered to constitute CT_{ik}. UDUM1 is labeled R3 intentionally, since its purpose is similar to that of R3 of P2. Observe that correctness is not lost by the absence of R3, it is rather the efficiency of the protocol that is affected. 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 8. *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.*

Proof. Since the accesses of T_{jk} and CT_{ik} to *sitemarks.k* conflict, and since the history at S_k is serializable, CT_i and T_j must be ordered. Since T_j accesses S_k while the site is undone with respect to T_i, and by R2, CT_{ik} adds the mark (T_i, UD), it must be that $CT_i \rightarrow T_j$ in SG_k. □

Lemma 9. *If T_j accesses a data item at site S_k while the site is unmarked with respect T_i, T_i has executed at S_k not preceded by T_j, and T_i finally aborts, then either:*

- *$T_j \rightarrow CT_i$ at all sites where both T_j and T_i appear.*

- *$CT_i \rightarrow T_j$ at all sites where both T_j and T_i appear.*

Proof. Similarly to the previous proof, T_j and CT_i must be ordered. We show a contradiction in case that $T_j \rightarrow CT_i$ in SG_a and $CT_i \rightarrow T_j$ in SG_b. By R1, S_b was marked undone with respect to T_i before it became unmarked by R3. By UDUM1, the latter transition is enabled only if there exists a T_k that was executed at S_a while S_a was undone with respect to T_i. That is, $T_j \rightarrow CT_i \rightarrow T_k$ in SG_a. Considering the transaction that executed R3 as part of T_k, we have $CT_i \rightarrow T_k \rightarrow T_j$ in SG_b. We have a regular cycle that includes no compensating transaction — a contradiction to Lemma 1. □

Given the above two lemmata, we can now establish that protocol P1 ensures that the stratification property S6 is indeed met. To do so all we need to observe is that the compatibility check (R4) enforces the rule form of P1.

6.4 Discussion

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, performance is not offset by overhead under the optimistic assumption.

- Alternatively to storing the marking sets as data items in the database, they can be stored and managed externally. A special software module whose responsibility is the scheduling of global transactions would maintain the marking sets. This module should implement a concurrency control scheme for accessing the marking sets. The concurrency control scheme can be customized to take full advantage of the simple access pattern to the marking sets. Such an architecture might be preferable in the multidatabase context, since storing the marking sets in the local database might be cumbersome and even prohibited.

  Typically, in a multidatabase system, at each site, an *agent* of the global transaction manager is running as an application program, that is, above the local transaction manager. These agents spawn local subtransactions, submit requests originating at the global transaction manager for local execution through these subtransactions, and participate in the 2PC protocol as the representatives of their sites. The functions of managing the marking sets can be integrated into these agents. There are yet more problems with a MDB environment like global deadlocks and implementation of persistence of compensation.

- 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*. Consequently, if contention to the

markings sets is may introduce deadlocks, these sets had better be stored as data in the local database, so that the deadlocks are detected and resolved by the local deadlock handling scheme.

- 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 a multidatabase environment, this particular technique is better suited when the marking sets are managed by the agents, since only then explicit unlocking is possible. To simulate the effect of unlocking *sitemarks.k* after the compatibility check in case the marking sets are managed by the local DBMS, a special local transaction have to be invoked to perform the check and then terminate, thereby releasing the locks.

- Another way to reduce contention to the marking sets is to split them into individually lockable entities, one for each mark. Observe that R3 requires locking only of the deleted mark and not the entire set. Multigranularity locking [GAPT75] would be very beneficial in this case since R1 and R2 require locking of the entire set.

- Another argument in favor of a customized locking scheme managed by the agents pertains only to R3 of P2. Observe that the transition specified in R3 can be executed without locking at all. If T_i is globally committed, there is no risk of forming regular cycles. Therefore deleting the (T_i, LC) mark while the marking set is locked by another transaction can be of no consequence. If the marking sets are managed by the local DBMS, R3 would have been implemented by a special transaction.

