Transaction Scheduling in Dynamic Composite Multidatabase Systems *

Dexter P. Bradshaw Per-Åke Larson
Jacob Slonim †
Department of Computer Science
University of Waterloo, Waterloo, Ontario N2L 3G1
Technical Report: CS-94-11

Abstract

Multidatabase systems based on a single monolithic multidatabase server are not realistic. Their performance and administration do not scale with increases in the radius of service or the number of component databases under their control. We propose that a composite multidatabase architecture that consists of multiple, possibly heterogeneous, peer multidatabase servers distributed on a communications network is inevitable. Global transactions should be able to span multiple multidatabase servers, sometimes forcing multidatabase servers to act as component database systems. Particular focus is given to the problem of guaranteeing the correct execution of interleaving global transactions across multiple multidatabase systems. Correctness is based on global serializability. Three algorithms for maintaining global serializability through transaction ordering during dynamic multidatabase composition are proposed. We examine the restrictions of these algorithms and the scalability of their performance.

Keywords: multidatabase composition, serializability, concurrency control, rigorous scheduling, timestamp ordering.

1 Introduction

A *multidatabase system (MDBS)* is a service that provides access to data stored in multiple autonomous and possibly heterogeneous database systems distributed on a communications network.

*This research is supported by IBM Canada Ltd. and by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

†Jacob Slonim is an adjunct Professor at the University of Waterloo. His mailing address is at the Centre for Advanced Studies, IBM Toronto Labs, 844 Don Mills, North York, Ontario, M3C 1V7.

To client applications multidatabase systems provide an integrated view of the data stored at component database systems under their control. The abstraction gives applications the illusion that they are accessing a single database system. This removes the complexity of distribution, heterogeneity, integration, transaction management, and administration from the application to the multidatabase system.

A centralized view of a single multidatabase server defeats the transparency aspect of the multidatabase abstraction. With a single centralized server, applications are exposed to the nuances of accessing data sources that cross multidatabase domains. We propose to maintain transparency by configuring a multidatabase system as dynamic sets of cooperating heterogeneous peer servers distributed on a communications network. We refer to this configuration as a *composite multidatabase system*. Also, we call each multidatabase system participating in a composition a *multidatabase cell*, or simply a *cell*. During composition, global data requests can span multiple cells composing multidatabase servers, dynamically and arbitrarily, as they are delegated from one multidatabase cell to the next.

Multidatabase composition provides a flexible framework for configuring and planning the evolution of component database systems in a network domain. Issues such as administration and control, security, reliability through redundancy, performance and load balancing, and system scalability also motivate concept of multidatabase composition. We claim that decentralized composite multidatabase architectures are inevitable since *closed* centralized architectures restrict functionality, transparency, reliability, and performance.

A Composition Example

We further motivate the usefulness and practicality of multidatabase composition through a travel example. Consider a traveler in Trinidad booking a flight to Vancouver with a stop-over in Toronto. In Vancouver, the traveler simply needs to stay at a hotel. However, in Toronto, the traveler needs a hotel room and the convenience of a rented vehicle. Figure 1 illustrates the flow of requests generated by the transaction, T, that books the trip itinerary.

The access pattern of T dynamically composes multidatabase cells at the Trinidad branch of Agency A, the Toronto branch of Agency A, and Agency B. T is submitted to the multidatabase server for Agency A in Trinidad. The original request is decomposed into subtransactions T_1, T_2, and T_4 that are submitted to component databases to update client data, book an airline seat and bill the customer's VISA account, respectively. Since the local branch does not have direct access to Canadian databases, all bookings in Canada are forwarded to the multidatabase server at the branch office in Toronto through transaction T_3.

The multidatabase server at Agency A, in Toronto, treats the forwarded subtransaction like

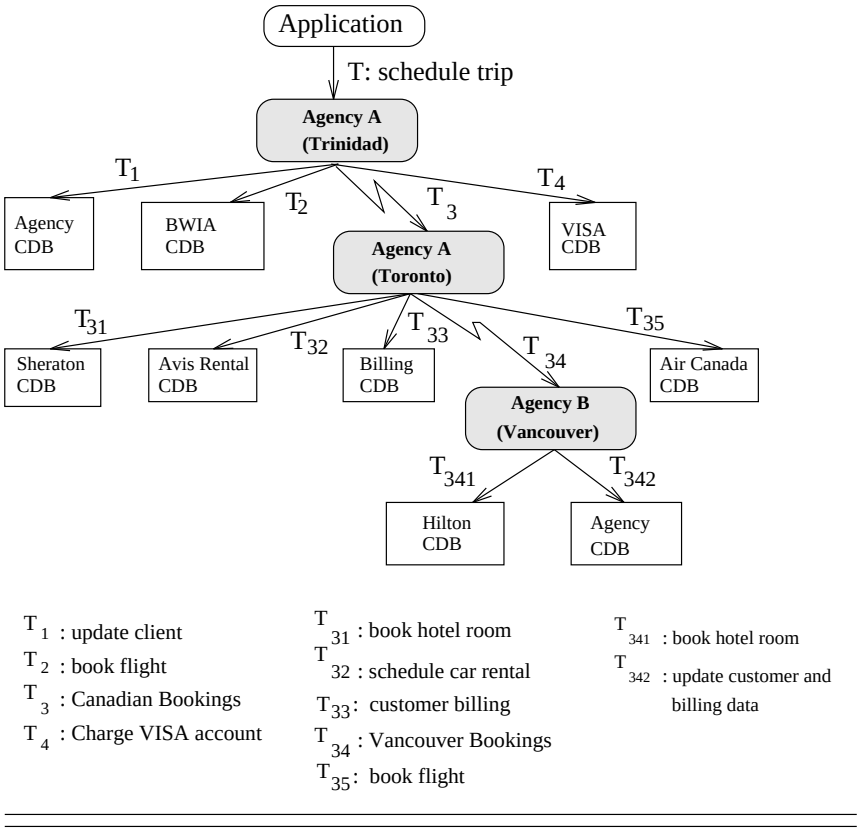

Figure 1: An example of multidatabase composition.

any other global transaction from an application using the service. The forwarded transaction T_3 is broken down into subtransactions T_{31}, T_{32}, T_{33}, and T_{35}. These transactions represent a hotel room booking, a car rental reservation, a customer billing and an airline reservation at component databases accessible from the Toronto branch office. The other subtransaction, T_{34}, is forwarded to Agency B in Vancouver where hotel reservations and customer updates are made directly to component systems through the subtransactions T_{341} and T_{342}, respectively.

