# Multidatabase Update Issues

*Yuri Breitbart*

Department of Computer Science
University of Kentucky
Lexington, KY 40506

*Avi Silberschatz*†

Department of Computer Science
University of Texas
Austin, Texas 78712

*ABSTRACT*

A formal model of data updates in a multidatabase environment is developed, and a theory of concurrency control in such an environment is presented. We formulate a correctness condition for the concurrency control mechanism and propose a protocol that allows concurrent execution of a set of global transactions in presence of local ones. This protocol ensures the consistency of the multidatabase and deadlock freedom. We use the developed theory to prove the protocol's correctness and discuss complexity issues of implementing the proposed protocol.

## 1. Introduction

*A Multidatabase System* (MDBS) is a collection of several databases, each of which is controlled by a database management systems (DBMS). An MDBS creates the illusion of a single database system. It allows users to manipulate data contained in the various databases without modifying current database applications and without migrating the data to a new database. The MDBS hides from users the intricacies of different DBMS's and different access methods. It provides uniform access to pre-existing databases without requiring the users to know either the location or the characteristics of different databases and their corresponding DBMS's. The MDBS query and data manipulation languages allows users to access multiple pre-existing databases in a single query or application.

† This research was partially supported by ONR Contract N00014-86-K-0161 and NSF Research Grant DCR-8507224



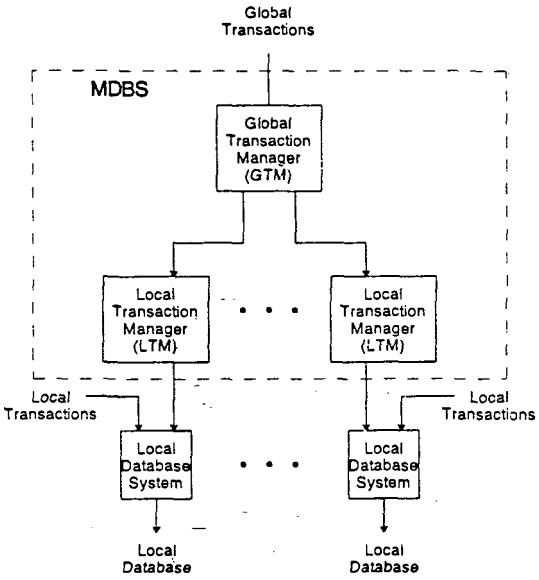


Figure 1. Multidatabase Model.

An MDBS consists of global and local components, as depicted in Figure 1. Interactions with a global(local) database are conducted by user programs called global(local) *transactions*. A transaction is simply a sequence of read and write operations defined on a database. A global transaction is a transaction that is submitted to the MDBS and is executed under the MDBS control. A local transaction, on the other hand, is a transaction submitted to a local DBMS, outside of the MDBS control.

The global transaction manager (GTM) controls the execution of global transactions. For each global transaction, the GTM allocates one local transaction manager (LTM) for each of the sites referenced by the transaction. Each LTM is responsible for translating global read and write operations into the languages of the local DBMS and transferring retrieved data to the GTM. Once an LTM is allocated it is not deallocated until the transaction either commits or aborts.

The MDBS model discussed in this paper is based on the following two principles:

a) No changes can be made in the local DBMS software in order to accommodate the MDBS. Modifying a local DBMS software creates significant development and maintenance problems. It is no longer possible to use a DBMS's vendor software without modifications and this, in turn, makes it infeasible from an economic standpoint to install new software releases from the vendor.

b) Each DBMS retains local autonomy of the database it controls. The concept of local autonomy requires that the local DBMS and local database administrator perform as if the MDBS does not exist. It means that local DBMS operation, maintenance, and performance tuning should proceed in the same way as without the MDBS.

We assume that the local DBMS's are not aware of each other, and if a local transaction is submitted to a local DBMS then no other local site is aware of that transaction.

