ON SERIALIZABILITY OF MULTIDATABASE TRANSACTIONS THROUGH FORCED LOCAL CONFLICTS

Dimitrios Georgakopoulos* and Marek Rusinkiewicz
Department of Computer Science
University of Houston
Houston, TX 77204-3475

Amit Sheth
Bellcore
444 Hoes Lane
Picataway, NJ 08854-4182

Abstract

The main difficulty in enforcing global serializability in a multidatabase environment lies in resolving indirect (transitive) conflicts between multidatabase transactions. Indirect conflicts introduced by local transactions are difficult to resolve because the the behavior or even the existence of local transactions is not known to the multidatabase system. To overcome these problems, we propose to incorporate additional data manipulation operations in the subtransactions of each multidatabase transaction. We show that if these operations create direct conflicts between subtransactions at each participating local database system, indirect conflicts can be resolved even if the multidatabase system is not aware of their existence. Based on this approach we introduce a multidatabase transaction management method that requires the local database systems to ensure only local serializability. The proposed method and its refinements do not violate the autonomy of the local database systems and guarantee global serializability by preventing multidatabase transactions from being serialized in different ways at the participating database systems.

1 Introduction

A *Multidatabase System* (MDBS) [17] is a facility that supports global applications accessing data stored in multiple databases. It is assumed that the access to these databases is controlled by autonomous and (possibly) heterogeneous *Local Database Systems* (LDBSs). The MDBS architecture allows *local transactions* and *global transactions* to coexist. Local transactions are submitted directly to a single LDBS, while the multidatabase (global) transactions are channeled through the MDBS interface. The objectives of multidatabase transaction management are to avoid inconsistent retrievals and to preserve the global consistency in the presence of multidatabase updates. These objectives are more difficult to achieve in MDBSs than in homogeneous distributed database systems because, in addition to the problems caused by *data distribution and replication* that all distributed database systems have to solve, transaction management in MDBSs must also cope with *heterogeneity* and *autonomy* of the participating LDBSs.

In a multidatabase environment, the serializability of local schedules is, by itself, not sufficient to maintain the multidatabase consistency. To assure that global serializability is not violated, local schedules must be validated by the MDBS. However, the local serialization orders are neither reported by the local database systems, nor can they be determined by controlling the submission of the global subtransactions or observing their execution order. To determine the serialization order of the global transactions at each LDBS, the MDBS must deal not only with *direct conflicts* that may exist between the subtransactions of multidatabase transactions but also with the *indirect conflicts* that may be caused by the local transactions. Since the MDBS has no information about the existence and behavior of the local transactions, determining if an execution of global and local transactions is globally serializable is difficult.

Several solutions have been proposed in the literature to deal with this problem, however, most of them are not satisfactory. The main problem with the majority of the proposed solutions is that they do not provide a way of assuring that the execution order of global transactions, which can be controlled by the MDBS, is reflected in their local serialization order produced by the LDBSs. In this paper we solve this problem by introducing a technique which disallows schedules in which a global transaction G_i is executed and committed by some LDBS before another transaction G_j, but their local serialization order is reversed. Our solution does not violate the local autonomy and

*Current address: GTE Laboratories, Waltham, MA 02254.

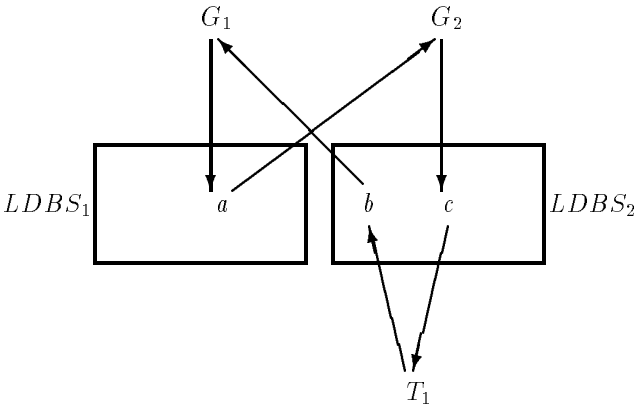
is applicable to all LDBSs that guarantee local serializability. We also discuss the class of schedules (and schedulers) in which the serialization order of each transaction can be determined by controlling its execution and commitment. We show that if the participating LDBSs use one of the many common schedulers that belong to this class, multidatabase transaction management is simplified.

The paper is organized as follows. In Section 2 we identify the difficulties in maintaining global serializability in MDBSs and review the related work. In Section 3 we introduce the concept of a subtransaction ticket and propose the *Optimistic Ticket Method* (OTM), for multidatabase transaction management. To guarantee global serializability, OTM requires that the LDBSs ensure local serializability. We also discuss the *Implicit Ticket Method* (ITM), a refinement of the OTM which reduces the overhead of OTM but works only for a subclass of the participating LDBSs (Section 3.4). Finally, in Section 4 we summarize our results.