Objective and Overview

This work focuses on concurrency control algorithms for guaranteeing the correctness of interleaved global transactions in a composite multidatabase environment. Our correctness criterion is based on serializability. We show that by implicitly or explicitly imposing ordering constraints on global transactions that span more than one multidatabase cell, we can still guarantee global serializability. The concurrency control algorithms need no changes to underlying multidatabase

concurrency control algorithms. However, they may impose some conditions on the histories at component database systems within participating cells. These algorithms permit multidatabase cells to act as peers and be composed dynamically at runtime according to the access patterns of global transactions. The performance of the algorithms is compared with those presented in related work.

We begin, in Section 2, with a review of previous work on multidatabase concurrency control and composition, and show how this work differs. In Section 3, we present a reference architecture for composite multidatabase transaction management, some terminology, and basic assumptions. The problem of scheduling global transactions in a composite multidatabase environment is introduced in Section 4. Section 5 outlines a set of algorithms for dealing with the scheduling problem. Section 6 highlights the effect of multi-cell transaction ordering on performance in composite systems. We conclude, in Section 7, with a summary of this work and its significance.

2 Related Work

The only known published work on multidatabase composition is on *superdatabases* [18]. A superdatabase is analogous to a composite multidatabase system. Superdatabases form a static hierarchy with a *master* superdatabase at the root composed of other superdatabases and component database systems. Each intermediate superdatabase has its own superdatabase hierarchy with component database systems occupying the leaves. Concurrency control is handled optimistically by certifying the serialization orders of the subtransactions of a global transaction with that of the subtransactions previously committed global transactions at the root. If the orders are consistent, the transaction is committed, otherwise, certification fails and it is aborted. Transactions orders at any intermediate superdatabase is maintained through a data structure called an order vector, or *O-vector*. Before a transaction can be committed its order vectors must all be shipped from leaf superdatabases to the root through intermediate superdatabases for certification.

Because the concurrency control algorithm is optimistic the serialization point of a global transaction is only known at the end of its execution. Hence, detection of a violation in the global serialization order is only known at end of a transaction, and later global aborts can cause an excessive waste of resources. The algorithms we present stress dynamic composition and determine serialization points much earlier in their execution.

Usually, the correctness of multidatabase algorithms is based on the classical notion of serializability. Several algorithms have been proposed in the literature to guarantee serializability and usually fall into one of three classes of mechanisms: exclusion, strong recoverability (or commitment ordering) [6, 9, 21] and rigorous scheduling [9, 13].

Exclusion prevents more than one transaction from executing at two or more sites, inherently preventing indirect conflicts through local transactions [1, 8, 23]. Concurrency control schedulers based on strong recoverability ensure that the serialization and commit orders at component database sites are equivalent. Certification of consistent serialization orders before atomic commitment guarantees serializability [19, 20, 21]. Rigorousness is a subclass of strong recoverability in which component databases guarantee that a transaction cannot execute a conflicting operation on data items until the transaction that last operated on them is committed or aborted. Rigorousness combined with an atomic commit protocol guarantees serializability [9, 13].

In Bradshaw [5], work on concurrency control in open nested transaction systems [3] was combined with the notion of multidatabase serializability [17] to derive the concept of *open nested multidatabase serializability (ONMSR)*. ONMSR is not a new serializability class. It generalizes two-level multidatabase serializability to asymmetric execution hierarchies of arbitrary depths. A composite multidatabase is ONMSR if all leaf component databases guarantee serializable histories and all multidatabases, participating in the composition, are multidatabase serializable. ONMSR combined with upward and downward ordering compatibilities in an execution hierarchy guarantees global serializability.

3 Composite Multidatabase Model

The model for transaction management in composite multidatabase comprises a transaction processing architecture and assumptions. The former describes abstract modules, their functions and interfaces for transaction processing, while the former describes expected characteristics at module interfaces.

Transaction Processing Architecture

A multidatabase cell consists of a multidatabase (MDB) server and a finite set of component database systems (CDB) under its control. Multidatabase servers and component database systems run as separate processes, or as a separate group of processes, on nodes (or sites) distributed on a communications network. Multidatabase cells can communicate directly, or indirectly through other intermediate multidatabase servers. Component database systems can only communicate with their multidatabase server through a multidatabase processing agent (MPA).

A multidatabase transaction processing system consists of a global transaction manager (GTM), a set of local transaction managers (LTM) belonging to the component systems, and multidatabase transaction processing agents (MTA) running as part of the MPA. The GTM coordinates the execution of global transactions at component sites through the MTA. The LTM schedules all

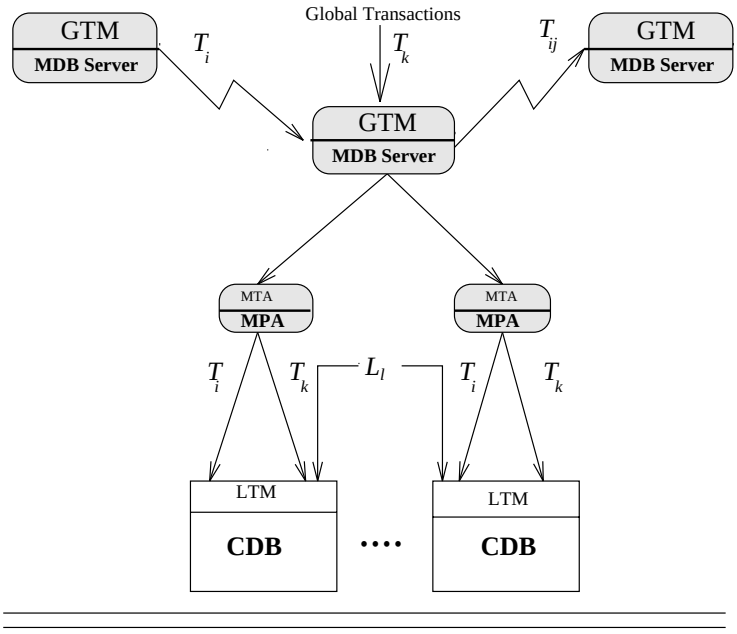

Figure 2: Composite multidatabase transaction processing model.