In order to ensure the correct behavior of the system, the MDBS must be able to synchronize the execution of global transactions with local ones. This is generally not possible to achieve if arbitrary local transactions can be submitted at local sites, since a local transaction may change a value of a replicated data item. To guard against such behavior the MDBS must provide a concurrency control scheme and formulate restrictions on the type of local transactions that can be tolerated by the MDBS concurrency control mechanism.

The purpose of this paper is to investigate concurrency control issues that arise when global and local transactions are processed concurrently in a multidatabase environment. Besides being of theoretical interest, concurrency control in a multidatabase environment is an important practical problem. A solution to this problem would allow semantically related data in pre-existing database applications to be maintained in a consistent state, thereby avoiding an expensive redesign.

## 2. Previous Work

The problems associated with concurrent access in multidatabase systems have been considered by a number of different researchers, ([BREI85], [MOTR81], [HAMM80], [FERR83], [LAND82], [TEMP85], [LITW82]) to name just a few. These researchers considered the case where updates to the database system can only be done by local transactions. Global transactions were restricted to retrieve-only access. One assumption frequently made is that an MDBS with retrieve-only global transactions does not require a concurrency control mechanism [LAND82]. This assumption is based on the premise that the probability of inconsistent retrieval in the presence of local transactions is quite low and therefore it is not necessary to synchronize execution of local and global transactions. However we will show that this may result in users retrieving inconsistent data.

The update problem in a multidatabase environment was recently discussed in [GLIG85]. The authors identified several issues that make an update problem difficult. Among them are:

1. How to generate and control execution of global transactions at local sites.
2. How to maintain global transaction atomicity at local sites.
3. How to detect and recover from, or how to prevent, global deadlocks.

While the authors presented extensive discussion concerning the update problem, they did not propose either a concurrency control scheme or a recovery algorithm that would solve the identified problems.

In ([PU86], [PU87], [ELMA87]) the update problem is addressed under the assumption that the MDBS is aware of local transactions. Making the MDBS aware of local transactions requires changes to the local concurrency control mechanisms to enable the local DBMSs to report local transaction execution information to the MDBS, which uses this information for local and global transactions synchronization. Introducing such changes allows any two DBMSs to communicate with each other and, therefore, reduces the multidatabase concurrency control problem to the concurrency control problem in distributed database management systems [BERN81].

The update problems in a homogeneous and a multidatabase environment, however, are entirely different. The major difference is that in the distributed case, as soon as a new transaction enters the system at a local site, every other site becomes aware of this new transaction. As a result, a concurrency control mechanism at each site is able to synchronize the transaction execution with other transactions executed at that site. In the multidatabase environment, however, the local DBMSs should perform their operations without the knowledge of other DBMSs or of the MDBS's existence.

In [ALON87] the update problem is considered under the assumption that the MDBS is not aware of local transactions and local DBMSs are not aware of each other. They present an 'altruistic locking' scheme that increases concurrency by allowing an early release of locks subject to some constraints that preserve serializability. The recoverability of global transactions is also considered, including compensating for local actions that have already committed. The authors defined a concept of 'sagas' for further increasing concurrency when global serializability and consistency constraints spanning multiple pre-existing databases do not need to be preserved. They do not, however, allow local transactions at the local sites.

In this paper we provide a theoretical basis for the approach outlined in [BREI87a]. We propose a formal model of data updates in a multidatabase environment and develop a theory of serializability in such an environment in the presence of local transactions. We show that a characterization of the correctness of a concurrency control schema in a distributed database environment [PAPA79] is not sufficient in a multidatabase environment. A correctness condition for the concurrency control mechanism in a multidatabase environment is formulated, including a characterization of local transactions that do not destroy multidatabase consistency. We discuss problems of global deadlocks. We propose a transaction protocol that allows concurrent execution of a set of global transactions in the presence of local ones and that ensures multidatabase consistency and the absence of global deadlocks. The developed theory is used to prove the protocol's correctness. Lastly, we discuss complexity issues of implementing the proposed protocol.