2 Problems in maintaining global serializability and related work

Many algorithms that have been proposed for transaction management in distributed systems are not directly applicable in MDBSs because of the possibility of indirect conflicts caused by the local transactions. To illustrate this point let us consider Figure 1 which depicts two multidatabase transactions G_1 and G_2, and a local transaction T_1. If a transaction G_i reads a data item a, we draw an arc from a to G_i. An arc from G_i to a denotes that G_i writes a. In our example, the global transactions have subtransactions in both LDBSs. In LDBS$_1$, G_1 writes a and G_2 reads it. Therefore, G_1 and G_2 directly conflict in LDBS$_1$ and the serialization order of the transactions is $G_1 \rightarrow G_2$. In LDBS$_2$, G_1 and G_2 access different data items, i.e., G_1 reads b and G_2 writes c. Hence, there is no direct conflict between G_1 and G_2 in LDBS$_2$. However, since the local transaction T_1 writes b and and reads c, G_1 and G_2 conflict indirectly in LDBS$_2$. This indirect conflict is caused by the presence of the local transaction T_1. In this case, the serialization order of the transactions in LDBS$_2$ becomes $G_2 \rightarrow T_1 \rightarrow G_1$.

In a multidatabase environment the MDBS has control over the execution of global transactions and the operations they issue. Therefore, the MDBS can detect direct conflicts involving global transactions, such as the conflict between G_1 and G_2 at LDBS$_1$ in Figure 1. However, the MDBS has no information about local transactions and the indirect conflicts they may cause. For example, since the MDBS has no information about the local transaction T_1, it cannot detect the indirect conflict between G_1 and G_2 at LDBS$_2$. Although both local schedules are serializable, the schedule is globally non-serializable, i.e. there is no global order involving G_1, G_2 and T_1 that is compatible with both local schedules.



LDBS$_1$: $w_{G_1}(a) r_{G_2}(a), i.e., G_1 \rightarrow G_2$
LDBS$_2$: $w_{T_1}(b) r_{G_1}(b) w_{G_2}(c) r_{T_1}(c), i.e., G_2 \rightarrow T_1 \rightarrow G_1$

Figure 1: Serial execution of multidatabase transactions may violate serializability.

In the early work in this area the problems caused by indirect conflicts were not fully recognized. In their early paper, Gligor and Popescu-Zeletin [14], stated that a schedule of multidatabase transactions is correct if multidatabase transactions have the same relative serialization order at each LDBS they (directly) conflict. Breitbart and Silberschatz have shown [4] that the above correctness criterion is insufficient to guarantee global serializability in the presence of local transactions. They proved that the sufficient condition for the global consistency requires the multidatabase transactions to have the same relative serialization order in all sites they execute. The solutions to the problem of concurrency control in MDBSs that have been proposed in the literature can be divided into several groups:

Observing the execution of the global transactions at each LDBS [10]. The execution order of global transactions does not determine their relative serialization order at each LDBS. For example, at LDBS$_2$ in Figure 1 the global transaction G_1 is executed before G_2, but G_2 precedes G_1 in the local serialization order there. To determine local conflicts between transactions, Logar and Sheth [18] proposed using the commands of the local operating system and DBMS to "snoop" on the LDBS. Such approach may

be not always possible without violating the autonomy of the LDBS.

Controlling the submission and execution order of the global transactions. Alonso and Garcia-Molina proposed to use site locking in the *altruistic locking* protocol [1] to prevent undesirable conflicts between multidatabase transactions. Given a pair of multidatabase transactions G_1 and G_2, the simplest altruistic locking protocol allows the concurrent execution of G_1 and G_2 if they access different LDBSs. If there is a LDBS that both G_1 and G_2 need to access, G_2 cannot access it before G_1 has finished its execution there. Du and Elmagarmid [8] have shown that global serializability may be violated even when multidatabase transactions are submitted serially, one after the other, to their corresponding LDBS. The scenario in Figure 1 illustrates the above problem. G_1 is submitted to both sites, executed completely and committed. Only then G_2 is submitted for execution; nevertheless the global consistency may be violated.

Limiting multidatabase membership to LDBSs which use strict schedulers. By disallowing local executions which are serializable but not strict this approach places additional restrictions on the execution of both global and local transactions at each participating LDBS. A solution in this category, called the 2PC Agent Method, has been recently proposed in [22]. The 2PC Agent Method assumes that the participating LDBSs use two-phase locking (2PL) [11] schedulers and produce only strict [2] schedules. The basic idea in this method is that strict LDBSs will not permit local executions which violate global serializability. However, even local strictness is not sufficient. To illustrate this problem consider the LDBSs in Figure 1 and the following local schedules:

LDBS$_1$:$w_{G_1}(a) commit_{G_1} r_{G_2}(a) commit_{G_2}$, $G_1 \rightarrow G_2$
LDBS$_2$:$r_{G_1}(b) r_{G_2}(b) w_{G_1}(b) commit_{G_1} commit_{G_2}$, $G_2 \rightarrow T_1 \rightarrow G_1$

Both schedules above are strict and are allowed by 2PL. However, global serializability is violated.

Assume the possibility of conflicts among global transactions whenever they execute at the same site. This idea has been used by Logar and Sheth [18] in the context of distributed deadlocks in MDBSs and by Breitbart et al. [5] for concurrency control in the Amoco Distributed Database System (ADDS). Both are based on the notion of the *site graph*. In the ADDS method, when a global transaction issues a subtransaction to a LDBS, a node corresponding to it is included to the site graph. Furthermore, undirected edges are added to connect the nodes of the LDBSs that participate in the execution of each global transaction. If the addition of the edges for a global transaction does not create a cycle in the graph, multidatabase consistency is preserved and the global transaction is allowed to proceed. Otherwise, inconsistencies are possible and the global transaction is aborted.