local transactions and global subtransactions local to a component database system. The MTA is responsible for mapping global subtransactions into local subtransactions that are scheduled for execution by an underlying LTM. Interleaving concurrent global transactions are scheduled by a component of the GTM called the global concurrency controller (GCC). The GCC cooperates with an global currency control agent (GCCA) in the MTA to guarantee correct global execution schedules at component database systems. This transaction processing architecture is illustrated in Figure 2.

During composition, global subtransactions are forwarded from one multidatabase cell to another to complete the work of a global transaction. At a multidatabase cell, subtransactions from other cells are treated like any other transaction from an application attempting to access the multidatabase. Therefore, composition is transparent to a multidatabase server that may act in isolation, or participate in a composite environment.

In general, we distinguish four types of transactions: simple global transactions, multi-cellular global transactions, delegated subtransactions, and local transactions. A *simple* global transaction executes is a global transaction that executes within one database cell. A *multi-cellular* global transaction begins in a single cell but may span multiple multidatabase cells through delegated subtransactions. Simple and multi-cellular global transactions are collectively called *global transactions*. *Delegated* subtransactions are global subtransactions forwarded from a global transaction

at one cell to execute at another. *Local* transactions execute solely at a single component database system. From Figure 2, T_i and T_k are multi-cellular and simple global transactions, respectively. T_{ij} is a delegated subtransaction, while L_l is a local transaction.

Assumptions

We assume that multidatabase cell servers communicate with each other and can participate in an atomic commit protocol. The composite multidatabase system architecture is peer-to-peer in that the client/server relationship between multidatabase cell servers can reverse depending on the multidatabase cell forwarding the transactional work. Hence, multidatabase composition is *not* static, but depends on the run-time structure of active multidatabase transactions.

No two multidatabase cells share the same component database system. We restrict multidatabase access to a component database system through a single multidatabase server. Applications from other multidatabase cells can only access a component database in a another multidatabase cell through delegated subtransactions to its multidatabase cell server. We further restrict the environment by assuming no shared or replicated data and data structures exist among multidatabase cells. We refer to this assumption as the *shared-nothing* assumption. It implies that distributed or replicated component databases are treated as single component database systems. The catalogue in a multidatabase server may import mappings to views and base tables managed by another multidatabase server.

All multidatabase cells guarantee multidatabase serializable histories [17]. Since multidatabase serializability is a more restrictive form of two-level serializability, this subsumes locally serializable schedules at all component database systems. In particular, we assume that local schedules are strict and global atomicity between a multidatabase server and its component sites is guaranteed with the 2PC protocol [4, 14, 15]. We also assume that local deadlock is resolved by deadlock resolution mechanisms at component database sites while global deadlock is resolved through timeouts. Other assumptions about the relationship between multidatabase servers and their component database systems depends on global concurrency control algorithm in use. The algorithms presented in this work guarantee that none of these assumptions would be violated unless otherwise stated.

Finally, no two subtransactions at any multidatabase server or component database system share a common ancestor transaction (or subtransaction). This assumption guarantees that the runtime composition lattice for any global transaction always takes the form of a directed acyclic graph (DAG).

4 Composite Transaction Scheduling Problem

In a composite multidatabase environment, guaranteeing serializability in each multidatabase cell is not enough for ensuring global serializability. This condition generates globally serializable schedules only if there are no global transactions that span more than one multidatabase cell. However, this case represents a degenerate composition. In this work, we focus only on non-degenerate multidatabase composition and show in our algorithms that little overhead is incurred in the degenerate case.

Let us consider the non-degenerate composition $CM = \{M_1, M_2, M_3\}$ illustrated in Figure 3. There are three multidatabase cells under the control of servers M_1, M_2, and M_3. Each server M_i having a set of component databases, C_{ij} $(j = 1, \ldots, n_i)$, under its control. T_1 and T_2 are two global transactions that begin execution at M_1 and M_3, respectively. These transactions are decomposed into subtransactions, T_{kl} $(k = 1, 2)$, that run at a component database or get delegated to run in another multidatabase cell. Delegated subtransactions are further decomposed into subtransactions T_{klm} that run at component databases in the foreign cell. Although M_1, M_2, and M_3 are all multidatabase serializable, it is possible to generate non-serializable global schedules as follows.

Let a, b, c, and d be data items stored at C_{12}. Also, let data items e and f, and data items g and h be stored at C_{13} and C_{21}, respectively. Finally, let L_1 be a local transaction running at C_{12}. The following histories are possible at C_12, C_{13}, and C_{21}:

$$
\begin{aligned}
H_{C_{12}}: &\quad r_{T_{12}}(a)\, w_{T_{12}}(b)\, r_{L_1}(c)\, w_{L_1}(a)\, c_{T_1}\, c_{L_1}\, r_{T_{261}}(d)\, w_{T_{261}}(c)\, c_{T_2} \\
H_{C_{13}}: &\quad r_{T_{14}}(e)\, c_{T_1}\, r_{T_{262}}(f)\, w_{T_{262}}(e)\, c_{T_2} \\
H_{C_{21}}: &\quad r_{T_{251}}(g)\, w_{T_{251}}(h)\, c_{T_2}\, w_{T_{131}}(g)\, c_{T_1}
\end{aligned}
$$

We ignore the histories at other component databases since single transactions run at these sites and there are no conflicts. The resulting serialization orders at the component databases follow.