The remainder of this paper is organized as follows. Section 3 contains a detailed description of a formal multidatabase model and a definition of serializable execution of a combination of local and global transactions in this model. In section 4 we prove the serializability condition for a set of global and local transactions. In section 5 we present a transaction protocol to ensure global database consistency for a set of global and local transactions, and prove its correctness. Section 6 discusses a problem of global deadlocks in a multidatabase environment and proves that the transaction protocol proposed in Section 5 ensures an absence of global deadlocks. In section 7 we prove that the problem of selecting sites to execute a global transaction's read/write operations in the framework of the proposed protocol is NP-complete.

## 3. Multidatabase Update Model

The multidatabase system discussed in this paper meets the following criteria:

1. The multidatabase concurrency control mechanism guarantees a serializable global transaction execution.
2. The local concurrency control mechanism(s) guarantees a serializable local transaction execution.
3. No modifications are allowed to a local DBMS's software in order to deal with a multidatabase system.
4. No direct communication exists between local DBMSs.

In this section we describe the global and local levels of this multidatabase system and the interrelationship between them.

*A Global Database* is a triple $<D, S, f>$, where $D$ is a set of global data items, $S$ is a set of sites, and $f$ is a function:

$$f : D \times S \rightarrow (0,1).$$

We will interpret $f(x,i) = 1$ as the availability of the data item $x$ at site $i$. A data item $x$ is replicated if it is located at more than one site; that is, there are sites $i$ and $j$ such that $i \neq j$ and $f(x,i) = f(x,j) = 1$.

We define a serializable global(local) schedule in the usual manner (see [BERN81]) and serializability is used as a correctness criteria for a concurrency control mechanism of a MDBS. We assume that MDBSs, as well as local DBMSs, always produce serializable schedules.

*A global(local) serialization graph* of a global(local) schedule $S$ is a directed graph whose nodes are global(local) transactions and $G_i \rightarrow G_j$ if there are conflicting operations $op_i$ and $op_j$ (i.e., both operate on the same data item and one of them is a write operation) in $G_i$ and $G_j$, respectively, such that $op_i$ precedes $op_j$ in $S$. It is well known that if a serialization graph of a schedule $S$ is acyclic, then $S$ is serializable (see [BERN85]).

To formalize the relationship between local and global operations, we define a translation operator $T$ which maps each global operations as follows:

1. A global read operation is mapped into some local operation $r(x_i)$ for some site $i$ that contains a copy of $x$, (i.e., $f(x,i) = 1)$).
2. Each global write operation is mapped into a set of local operations $w(x_{i1})$, $w(x_{i2})$, ... $w(x_{il})$, where $i1, i2, \ldots il$ are the only sites that contain a copy of $x$.

It should be noted that a translation operator $T$ does not impose any order on the execution of local writes required to execute a global write operation on $x$. Applying a translation operator $T$ to a global transaction, we obtain an execution sequence of local operations whose execution is equivalent to an execution of the global transaction.

Let $G$ be a set of global transactions. *A multidatabase schedule (or md-schedule)* $S$ over $G$ is a sequence of local operations where:

1. The set of local operations is obtained from $G$ by applying a translation operator to each atomic operation of every transaction from $G$.
2. For each global transaction $G_i$, if $op_i$ precedes $op_i'$, in $G_i$, then every operation in $T(op_i)$ precedes every operation in $T(op_i')$ in $S$.

Let $S$ be an arbitrary md-schedule over a set of global transactions $G$. Transaction $G_j$ *reads-x-from* $G_i$ in $S$ if for some local data item $x_m$ the following holds:

1. $w_i(x_m)$ and $r_j(x_m)$ are operations in $S$.
2. $w_i(x_m)$ precedes $r_j(x_m)$ in $S$.
3. No $w_k(x_m)$ falls between $w_i(x_m)$ and $r_j(x_m)$ is $S$.