The site graph method does not violate the local autonomy and correctly detects possible conflicts between multidatabase transactions. However, when used for concurrency control, it has significant drawbacks. First, the degree of concurrency allowed is rather low because multidatabase transactions cannot be executed at the same LDBS concurrently. Second, and more importantly, the MDBS using site graphs has no way of determining when it is safe to remove the edges of a committed global transaction. Consider global transactions G_1 and G_2. Suppose that G_2 is aborted because it may potentially conflict with G_1 which is currently executing as in Figure 1. If the edges corresponding to G_1 are removed immediately following its commitment and if G_2 is restarted, the global serializability may be violated. This is because a local transaction (e.g., T_1 in Figure 1) whose execution overlaps with the execution of the subtransactions of two global transactions may make the serialization order of global transaction different than their execution order. The method may work correctly if the removal of the edges corresponding to a committing transaction is delayed. However, the concurrency will be sacrificed. In the scenario in Figure 1, the edge corresponding to G_1 can be removed after the commitment of the local transaction T_1. However the MDBS has no way of determining the time of commitment or even the existence of the local transaction T_1.

Modifying the local database systems and/or applications. Pu [20] has shown that global serializability can be assured if the LDBSs present the local serialization orders to the MDBS. Since the traditional DBMSs usually do not provide their serialization order, Pu suggests modifying the LDBSs to provide it. Pons and Vilarem [19] suggest modifying existing applications so that all transactions (including the local) are channeled through multidatabase interfaces. Both methods mentioned here preserve the multidatabase consistency, but at the expense of partially violating the local autonomy.

Rejecting serializability as the correctness criterion. The concepts of *sagas* [15, 12] has been proposed do deal with long-lived transactions by releasing transaction atomicity and isolation. *Quasi-serializability* [7], assumes that no value dependencies exist among databases so indirect conflicts can be ignored. *S-transactions* [9] and *flexible transactions* [21] use transaction semantics to allow non-serializable ex-

ecutions of global transactions. These solutions do not violate the autonomy of the LDBSs and can be used, whenever the correctness guarantees they offer are applicable. In this paper we will assume that the global schedules must be serializable.

3 The Optimistic Ticket Method (OTM)

In this section we describe a method for multidatabase transaction management, called OTM, which does not violate the LDBS autonomy and guarantees global serializability if the participating LDBSs assure local serializability. The proposed method addresses two complementary issues:

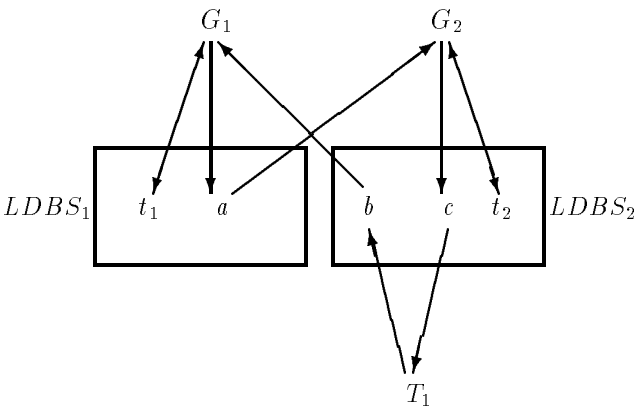
1. how the MDBS can obtain the information about the relative serialization order of subtransactions of global transactions at each LDBS, and
2. how the MDBS can guarantee that the subtransactions of each multidatabase transaction have the same relative serialization order in all participating LDBSs.

In the following discussion we do not consider site failures (commitment and recovery of multidatabase transactions are discussed, among others, in [6, 13]).

3.1 Determining the local serialization order

OTM uses *tickets* to determine the relative serialization order of the subtransactions of global transactions at each LDBS. A ticket is a (logical) timestamp whose value is stored as a regular data item in each LDBS. Each subtransaction of a global transaction is required to issue the *Take-A-Ticket* operation which consist of reading the value of the ticket (i.e., $r(ticket)$) and incrementing it (i.e., $w(ticket+1)$) through regular data manipulation operations. The value of a ticket and all operations on tickets issued at each LDBS are subject to the local concurrency control and other database constraints. Only a single ticket value is needed per LDBS. The Take-A-Ticket operation does not violate local autonomy because no modification of the local systems is required. Only the subtransactions of global transactions have to take tickets [1]; local transactions are not affected.

[1] This may create a "hot spot" in the LDBSs. However, since only subtransactions of multidatabase transactions and not local LDBS transactions have to compete for tickets, we do not consider this to be a major problem affecting the performance of our method. In fact, if the volume of global transactions is high it is likely that the value of the ticket can be read from the database buffer with minimal I/O overhead.



$\text{LDBS}_1 : r_{G_1}(t_1)w_{G_1}(t_1+1)w_{G_1}(a)r_{G_2}(t_1)w_{G_2}(t_1+1)$
$r_{G_2}(a), i.e., G_1 \rightarrow G_2$

$\text{LDBS}_2 : w_{T_1}(b)r_{G_1}(t_2)w_{G_1}(t_2+1)r_{G_1}(b)r_{G_2}(t_2)$
$w_{G_2}(t_2+1)w_{G_2}(c)r_{T_1}(c), i.e.,$
$G_2 \rightarrow T_1 \rightarrow G_1$

Figure 2: The effects of the Take-A-Ticket approach.