$$
\begin{array}{llll}
C_{11}: T_{11} & C_{12}: T_{12} \rightsquigarrow T_{261} & C_{13}: T_{14} \rightsquigarrow T_{262} & \\
C_{21}: T_{251} \rightsquigarrow T_{131} & C_{22}: T_{252} & & \\
C_{31}: T_{21} & C_{32}: T_{22} & C_{33}: T_{23} & C_{34}: T_{24}
\end{array}
$$

At component databases the $T_i \rightsquigarrow T_j$ means that T_i is serialized before T_j, and there may be other local transactions, L_i $(i = 1, 2, \ldots, n)$, serialized between T_i and T_j such that,

$$T_i \longrightarrow L_1 \longrightarrow L_2 \longrightarrow \cdots \longrightarrow L_n \longrightarrow T_j.$$

Since each multidatabase cell is multidatabase serializable, the following serialization orders are induced on multidatabase transactions:

$$M_1: T_1 \longrightarrow T_{26} \qquad M_2: T_{25} \longrightarrow T_{13} \qquad M_3: T_2$$

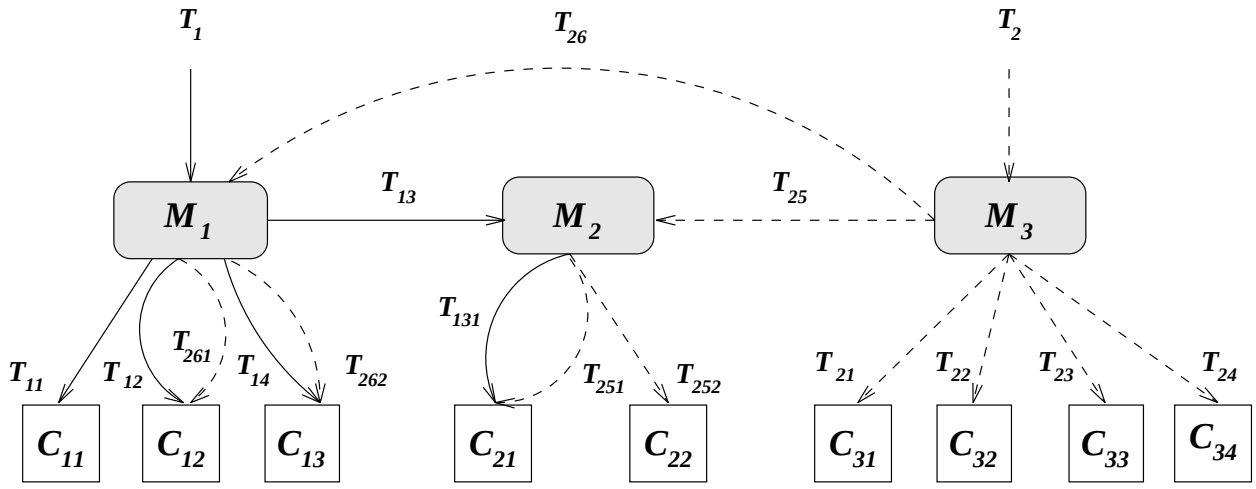

Figure 3: An illustration of the scheduling problem in a composite MDBS.

Therefore, $T_1 \rightarrow T_2 \rightarrow T_1$ is a possible global serialization order and the global committed history is not serializable.

5 Scheduling Transactions in Composite Multidatabase Systems

Ordering constraints need to be imposed on the multi-cellular transactions to guarantee globally serializable histories. In the previous section, no restrictions were imposed on the multi-cellular global transactions T_1 and T_2, in M_1 and M_2, resulting in inconsistent serialization orders at the global level. We describe three mechanisms for guaranteeing global consistent orders and prove their correctness. These mechanisms impose consistent global ordering constraints on underlying multidatabase cells, leaving their concurrency control schedulers unaffected.

5.1 Composite Rigorous Scheduling Algorithm

The algorithm described in this section extends the rigorous multidatabase scheduling algorithm [7, 13, 22] to composite multidatabase environments. Recall a component database system, C, is said to have a *rigorous* history if when two transactions T_i and T_j have operations such that $o_j(x)$ conflicts with $o_i(x)$ on a data item x in C, T_i must commit or abort before $o_j(x)$ executes. Rigorous histories are a subclass of strongly recoverable histories. Therefore, they inherit the property that the serialization order of any two transactions is equivalent to their commitment order [9]. We use this property to guarantee consistent global transaction orders across multidatabase cells.

The following theorem is an application of the rigorous scheduling algorithm for simple multi-

database systems to composite multidatabase environments.

Theorem 1 (Composite Rigorous Scheduling Algorithm (CRS)) *Let CM be a composite multidatabase system consisting of n multidatabase cells $M_1, M_2, \ldots, M_n$. If*

(i) all component databases of each M_i $(i = 1, \ldots, n)$ guarantee rigorous serializable histories, and

(ii) global atomicity at and among the M_i is guaranteed through the two-phase commit (2PC) protocol

then the global committed history of CM is serializable.

Proof

Let H_{CM} be a global committed history of $CM = \{M_1, M_2, \ldots, M_n\}$. Assume that H_{CM} is not serializable. Then, the serialization graph SG_{CM} has a cycle,

$$T_1 \xrightarrow{M_{\sigma_1}} T_2 \xrightarrow{M_{\sigma_2}} \cdots \xrightarrow{M_{\sigma_{k-1}}} T_k \xrightarrow{M_{\sigma_k}} T_1$$

where $T_i \xrightarrow{M_{\sigma_l}} T_j$ refers to the serialization order of T_i and T_j imposed by cell M_{σ_l}. It is possible that $M_{\sigma_l} \equiv M_{\sigma_m}$. If $M_{\sigma_1} \equiv M_{\sigma_1} \equiv \cdots \equiv M_{\sigma_k}$ then we have a contradiction because we assume in Section 3 that M_i $(i = 1, 2, \ldots, n)$ are all multidatabase serializable and therefore cannot contain cyclic serialization graphs. So, there must be M_{σ_l} and M_{σ_m} such that $M_{\sigma_l} \not\equiv M_{\sigma_m}$.