Two md-schedules are equivalent if they have the same read-$x$-from relation for every data item $x$. A md-schedule is serializable if it is equivalent to some serial md-schedule.

**Lemma 1:** Any serial md-schedule is equivalent to some serial global schedule. □

**Theorem 1:** Any serializable md-schedule is equivalent to some serial global schedule. □

The serialization graph of a md-schedule $S$, is a directed graph whose nodes represent global transactions and $G_i \rightarrow G_j$ if there are conflicting operations $op_i$ and $op_j$ operating on the same copy of a data item in $G_i$ and $G_j$, respectively such that $op_i$ precedes $op_j$ in $S$. It was shown in [BERN85] that if the serialization graph of a md-schedule is acyclic, then the schedule is serializable.

The problem of deciding whether there is a site selection for read operations that makes an md-schedule serializable is NP-complete. Indeed, it was proven in [PAPA79] that even the problem of deciding whether an md-schedule is serializable is NP-complete. Therefore, it is important to find a class of md-schedules that are serializable for any selection of reading sites.

**Theorem 2:** Let $S$ be a global schedule. Any md-schedule $T(S)$ obtained by applying a translation operator $T$ to each global operation of $S$ is equivalent to $T(S)$. □

Given a set of global transactions in a multidatabase system and in the absence of any other transactions at local sites, the correct execution of these transactions is equivalent to the existence of a serializable md-schedule. Hence, to execute a global transaction is equivalent to an execution of a set of local transactions that are generated using a translation operator $T$.

## 4. Global Consistency Conditions

Let us first extend a definition of global database consistency for the case when arbitrary local transactions are allowed at local sites.

Let $G$ be a set of global transactions and let $L = \{L_1, ..., L_m\}$ be sets of local transactions at various local sites. We define a function $T^{-1}$ on local operations of transaction $L_i$ as follows:

$$T^{-1}(r_j(x_i)) = r_j(x)$$
$$T^{-1}(w_j(x_i)) = w_j(x).$$

By applying $T^{-1}$ to each atomic operation of each local transaction from $L$ we obtain a system $G'$ of global transactions. We say that $G$ retains global database consistency if and only if there is a serializable global schedule $S$ for $G \cup G'$ such that a result of the execution of $S$ is computationally equivalent to the result of the execution of $G$ by the MDBS and $L$ by their respective DBMSs.

Papadimitriou [PAPA79] has shown that in order to generate a serializable md-schedule it is both necessary and sufficient that at each site, for each pair of global conflicting operations $op_i$ and $op_j$, $op_i$ should precede $op_j$ in any local schedule. The criteria is not sufficient if local transactions are allowed in the MDBS environment.

Example 1

Consider a multidatabase system where data items $a$ and $b$ are at site 1 and $c$ and $d$ are at site 2. The following global transactions are submitted to the MDBS:

$$G_1: r_1(a)\, r_1(d)$$
$$G_2: r_2(b)\, r_2(c).$$

Let $L_1$ and $L_2$ be two local transactions that are submitted at sites and 1 and 2, respectively:

$$L_1: r_3(a_1)\, r_3(b_1)\, w_3(a_1)\, w_3(b_1)$$
$$L_2: r_4(c_2)\, r_4(d_2)\, w_4(c_2)\, w_4(d_2).$$

There are no conflicting atomic operations between global transactions $G_1$ and $G_2$, therefore any md-schedule is serializable and should consequently retain global database consistency.

On the other hand, by applying $T^{-1}$ to $L_1$ and $L_2$, respectively, we obtain global transactions

$$G_3: r_3(a)\, r_3(b)\, w_3(a)\, w_3(b)$$
$$G_4: r_4(c)\, r_4(d)\, w_4(c)\, w_4(d).$$

These two transactions contain atomic operations that are in conflict with some atomic operations of transactions $G_1$ and $G_2$.