Figure 2 illustrates the effects of the Take-A-Ticket process on the example in Figure 1. The ticket data items at LDBS_1 and LDBS_2 are denoted by t_1 and t_2, respectively. In LDBS_1 the t_1 values obtained by the subtransactions of G_1 and G_2 reflect their relative serialization order. This schedule will be permitted by the local concurrency controller at LDBS_1. In LDBS_2 the local transaction T_1 causes an indirect conflict such that $G_2 \rightarrow T_1 \rightarrow G_1$. However, by requiring the subtransactions to take tickets we force an additional conflict $G_1 \rightarrow G_2$. This additional ticket conflict causes the execution at LDBS_2 to become locally non-serializable. Therefore, the local schedule:

$w_{T_1}(b)r_{G_1}(t_2)w_{G_1}(t_2+1)r_{G_1}(b)\ r_{G_2}(t_2)\ _{G_2}(t_2+1)$
$w_{G_2}(c)r_{T_1}(c)$

will be not allowed by the local concurrency control (i.e., the subtransaction of G_1 or the subtransaction of G_2 or T_1 will be blocked or aborted). On the other hand, if the local schedule in LDBS_2 were for example:

$r_{G_1}(t_2)w_{G_1}(t_2+1)r_{G_1}(b)r_{G_2}(t_2)\ w_{G_2}(t_2+1)\ w_{T_1}(b)$
$w_{G_2}(c)r_{T_1}(c)$

the tickets obtained by G_1 and G_2 would reflect their relative serialization order there and the local schedule would be permitted by the local concurrency control at LDBS_2. Theorem 1 formally proves that the tickets obtained by the subtransactions at each LDBS are guaranteed to reflect their relative serialization order.

Theorem 1 *The tickets obtained by the subtransactions of multidatabase transactions determine their relative serialization order.*

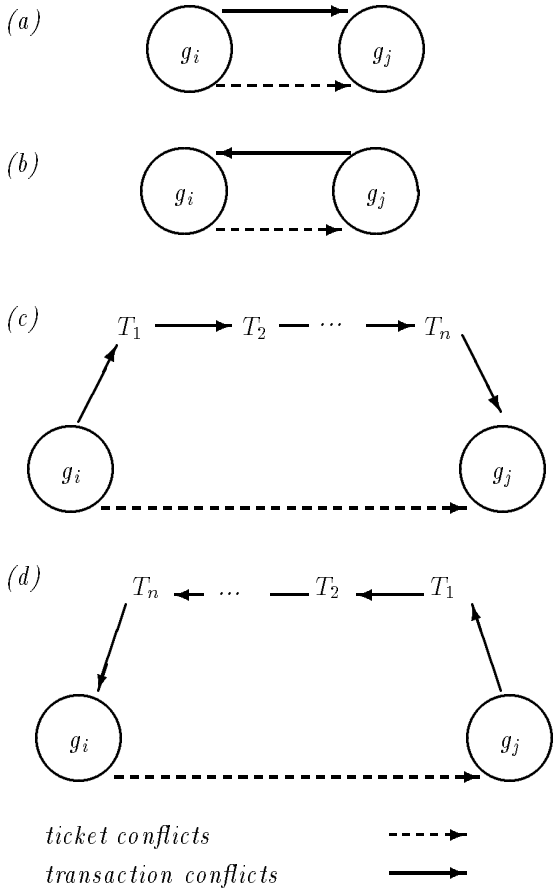


Figure 3: The effects of ticket conflicts in OTM.

Proof: Let g_i and g_j be the subtransactions of global transactions G_i and G_j, respectively, at some LDBS. Without loss of generality we can assume that g_i takes its ticket before g_j, i.e., $r_{g_i}(ticket)$ precedes $r_{g_j}(ticket)$ in the local execution order. Since a subtransaction takes its ticket first and then increments the ticket value, only the following execution orders are possible:
E_1:$r_{g_i}(ticket)r_{g_j}(ticket)w_{g_i}(ticket+1)w_{g_j}(ticket+1)$
E_2:$r_{g_i}(ticket)r_{g_j}(ticket)w_{g_j}(ticket+1)w_{g_i}(ticket+1)$
E_3:$r_{g_i}(ticket)w_{g_i}(ticket+1)r_{g_j}(ticket)w_{g_j}(ticket+1)$
However, among these executions only E_3 is serializable and can be allowed by the LDBS concurrency control. Therefore, g_i increments the ticket value before g_j reads it and g_i obtains a smaller ticket than g_j.

To show now that g_i can only be serialized before g_j, it is sufficient to point out that the operations to take and increment the ticket issued first by g_i and then by g_j create a direct conflict $g_i \rightarrow g_j$. This direct conflict forces g_i and g_j to be serialized according to the order in which they take their tickets. More specifically, if there is another direct conflict between g_i and g_j, such that $g_i \rightarrow g_j$ (Figure 3 (a)) or indirect conflict caused by local transactions, such that $g_i \rightarrow T_1 \rightarrow T_2 \ldots \rightarrow T_n \rightarrow g_j (n \geq 1)$ (Figure 3 (c)), the resulting schedule is serializable and both g_i and g_j are allowed to commit. In this case, g_i is serialized before g_j and this is reflected by the order of their tickets. However, if there is an indirect conflict $g_j \rightarrow T_1 \rightarrow T_2 \ldots \rightarrow T_n \rightarrow g_i (n \geq 1)$ (Figure 3 (d)), the ticket conflict $g_i \rightarrow g_j$ creates a cycle in the local serialization graph. Hence, this execution becomes non-serializable and is not allowed by the LDBS concurrency control. Therefore, indirect conflicts can be resolved through the use of tickets by the local concurrency control even if the MDBS cannot detect their existence. □

Case (b) in Figure 3 is explained separately in Section 3.3.