Since M_{σ_l} $(l = 1, 2, \ldots, k)$ are all multidatabase serializable, $T_i \xrightarrow{M_{\sigma_k}} T_j$ implies that $T_i \xrightarrow{C_{\sigma_l m}} T_j$ at every component database of $C_{\sigma_l m}$ of M_{σ_l} where T_i and T_j conflict. But all local histories at the $C_{\sigma_l m}$ are rigorous, so the commit order of T_i and T_j is equivalent to their serialization order. This means that every transaction in any cycle of SG_{CM} has committed before itself. This contradicts the atomicity property of the 2PC protocol. Therefore our assumption is false and H_{CM} is serializable. □

Let us apply the algorithm to the composition in Figure 3. All the component databases C_{ij} guarantee rigorous serializable histories and the atomicities of the global transactions T_1 and T_2 are guaranteed by the repeated application of the 2PC protocol. The proof shows that under these conditions either

$$C_{12}: T_{12} \rightsquigarrow T_{261} \qquad C_{13}: T_{14} \rightsquigarrow T_{262} \qquad C_{21}: T_{131} \rightsquigarrow T_{251}$$

or,

$$C_{12}: T_{261} \rightsquigarrow T_{12} \qquad C_{13}: T_{262} \rightsquigarrow T_{14} \qquad C_{21}: T_{251} \rightsquigarrow T_{131}.$$

Both cases lead to serializable global histories.

5.2 Composite Forced Conflicts Algorithm

The rigorous history condition, (i) in Theorem 1, has restricted applicability because some component database systems may not support rigorous serializable histories. Also, some multidatabase cells may use mechanisms other than rigorous scheduling to guarantee serializable cell histories. We relax the rigorousness condition by requiring that all multi-cellular global transactions conflict at rigorous component database in cells where they execute. This requires that all multidatabase cells have at least one component database, R, that supports rigorous serializable histories and each multi-cellular global transaction reads and updates a designated data item, or ticket at R. We outline the composite forced conflicts algorithm and prove its correctness below.

Algorithm: Composite Forced Conflict (CFC)

Let $CM = \{M_1, M_2, \ldots, M_n\}$ be a composite multidatabase environment where each M_i $(i = 1, 2, \ldots, n)$ has a rigorous component database R_i.

$\forall$ global transactions T executing at $M_i \in CM$

$SubTrasactionSet = decompose(T)$

if $globalTransaction(T)$ is a multi-cell transaction *or* $\exists\, T_j \in \{\, T_k \mid T_k \in SubTransactionSet$ *and* T_k executes at $M_j \neq M_i\}$

create subtransaction, T_{ticket}, to update the ticket at R_i

$SubTransactionSet = SubTransactionSet \bigcup \{T_{ticket}\}$

endif

$\forall\, T_j \in SubTransactionSet$

$submit(\, T_j,\, Site(T_j)\,)$

end$\forall$

end$\forall$

$\diamond$

The *Composite Forced Conflict (CFC)* algorithm assumes multidatabase serializability is guaranteed in each cell and at least one component database is rigorous. All multi-cellular global transactions are forced to read and update a *ticket* data item at a designated rigorous component database system by creating a dummy subtransaction, T_{ticket}, to perform the update and adding it to the decomposition set of the multi-cellular global transaction. The ticket is read and accessed by multi-cellular global transactions only [1]. Simple global transactions do not read the ticket. Notice

[1] Permitting other global transactions and local transactions to access the ticket does not affect the correctness of the algorithm, but does affect performance.

that the algorithm assumes that a multi-cellular global transaction can be distinguished from a simple global transaction. This assumption is reasonable since it is possible to include the address of a parent multidatabase cell server as part of the delegation protocol between two multidatabase cell servers. The following theorem guarantees the correctness of the CFC algorithm.

Corollary 1 (CFC Correctness) *Let CM be a composite multidatabase system consisting of n multidatabase cells $M_1, M_2, \ldots, M_n$. If*

(i) every multi-cellular transaction T that executes at M_i $(i = 1, 2, \ldots, n)$ is forced to conflict with all other concurrent multi-cellular transactions at a rigorous component database system, R_i, and

(ii) the atomicity of global transactions are guaranteed through the 2PC protocol

then the global committed history of CM is serializable.

Proof

The proof of the corollary follows from the proof of Theorem 1. Assume that the global committed history H_{CM} is not serializable. Then, the serialization graph SG_{CM} contains a cycle,

$$T_1 \xrightarrow{M_{\sigma_1}} T_2 \xrightarrow{M_{\sigma_2}} \cdots \xrightarrow{M_{\sigma_{k-1}}} T_k \xrightarrow{M_{\sigma_k}} T_1.$$

Since M_{σ_l} $(l = 1, 2, \ldots, k)$ are all multidatabase serializable, $T_i \xrightarrow{M_{\sigma_k}} T_j$ implies that $T_i \xrightarrow{C_{\sigma_l m}} T_j$ at every component database of $C_{\sigma_l m}$ of M_{σ_k} where T_i and T_j conflict. In particular, $T_i \xrightarrow{R_{\sigma_k}} T_j$. But all local histories at R_{σ_k} are rigorous, so the commit order of T_i and T_j is equivalent to their serialization order. This means that every transaction in any cycle of SG_{CM} has committed before itself. This contradicts the atomicity property of the 2PC protocol. Therefore, our assumption is false and H_{CM} is serializable. □