- Each of the two protocols is composed out of a *permissive* clause and a *restrictive* clause. The permissive clause of P1, for example, allows transactions to access both sites that are marked *locally committed* and sites that are unmarked with respect to a particular transaction. The permissive clause of P2, on the other hand, allows transactions to access both sites that are marked *undone* and sites that are unmarked with respect to a particular transaction. Based on our optimistic assumptions that transaction aborts are the exception rather than the rule, it is more likely to have many locally committed markings and few undone markings. Therefore, it seems that having a permissive clause based on locally committed markings (as in P1) would result in a better protocol. A restrictive clause based on locally committed marks is more likely to cause failures in the compatibility checks and hence rejections of subtransactions. These qualitative assertions, however, must be supported by an experimental study.

- The optimistic assumption favors P1 over P2 in one more important aspect. In P2, the concurrency control on the marks sets induces a total order among subtransactions at each site, even if they do not conflict on regular data. This order is determined by the order in which subtransactions execute R1 and R2 of P2. In P1, on the other hand, R1 is executed only in the rare cases of a transaction abort, hence contention for the markings sets and the total order effect is diminished significantly. Under the optimistic assumption, most of the accesses to the marking sets in P1 would be read accesses due to R4! For the last two reasons, it is likely that P1 will out-perform P2 under such optimistic circumstances.

- In addition to protocols P1 and P2 there are a variety of other protocols resulting from the 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. This protocol corresponds to S6. There is a trade-off between the protocol's simplicity and the degree of concurrency it allows.

7 Conclusion

Using the 2PC protocol to ensure the atomicity of transaction in distributed environments creates severe, yet inevitable difficulties. Our O2PC protocol avoids these difficulties by trading standard atomicity for semantic atomicity. The basic protocol as presented in Section 2 is valuable in the context of global transactions that are executed as sagas or multi-transactions, and for the purposes of relieving local transactions from the maladies of using 2PC for global transactions. When standard global transactions are used, 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. The changes are in local reactions to the protocol's messages. Therefore, O2PC does not contradict standardization efforts of the 2PC protocol.

Regarding future research, we plan to investigate the duality in the approach to relaxed atomicity notions. Dually to the way semantic atomicity is obtained by backward recovery using compensating transactions, one can recover forward using retrial [RELL90]. The duality of retrial and compensation is rooted in the traditional redo/undo paradigms. Recently, the traditional redo/undo duality was examined in the context of obtaining standard atomicity in multidatabases [MR91].

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 *Proceedings 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.

[GAPT75] J. N. Gray, Lorie R. A., G. R. Putzolu, and I. L. Traiger. Granularity of locks and degrees of consistency in a shared data base. In *IFIP Working Conference on Modeling of Data Base Management Systems*, pages 1–29, 1975. Also available as Research Report RJ1654, IBM, September 1975.

[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.

[Gup89] A. Gupta. *Integration of Information Systems: Bridging Heterogeneous Databases.* IEEE Press, New York, 1989.

[Joh90] W. Johannsen. Transaction models for federative distributed database systems. In *Proceedings of the International Conference on Information Technology (InfoJapan 90)*, pages 285–292, 1990.

[KLS90] 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.

[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 non-atomicity, or what happens once atomicity is given up for semantic atomicity. Unpublished manuscript, 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 University of Dartmund.

[MR91] P. Muth and T. C. Rakow. Atomic commitment for integrated database systems. In *Proceedings of the Seventh International Conference on Data Engineering, Kobe, Japan*, April 1991.

[RELL90] M. E. Rusinkiewicz, A. K. Elmagarmid, Y. Leu, and W. Litwin. Extending the transaction model to capture more meaning. *SIGMOD Record*, 19(1):3–7, March 1990.

[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.

[SGMA89] K. Salem, H. Garcia-Molina, and R. Alonso. Altruistic locking: A strategy for coping with long lived transactions. In D. Gawlick, M. Haynie, and A. Reuter, editors, *Lecture Notes in Computer Sciences, High performance Transaction Systems*, volume 359, pages 175–199. Springer-Verlag, 1989. Also available as CS-TR-087-87, Computer Science Department, Princeton University. A more recent version appears as UMIACS-TR-90-104, University of Maryland Institute for Advanced Computer Studies.

[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.