3.2 Enforcing global serializability

To maintain global consistency, OTM must ensure that the subtransactions of each global transactions have the same relative serialization order in their corresponding LDBSs [4]. Since, the relative serialization order of the subtransactions at each LDBS is reflected in the values of their tickets, the basic idea in OTM is to allow the subtransactions of each global transaction to proceed but commit them only if their ticket values have the same relative order in all participating LDBSs. This requires that the local database systems support a visible *prepared to commit state* for all subtransactions of global transactions. We say that a transaction enters its prepared to commit state when it completes the execution of its operations and leaves this state when it is committed or aborted. During this time, all updates reside in its private workspace and are installed in the database when the transaction is committed. The prepared to commit state is *visible* if the application program can decide whether the transaction should commit or abort. Many database management systems, designed using the client-server architecture (e.g., SYBASE) provide a visible prepared to commit state and can directly participate in a multidatabase commitment. However, even if the prepared state is not explicitly supported by the local systems, it can be simulated by forcing a handshake after each read and write operation [13]. Under this assumption, a transaction enters its (simulated) prepared to commit state when the completion of its last operation is acknowledged.

OTM processes a multidatabase transaction G as follows. Initially, it sets a timeout for G and submits its subtransactions to their corresponding LDBSs. All subtransactions are allowed to interleave under the control of the LDBSs until they enter their prepared

to commit state. If they all enter their prepared to commit states, they wait for the OTM to validate G. The validation can be performed using a *Global Serialization Graph* (GSG) test. The nodes in GSG correspond to "recently" committed global transactions. In its simplest form, the set of recently committed global transactions in OTM does not contain transactions committed before the oldest of the currently active global transactions started its execution. For any pair of recently committed global transactions G_i^c and G_j^c, GSG contains a directed edge $G_i^c \rightarrow G_j^c$ if at least one subtransaction of G_i^c was serialized before (obtained a smaller ticket than) the subtransaction of G_j^c in the same LDBS. Similarly, if the subtransaction of G_j^c in some LDBS was serialized before the subtransaction of G_i^c a directed edge $G_i^c \leftarrow G_j^c$ connects their nodes in GSG.

Initially, GSG contains no cycles. During the validation of G, OTM first creates a node for G in GSG. Then, it attempts to insert edges between G's node and nodes corresponding to every recently committed multidatabase transaction G^c. More specifically, if the ticket obtained by a subtransaction of G at some LDBS is smaller (larger) than the the ticket of the subtransaction of G^c there, an edge $G \rightarrow G^c$ ($G \leftarrow G^c$) is added to GSG. If all such edges can be added without creating a cycle in GSG, G is validated. Otherwise, G does not pass validation, its node together with all incident edges is removed from the graph and G is restarted. This validation test is enclosed in a single critical section [2].

G is also restarted, if at least one LDBS forces a subtransaction of G to abort for local concurrency control reasons (e.g., local deadlock), or its timeout expires (e.g., global deadlock). Alternatively, OTM may set a new timeout and restart only the subtransactions that did not report prepared to commit in time. If more than one of the participating LDBSs uses a blocking mechanism for concurrency control, the timeouts above are necessary to resolve global deadlocks. An alternative approach is to maintain a *wait-for graph* (WFG) having LDBS as nodes. Then, if a cycle is found in the WFG and the cycle involves LDBS that use a blocking technique to synchronize conflicting transactions, a deadlock is possible. Dealing with deadlocks in MDBSs constitutes a problem for further research [18, 6].

Theorem 2 *OTM guarantees global serializability if the following conditions are satisfied by the LDBSs:*

1. *The concurrency control mechanisms of the LDBSs assure local serializability.*
2. *Each multidatabase transaction has at most one subtransaction at each* LDBS.
3. *Each subtransaction has a visible prepare to commit state.*

Proof: We have already shown that the order in which the subtransactions take their tickets reflects their relative serialization order (Theorem 1). After the tickets are obtained by a global transaction at all sites it executes, OTM performs the global serialization test described in earlier in this section. Global transactions pass validation and are allowed to commit only if their relative serialization order is the same at all participating LDBSs which guarantees global serializability. □

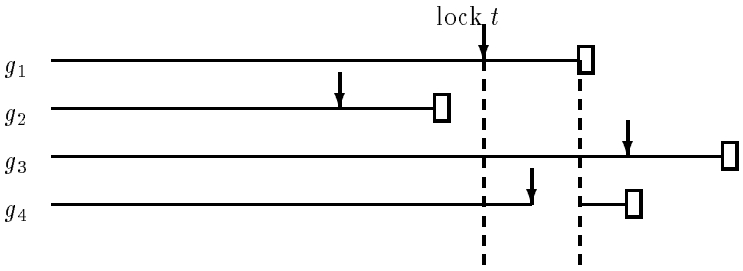
3.3 Effect of the Ticketing Time on the Performance of OTM

OTM can process any number of multidatabase transactions concurrently, even if they conflict at multiple LDBS. However, since OTM forces the subtransactions of multidatabase transactions to directly conflict on the ticket, it may cause some subtransactions to get aborted or blocked because of ticket conflicts (Figure 3 (b)). Since subtransactions may take their tickets at any time during their lifetime without affecting the correctness of OTM, optimization based on the characteristics of each subtransaction (e.g. number, time and type of the data manipulation operations issued or their semantics) is possible. For example, if all global transactions conflict directly at some LDBS, there is no need for them to take tickets. To determine their relative serialization order there, it is sufficient to observe the order in which they issue their conflicting operations.