If we apply CFC to the composition $CM = \{ M_1, M_2, M_3 \}$ in Figure 3, we need to augment M_1, M_2, and M_3 with the rigorous component database systems R_1, R_2, and R_3 respectively. The dummy transactions $T_{ticket_{T_1}}$ and $T_{ticket_{T_{26}}}$ are forced to conflict at R_1, while the ticket transactions $T_{ticket_{T_{13}}}$ and $T_{ticket_{T_{25}}}$ conflict at R_2. The proof shows that either

$$R_1 : T_{ticket_{T_1}} \longrightarrow T_{ticket_{T_{26}}} \qquad C_{12} : T_{12} \rightsquigarrow T_{261} \qquad C_{13} : T_{14} \rightsquigarrow T_{262}$$
$$R_2 : T_{ticket_{T_{13}}} \longrightarrow T_{ticket_{T_{25}}} \qquad C_{21} : T_{131} \rightsquigarrow T_{251}$$

or,

$$R_1 : T_{ticket_{T_{26}}} \longrightarrow T_{ticket_{T_1}} \qquad C_{12} : T_{261} \rightsquigarrow T_{12} \qquad C_{13} : T_{262} \rightsquigarrow T_{14}$$
$$R_2 : T_{ticket_{T_{25}}} \longrightarrow T_{ticket_{T_{13}}} \qquad C_{21} : T_{251} \rightsquigarrow T_{131}.$$

Both sets of serialization orders lead to serializable schedules in the global committed history.

5.3 Composite Timestamp Ordering Algorithm

The *Composite Timestamp Ordering (CTO)* imposes an *a priori* total order on all multi-cellular transactions. This algorithm does not have any special requirements of component databases as in the CRS and CFC algorithms, and may be used as an alternative in compositions where some multidatabase cells have no component databases that enforce rigorous histories.

The CTO algorithm assigns and validates timestamp orders as follows. All delegated subtransactions are tagged as multi-cellular and inherit the global timestamp from their parent transaction. *Untagged* transactions that enter a multidatabase cell are new transactions submitted directly from an application. New transactions are first decomposed. If any of the resulting subtransactions are delegated to other multidatabase cells, then both they and their parent are tagged as multi-cellular and assigned the same unique timestamp. *Tagged* transactions are delegated from other multidatabase cells. These transactions are first validated by checking that their timestamp is not older than that of the last committed transaction, T_{last}. If it is older, it is aborted. Otherwise, it is decomposed. All resulting subtransactions inherit the tags and the unique timestamp from the parent and are submitted to component databases for execution.

The CTO algorithm ensures that new and preparing delegated subtransactions execute and commit in timestamp order. The the global cocurrency control scheduler of each cell maintains a *multi-cellular commit order list (MCOL)*. The MCOL is never empty because it always maintains T_{last} as a node. Every multi-cellular transaction is added to the MCOL sorted on their timestamp value and placed in a *running* state. When any subtransaction on the list is ready to prepare, it first checks if it is older than T_{last}. If it is older it is aborted. Otherwise, it checks if any younger transactions are prepared. If so, it is placed in a *waiting* state and rechecks after a set interval, or after a prepared transaction is forced to abort through the 2PC protocol. Once a transaction passes the validation step, it prepares, enters a *prepare* state, and waits for a commit signal from its 2PC coordinator. On receiving a commit, a prepared transaction aborts all older transactions in the *running* and *waiting* states then commits. If it receives an abort signal it is removed from MCOL and signals all older transactions in the *waiting* state. The algorithm is outlined below.

Algorithm: Composite Timestamp Ordering (CTO)

START:

Let M_i be a multidatabase cell in a composition, $CM = \{ M_1, M_2, \ldots, M_n \}$.

NEW:

$\forall$ transactions, T_i, starting at M_i:

$$SubTransactionSet_{T_i} = Decompose(T_i)$$

if $Untagged(T_i)$ *and* $\exists\ T_{ik} \in \{\ T_{ij} \mid T_{ij} \in SubTransactionSet_{T_i}$
and T_k executes at $M_j \neq M_i\ \}$

$SetTag(T_i)$

$SetTimeStamp(T_i, NewTimeStamp())$

endif

if $Tagged(T_i)$

if $TimeStamp(T_i) < TimeStamp(T_{last})$

abort T_i

$Exit()$

endif

$Insert(T_i,\ MCOL)$

$SetState(T_i,\ RUNNING)$

$\forall\ T_{ik} \in SubTransactionSet_{T_i}$

$SetTag(T_{ik})$

$SetTimeStamp(T_{ik},\ TimeStamp(T_i))$

$Submit(T_{ik},\ Site(T_{ik}))$

end$\forall$

endif

end$\forall$

PREPARE:

if $TimeStamp(T_i) < TimeStamp(T_{last})$

abort T_i

$Exit()$

endif

if $\exists\ T_k \in \{\ T_j \mid T_j \in MCOL$ *and* $TimeStamp(T_j) > TimeStamp(T_i)$ *and* $Prepared(T_j)\ \}$

$SetState(T_i,\ WAITING)$

else

prepare T_i with 2PC coordinator

$SetState(T_i,\ PREPARED)$

endif

COMMIT:

$\forall\ T_k \in \{\ T_j \mid T_j \in MCOL$ *and* $TimeStamp(T_j) < TimeStamp(T_i)$ *and*
$Running(T_j)$ *or* $Waiting(T_j)\ \}$

abort T_k

end$\forall$

commit T_i and all subtransactions

ABORT:

$Delete(T_i,\ MCOL)$

abort T_i and all subtransactions

$\diamond$

In CTO, we assume that all timestamps within a composition are globally unique and are related in a total order. Timestamps with these properties can be generated by concatenating the global time (from some global distributed time service) with a unique site-dependent identifier at each site. Given that the timestamps are a total order the following theorem guarantees the correctness of CTO.