The appropriate choice of the point in time to take the ticket during the lifetime of a subtransaction can minimize the synchronization conflicts among subtransactions. For instance, if a LDBS uses 2PL it is more appropriate to take the ticket immediately before a subtransaction enters its prepared to commit state. To show the effect of this convention consider a LDBS that uses 2PL for local concurrency control (Figure 4 (a)). 2PL requires that each subtransaction sets a write lock on the ticket before it increments its value. Given four concurrent subtransactions g_1, g_2, g_3 and g_4, g_1 does not interfere with g_2 which can take its ticket and commit before g_1 takes its ticket. Similarly, g_1 does not interfere with g_3, so g_1 can take its ticket and commit before g_3 takes its ticket. However, when g_4 attempts to take its ticket after g_1 has taken its ticket but before g_1 commits and releases its ticket lock, it gets blocked until g_1 is committed. The

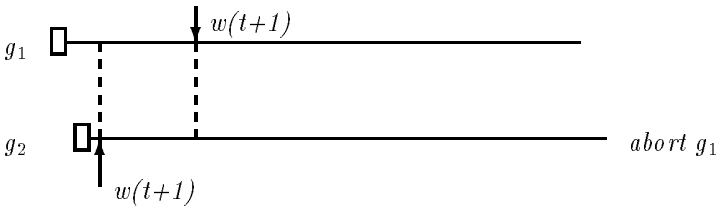
[2] Other validation tests such as the certification scheme proposed in [20] can be also used to validate global transactions.

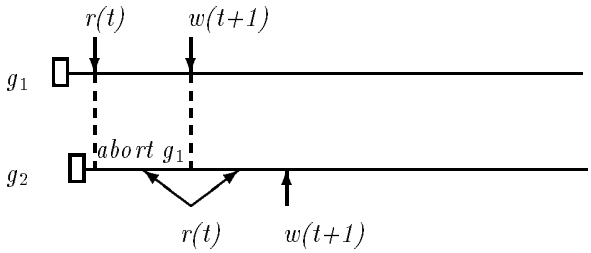
(a) Preferred ticketing in a LDBS using 2PL



(b) Preferred ticketing in a LDBS using TO

let $\text{ts}(g_1) < \text{ts}(g_2)$





(c) Preferred ticketing in a LDBS using an optimistic protocol

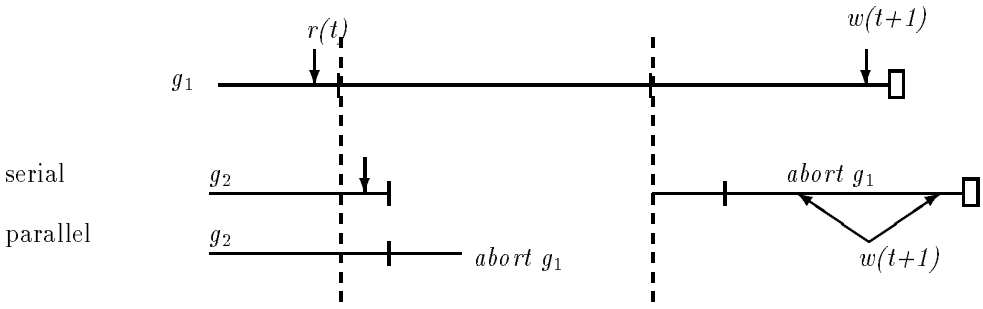


Figure 4: Preferred ticketing in LDBSs.

ticket values always reflect the serialization order of the subtransactions of multidatabase transactions but the ticket conflicts are minimized if the time when g_1 takes its ticket is as close as possible to its commitment time.

If a LDBS uses timestamp ordering (TO) [2] (Figure 4 (b)), it is better to obtain the ticket when the subtransaction begins its execution. More specifically, TO assigns a a timestamp $ts(g_1)$ to a subtransaction g_1 when it begins its execution. Let g_2 be another subtransaction such that $ts(g_1) < ts(g_2)$. If the ticket obtained by g_1 has a larger value than the ticket of g_2 then g_1 is aborted. Clearly, if g_2 increments the ticket value before g_1 then, since g_2 is younger than g_1, either $r_{g_1}(ticket)$ or $w_{g_1}(ticket)$ conflicts with the $w_{g_2}(ticket)$ and g_1 is aborted. Hence, only g_1 is allowed to increment the ticket value before g_2. Similarly, if g_2 reads the ticket before g_1 increments it, then when g_1 issues $w_{g_1}(ticket)$ it conflicts with the $r_{g_2}(ticket)$ operation issued before and g_1 is aborted. Therefore, given that $ts(g_1) < ts(T_2)$, either g_1 takes its ticket before g_2 or it is aborted. Therefore, its is better for subtransactions to take their tickets as close as possible to the point they are assigned their timestamps under TO, i.e., at the beginning of their execution.