Theorem 2 (CTO Correctness) *For a composition* $CM = \{M_1, M_2, \ldots, M_n\}$ *if*

(i) each M_i $(i = 1, 2, \ldots, n)$ *executes the CTO algorithm, and*

(ii) global atomicity at and among the M_i *is guaranteed through the two-phase commit (2PC) protocol*

then the global committed history of CM *is serializable.*

Proof

The inheritance of timestamps by subtransactions from parent transactions coupled with the validations in the **NEW**, **PREPARE** and **COMMIT** steps of CTO guarantee that multi-cellular transactions are serialized and committed in timestamp order. The rest of this proof follows the proof by contradiction of TO in [4]. If we assume the global committed history H_{CM} is not serializable, then its serialization graph SG_{CM} has a cycle. Therefore every transaction T_i in the cycle has $TimeStamp(T_i) > TimeStamp(T_i)$, since the serialization order corresponds to the timestamp order. But this contradicts the total ordering of timestamps. Thus our assumption is false and the global committed history H_{CM} is serializable. $\square$

In applying CTO to the example in Figure 3, if $TimeStamp(T_1) > TimeStamp(T_2)$ then

$$C_{12}:\ T_{12} \rightsquigarrow T_{261} \qquad C_{13}:\ T_{14} \rightsquigarrow T_{262} \qquad C_{21}:\ T_{131} \rightsquigarrow T_{251}$$

otherwise, if $TimeStamp(T_2) < TimeStamp(T_1)$ then

$$C_{12}: T_{261} \rightsquigarrow T_{12} \qquad C_{13}: T_{262} \rightsquigarrow T_{14} \qquad C_{21}: T_{251} \rightsquigarrow T_{131}.$$

Note $TimeStamp(T_i)$ represents only the timestamp of the commited transaction. Both cases lead to serializable global histories.

6 Implementation Issues

Our investigation in transaction management for multidatabase composition falls under the CORDS Multidatabase project. We have built a multidatabase prototype that provides transparent transactional access to DB2/6000[2] and Oracle[3] V7 component database systems running under AIX[4] 3.2.x a set of RS/6000s. Although we use DB2/6000 and Oracle V7 in particular, nothing restricts us from using other X/Open[5] XA compliant databases. For details on the CORDS Multidatabase architecture and prototypes, see [10, 12].

Multidatabase cells are accessed through multidatabase servers that export an ODBC[6] interface to applications and other multidatabase servers. Global distributed transaction processing functionality is provided by the Encina[7] transaction toolkit facility. The transaction and resource access interfaces at component databases are exactly the same as those exported by multidatabase servers. Each component database system has set of agents that run as applications to provide the necessary interface functionality. Both the agents and the multidatabase cell servers run as Encina TP monitor applications. The monitor environment tightly integrates Encina distributed transaction management, 2PC through the X/Open XA standard [11], distributed deployment of the system, and fault tolerance. The agents and cell servers are all multi-threaded and communicate via OSF[8] DCE remote procedure calls (RPC). Details on the transaction management architecture are described in [2].

Currently, we support only the composite rigorous scheduling algorithm. Work on adding the CFC and CTO algorithms to the the GCC schedulers of our multidatabase cell servers will soon be completed. Initial tests validate both the correctness and viability of the algorithms. These tests also validate the following performance issues.

[2] DB2/6000 is a trademark of the IBM Corporation.

[3] Oracle is a trademark of the Oracle Corporation.

[4] AIX is a trademark of the IBM Corporation.

[5] X/Open is a trademark of the X/OPEN company.

[6] Open Database Connect from the Microsoft Corporation.

[7] Encina is a trademark of Transarc Corporation.

[8] OSF is a trademark of the Open Software Foundation.

Dynamic Composition and Scalability

In our prototype, multidatabase composition is dynamic and depends on collective runtime access patterns of all global transactions. This is unlike the static hierarchies required by Schek and Weikum in [24], and by Pu in [18]. This feature allows the dynamic removal and addition of multidatabase cell servers from a composite environment. Transaction access patterns are also dynamically controlled at runtime by manipulating catalogue data at active multidatabase cell servers.

Dynamic composition facilitates scalability. Our concurrency control algorithms do not limit the number of multidatabase cells that participate in a composition nor the number of active transactions. However, the scale of a composite multidatabase environment may be limited by the latency 2PC protocol and the timeout interval at component database systems.

Message Complexity

The message complexity of our algorithms is equivalent to that of the 2PC protocol. In each of our algorithms, control information is *piggy-backed* as transaction properties on 2PC protocol messages. Our algorithms do not suffer from the communication limitations of explicitly communicating global ordering data structures like O-vectors in [16, 18][9].

Global Aborts

Each of the algorithms guarantees serializable global transaction orders through the preemption of non-serializable transactions by global aborts. The CRS and CFC algorithms work by creating deadlock among non-serializable multi-cellular transactions. The deadlock is resolved by forcing a transaction in the deadlock cycle to rollback through timeouts at component databases, timeouts in the 2PC coordinator, or deadlock resolution algorithms at component database systems. The CTO algorithm maintains the total ordering of the timestamps of committed transactions by explicitly preempting any multi-cellular transaction whose commitment will violate the total order. Global aborts degrade system performance by reducing both global and local transaction throughput and wasting valuable system resources. We expect abort frequencies to increase significantly as the locality of multi-cellular transactions increases and their read/write ratios decrease. We believe that a smart recovery/resubmission algorithm can avert these effects. However, the design of the recovery algorithm may be limited by the underlying properties of the underlying multidatabase concurrency control mechanism.

[9] Pu cited this as a major reason maintaining a static hierarchy of databases.

7 Conclusion

The significance of issues such as location transparency, scalability, performance, and administration make multidatabase composition inevitable. We have shown that scheduling transactions that span multiple multidatabase cells is not simply a matter of guaranteeing transaction atomicity and serializability at each multidatabase cell. To guarantee global serializability in a composite multidatabase environment, ordering constraints must be imposed on multi-cellular transactions. We presented and proved the correctness three multi-cellular transaction scheduling algorithms: *Composite Rigorous Scheduling (CRS)*, *Composite Forced Conflicts (CFC)*, and *Composite Timestamp Ordering (CTO)*. CRS and CFC enforce ordering constraints implicitly through rigorous histories at component database. While CTO does so explicitly through a global total order on timestamps.

Our algorithms scalable and support dynamic multidatabase composition. They incur no communications overhead since all control information is superimposed on 2PC messages. However, we expect abort rates to increase as multi-cellular transaction locality increases and read/write ratios decrease. This implies that composite multidatabase recovery must be smart and efficient.

References

[1] R. Alonso, H. Garcia-Molina, and K. Salem. Concurrency Control and Recovery for Global Procedures in Federated Database Systems. In *Proceedings of the IEEE Conference on Data Engineering*, pages 5–11. IEEE Computer Society Press, September 1987.

[2] G. Attaluri and D. P. Bradshaw. Architecture for Transaction Management in the CORDS Multidatabase System. In A. Gawman, W. Morven Gentleman, E. Kidd, P.-Å. Larson, and J. Slonim, editors, *"Proceedings of CASCON'93 Volume II: Distributed Computing"*, pages 873–887, Toronto, Canada, October 1993. IBM Toronto Labs, IBM Centre for Advanced Studies. To appear in Proceedings of the 1993 CAS Conference.

[3] C. Beeri, P. A. Bernstein, and N. Goodman. A Model for Concurrency Control in Nested Transaction Systems. *JACM*, 36:230–269, 1989.

[4] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Series in Computer Science. Addison-Wesley, United States of America, 1987.

[5] D. P. Bradshaw. Open Nested Serializability in Multidatabase Systems. In M. Bauer, J. Botsford, P.-Å. Larson, and J. Slonim, editors, *Proceedings of the 1992 CAS Conference*, pages

93–109, Toronto, Canada, November 1992. IBM Toronto Labs, IBM Centre for Advanced Studies.

[6] Y. Breitbart and H. Garcia-Molina. Overview of Multidatabase Transaction Management. Technical Report TR-92-21, Department of Computer Services, University of Texas, Austin, Texas 78712, May 1992.

[7] Y. Breitbart, D. Georgakopoulos, M. Rusinkiewicz, and A. Silberschatz. On Rigorous Transaction Scheduling. *IEEE Transactions on Software Engineering*, 17(4):954–960, September 1991.

[8] Y. Breitbart and A. Silberschatz. Multidatabase Update Issues. In *Proceedings of the ACM SIGMOD Conference on the Management of Data*, pages 135–142. ACM, ACM Press, June 1988.

[9] Y. Breitbart and A. Silberschatz. Strong Recoverability in Multidatabase Systems. In P. S. Yu, editor, *Second International Workshop on Research Issues on Data Engineering: Transaction and Query Processing*, pages 170–175, Los Alamitos, Ca, February 1992. IEEE Computer Society Technical Committee on Data Engineering, IEEE Computer Society Press.

[10] N. Coburn and P.-Å. Larson. Multidatabase Services: Issues and Architectural Design. In M. Bauer, J. Botsford, P.-Å. Larson, and J. Slonim, editors, *Proceedings of the 1992 CAS Conference*, pages 57–66, Toronto, Canada, November 1992. IBM Toronto Labs, IBM Centre for Advanced Studies.

[11] X/Open Company. *CAE Specification. Distributed Transaction Processing: The XA Specification.* X/Open Company Limited, United Kingdom, 1991.

[12] D. L. Erickson, P. J. Finnigan, G. K. Attaluri, M. A. Bauer, D. P. Bradshaw, N. Coburn, M. P. Consens, M. Z. Hasan, J. W. Hong, K. A. Lyons, T. P. Martin, G. W. Neufeld, W. Powley, D. Rappaport, D. J. Taylor, T. J. Teorey, and Y. Yemini. CORDS: An Update to Prototypes. Technical Report TR-74.120, Centre for Advanced Studies, IBM Toronto Labs, August 1993.

[13] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. On Serializability of Multidatabase Transactions Through Forced Local Conflicts. In *Proceedings of the Seventh International Conference on Data Engineering*, pages 314–323, Los Alamitos, Ca, April 1991. IEEE, IEEE Computer Society Press.

[14] J. N. Gray. Notes on Database Operating Systems. In R. Bayer, R. M. Graham, and G. Seegmuller, editors, *Operating Systems: An Advanced Course*, pages 393–481. Springer-Verlag, Berlin, Germany, 1978.

[15] J. N. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, San Mateo, Ca, 1993.

[16] A. Leff and C. Pu. Classification of Transaction Processing Systems. *IEEE Computer*, 24(6):63–76, June 1991.

[17] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz. Ensuring Transaction Atomicity in Multidatabase Systems. Technical Report TR-92-12, Department of Computer Sciences, University of Texas at Austin, Austin, Texas, 78712, June 1992.

[18] C. Pu. Superdatabases for Composition of Heterogeneous Databases . In *Proceedings of the Fourth International Conference on Data Engineering*, pages 548–555, Los Alamitos, Ca, May 1988. IEEE, IEEE Computer Society Press.

[19] Y. Raz. Extended Commitment Ordering. Technical Report DEC-TR 842, Digital Equipment Corporation, 151 Taylor St, Littleton, Ma 01460, December 1991.

[20] Y. Raz. Guaranteeing global serializability via Commitment Ordering. Technical Report DEC-TR 843, Digital Equipment Corporation, 151 Taylor St, Littleton, Ma 01460, December 1991.

[21] Y. Raz. Principle of Commitment Ordering. Technical Report DEC-TR 841, Digital Equipment Corporation, 151 Taylor St, Littleton, Ma 01460, November 1991.

[22] Y. Raz. Locking Based Strict Commitment Ordering. Technical Report DEC-TR 844, Digital Equipment Corporation, 151 Taylor St, Littleton, Ma 01460, February 1992.

[23] K. Salem, H. Garcia-Molina, and R. Alonso. Altruistic Locking: A Startegy for Coping with Long-Lived Transactions. In D. Gawlick, M. Haynie, and A. Reuter, editors, *Lecture Notes in Computer Sciences, High Performance Transaction Processing Systems*, volume 359, pages 175–199. Springer-Verlag, 1989.

[24] H-J. Schek, G. Weikum, and W. Schaad. A Multi-Level Transaction Approach to Federated DBMS Transaction Management. In *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems*, pages 280–287. IEEE Computer Society Press, April 1991.