Finally, if a LDBS uses an optimistic [16] protocol which uses transaction readsets and writesets to validate transactions, there is no best time for the subtransactions to obtain their tickets (Figure 4 (c)). Each subtransaction g_1 reads the ticket value before it starts its (serial or parallel) validation but increments it at the end of its write phase. If another transaction g_2 is able to increment the ticket in the meantime, g_1 is restarted.

The basic advantage of OTM is that it requires the local systems to ensure only local serializability. It main disadvantage is that it introduces additional conflicts between global transactions which may not conflict otherwise. If additional assumptions can be made, concerning the schedules that are produced by the local systems, the OTM can be simplified and the ticket conflicts can be eliminated.

3.4 Implicit tickets, a refinement for rigorous LDBSs

As we have discussed, the basic problem in multidatabase concurrency control is that the local serialization orders do not necessarily reflect the order in which global transactions are submitted, executed and committed in the LDBSs. In this section we show that if the LDBSs, do not produce schedules with such anomalies, the MDBS can determine the local serialization order by controlling the execution of global transactions.

To simplify transaction processing and recovery the transaction management mechanisms in most DBMSs, produce schedules that are not only serializable, but also cascadeless or strict [2]. Under a strict scheduler, no transaction can read or write a data item until all transactions that previously wrote it commit or abort. In [3] we have introduced the concept of *rigorous* transaction processing mechanism. In addition of guaranteeing strictness, a rigorous scheduler does not allow transactions to write a data item until the transactions that previously read it either commit or abort. That is, the notion of rigorousness effectively eliminates conflicts between uncommitted transactions. In [3] we have also shown that the class of rigorous transaction management mechanisms includes several common transaction management mechanisms, such as, conservative TO [2], the optimistic protocol with serial validation [16], and a variant of strict two-phase locking (2PL) under which a transaction must hold its read and write locks until it terminates.

The point out the importance of rigorousness in a multidatabase environment consider a MDBS in which the transaction management mechanisms of all LDBSs are rigorous. In such a multidatabase environment, the MDBS can determine the serialization order of global transactions by controlling the submission and execution order of their subtransaction at the participating LDBSs. In particular, we have shown [3] that rigorous schedulers guarantee that for any pair of transactions T_i and T_j, such that T_i is committed before T_j, T_i also precedes T_j in the serialization order corresponding to the execution schedule. It should be observed that strictness of the local transaction management mechanisms is not sufficient to assure this property.

To take advantage of rigorous LDBSs, we introduce a refinement of OTM, called the *Implicit Ticket Method* (ITM). ITM takes advantage of the fact that if all LDBSs produce rigorous schedules then ticket conflicts can be eliminated. To guarantee global serializability in the presence of local transactions, ITM requires the following condition to be satisfied in addition to Conditions 2 and 3 which are stated in the Theorem presented in Section 3.2 (Rigorous schedules are serializable [3]; therefore, Condition 4 can replace Condition 1):

4. All local database systems use rigorous transaction management mechanisms.

Like OTM, ITM ensures global serializability by preventing the subtransactions of each multidatabase transaction from being serialized in different ways at their corresponding LDBSs. Unlike OTM, ITM does

not need to maintain tickets and the subtransactions of global transactions do not need to explicitly take and increment tickets. In a rigorous LDBS, the implicit ticket of each subtransaction executed there is determined by its commitment order. That is, the order in which we commit subtransactions at each LDBS determines the relative values of their implicit tickets. To achieve global serializability, ITM controls the commitment (execution) order and thus the serialization order of multidatabase subtransactions as follows. Let G_i and G_j be two multidatabase transactions. Assuming rigorous LDBSs, ITM guarantees that in all participating LDBSs either the subtransactions of G_i are committed before the subtransactions of G_j or the subtransactions of G_j are committed prior to the subtransactions of G_i.

ITM achieves this objective as follows. Initially, the MDBS sets timeouts for G_i and G_j and submits their subtransactions to the corresponding LDBSs. All subtransactions are allowed to interleave under the control of the LDBSs until they enter their prepared to commit state. If all subtransactions of G_i report prepared to commit to the ITM before the subtransactions of G_j do, the ITM commits each subtransaction of G_i before any subtransaction of G_j. If the subtransactions of G_j are prepared to commit first, each subtransaction of G_j is committed before any subtransaction of G_i. If neither of these happens, the MDBSs aborts and restarts any multidatabase transaction that has subtransactions which did not report their prepared to commit state before the timeout expired.

Theorem 3 *ITM ensures global serializability if all LDBSs satisfy conditions 2, 3 and 4.*

Proof: Given two multidatabase transactions G_i and G_j, ITM commits each subtransaction of G_i before the corresponding subtransaction of G_j or vice versa. In the beginning of this section we explained that in rigorous LDBSs the commitment order of each subtransaction (implicit ticket order) determines its relative serialization order. Therefore, all subtransactions of each multidatabase transaction are serialized the same way in their corresponding LDBSs. □

Although ITM works only for rigorous LDBSs it can be combined with OTM into a single comprehensive mechanism where OTM is first used to synchronize the subtransactions in all non-rigorous LDBSs and then ITM is applied to ensure global serializability of remaining subtransactions.

4 Summary and conclusion

Enforcement of serializability of global transactions in a MDBS environment is much harder than in distributed databases systems. The additional difficulties in this environment are caused by the autonomy and the heterogeneity of the participating LDBSs.

To enforce global serializability we introduced OTM, an optimistic multidatabase transaction management mechanism that permits the commitment of multidatabase transactions only if their relative serialization order is the same in all participating LDBSs. OTM requires the LDBSs to guarantee only local serializability. The basic idea in OTM is to create direct conflicts between multidatabase transactions at each LDBS that allow us to determine the relative serialization order of their subtransactions. ITM is a refinement of OTM that uses implicit tickets and eliminates ticket conflicts, but works only when the participating LDBSs use rigorous transaction scheduling mechanisms. ITM uses the local commitment order of each subtransaction to determine its implicit ticket value. It achieves global serializability by controlling the commitment (execution order) and thus the serialization order of multidatabase transactions. Compared to the the ADDS approach and Altruistic Locking, ITM can process any number of multidatabase transactions concurrently, even if they have concurrent and conflicting subtransactions at multiple sites. Both OTM and ITM do not violate the autonomy of the LDBSs and can be combined in a single comprehensive mechanism.

Rigorousness is a very useful property in a MDBS. For example, it can be shown that ADDS scheme [4, 5], Altruistic Locking [1] and 2PC Agent Method [22] produce globally serializable schedules if the participating LDBSs are rigorous. Similarly, quasi-serializable schedules [8] become serializable if all the LDBSs are rigorous. On the other hand, if the local systems are not rigorous some of the above methods may lead to schedules that are not globally serializable.

Acknowledgments

The idea to use tickets in multidatabase transaction management had emerged during a discussion with Gomer Thomas. We thank Yuri Breitbart for pointing out an error in one of our definitions in an earlier version of this paper.

References

[1] R. Alonso, H. Garcia-Molina, and K. Salem. Concurrency control and recovery for global procedures in federated database systems. *A quarterly bulletin of the Computer Society of the IEEE technical committee on Data Engineering*, 10(3), September 1987.

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

[3] Y. Breitbart, D. Georgakopoulos, M. Rusinkiewicz, and A. Silberschatz. Rigorous scheduling in multidatabase systems. In *Submitted to: Workshop in Multidatabases and Semantic Interoperability*, October 1990.

[4] Y. Breitbart and A. Silberschatz. Multidatabase update issues. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, June 1988.

[5] Y. Breitbart, A. Silberschatz, and G. Thompson. An update mechanism for multidatabase systems. *A quarterly bulletin of the Computer Society of the IEEE technical committee on Data Engineering*, 10(3), September 1987.

[6] Y. Breitbart, A. Silberschatz, and G. Thompson. Reliable transaction management in a multidatabase system. Technical Report 157-90, University of Kentucky, 1990.

[7] W. Du and A. Elmagarmid. QSR: A correctness criterion for global concurrency control in InterBase. In *Proceedings of the 15th International Conference on Very Large Databases*, 1989.

[8] W. Du, A. Elmagarmid, Y. Leu, and S. Osterman. Effects of autonomy on maintaining global serializability in heterogeneous distributed database systems. In *Proceedings of the second International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, October 1989.

[9] F. Eliassen, J. Veijalainen, and H. Tirri. Aspects of transaction modeling for interoperable information systems. Technical report, Report of the COST Project, 1988.

[10] A.K. Elmagarmid and A.A. Helal. Supporting updates in heterogeneous distributed database systems. In *IEEE Proceedings of the 4th International Conference on Data Engineering*, 1988.

[11] K.P. Eswaran, J.N. Gray, R.A. Lorie, and I.L. Traiger. The notions of consistency and predicate locks in a database system. *Communications of ACM*, 19(11), November 1976.

[12] H. Garcia-Molina and K. Salem. SAGAS. In *Proceedings of ACM SIGMOD Conference on Management of Data*, 1987.

[13] D. Georgakopoulos and M. Rusinkiewicz. Transaction management in multidatabase systems. Technical Report UH-CS-89-20, Department of Computer Science, University of Houston, September 1989.

[14] V.D. Gligor and R. Popescu-Zeletin. Concurrency control issues in distributed heterogeneous database management systems. In F.A. Schreber and W. Litwin, editors, *Distributed Data Sharing Systems*. North-Holland, 1985.

[15] J.N. Gray. The transaction concept: Virtues and limitations. In *Proceedings of the 7th International Conference on VLDB*, September 1981.

[16] H.T. Kung and J.T. Robinson. On optimistic methods for concurrency control. *ACM TODS*, 6(2), June 1981.

[17] W. Litwin. From database systems to multidatabase systems: Why and how. In *British National Conference on Databases*. Cambridge Press, 1988.

[18] T. Logan and A. Sheth. Concurrency control issues in heterogeneous distributed database management systems. Unpublished, July 1986.

[19] J. Pons and J. Vilarem. Mixed concurrency control: Dealing with heterogeneity in distributed database systems. In *Proceedings of the Fourteenth International VLDB Conference*, August 1988. Los Angeles.

[20] C. Pu. Superdatabases for composition of heterogeneous databases. In *IEEE Proceedings of the 4th International Conference on Data Engineering*, 1988.

[21] M. Rusinkiewicz, A. Elmagarmid, Y. Leu, and W. Litwin. Extending the transaction model to capture more meaning. *SIGMOD record*, 19(1), March 1990.

[22] A. Wolski and J. Veijalainen. 2PC Agent method: Achieving serializability in presence of failures in a heteroneous multidatabase. In *Proceedings of PARBASE-90 Conference*, February 1990.