Let $S1$ and $S2$ be local schedules at sites 1 and 2, respectively:

$$S1: r_1(a_1)\, r_3(a_1)\, r_3(b_1)\, w_3(a_1)\, w_3(b_1)\, r_2(b_1)$$
$$S2: r_2(c_2)\, r_4(c_2)\, r_4(d_2)\, w_4(c_2)\, w_4(d_2)\, r_1(d_2).$$

The result of these executions at local sites is not equivalent to any serial execution of $G_1$, $G_2$, $T^{-1}(L_1)$, and $T^{-1}(L_2)$. Schedules $S1$ and $S2$, however, do satisfy Papadimitriou's criteria since there are no conflict operations in $G_1$ and $G_2$. □

This example emphasizes the necessity of a concurrency control mechanism even in an environment where global transactions have no write operations. The next theorem formulates sufficient criteria for global database consistency.

**Theorem 3:** Let $G$ and $L$ be a set of global and local transactions, respectively. Global database consistency is assured if the following two conditions hold:

1. Any local transaction from $L$ is a read-only or can write only into non-replicated data items.
2. There exists a total ordering of transactions from $G$ such that if $G_i$ precedes $G_j$ in this ordering, then for every pair of atomic operations $op_i$ and $op_j$ from $G_i$ and $G_j$ respectively, $op_i$ precedes $op_j$ in each local schedule. □

If local transactions from $L$ do not contain any local atomic operations that are in conflict with local atomic operations from global transactions, then condition 2 of Theorem 3 is not necessary, as the next example demonstrates.

Example 2

Consider a multidatabase system where data items $a$ and $b$ are at site 1 and $c$ and $d$ are at site 2. The following global transactions are submitted to the MDBS:

$$G_1: r_1(a)\, r_1(c)\, w_1(a)$$
$$G_2: r_2(b)\, r_2(d)\, w_2(d).$$

In addition, let $L_1$ and $L_2$ be two local transactions that are submitted at sites 1 and 2, respectively:

$$L_1: r_3(b)$$
$$L_2: r_4(c).$$

There are no conflicting operations between transactions generated from $G_1$ and $G_2$ by the MDBS at sites 1 and 2 and local transactions $L_1$ and $L_2$. This means that at local sites any schedule will be serializable and will also retain global database consistency. Thus, a transaction's execution order at each site need not be retained. Therefore, condition 2 of Theorem 3 is violated. □

## 5. Global Database Consistency Protocol

Theorem 3 states sufficient conditions to ensure the global database consistency of a multidatabase system. The second condition of the theorem states that global transactions should be executed in the same order in each local schedule. Unfortunately, there is no simple way to ensure that this condition holds in each local schedule. There is, however, another approach to ensure global database consistency. This approach would require the MDBS to impose some restrictions on the order of submitting global transactions for execution in such a way that, even in presence of arbitrary local read-only transactions or local write transactions, no possibility of global database inconsistency may occur. In this section we present one such approach and prove its correctness.

Let us first introduce the notion of a *site graph* for a global transaction $G_i$ [BREI87a]. The nodes of a site graph are the names of the sites that contain copies of the global data items referenced in $G_i$. These nodes are connected by undirected edges, labeled with the transaction name $G_i$, in any order that forms an acyclic graph. Obviously, a site graph of a global transaction $G_i$ is not unique.

Given a set of global transactions $G$, and taking the union of site graphs for each global transaction, we obtain a site graph of the system of global transactions $G$. To illustrate this, consider the following example.

Example 3

Consider a global database that contains data item $a$ at sites 1 and 2, $b$ at sites 1 and 3, and $c$ at sites 2 and 3. Let $G_1$ and $G_2$ be the following global transactions:

$$G_1: r_1(a)\, w_1(b)$$
$$G_2: r_2(b)\, w_2(c).$$

The GTM may generate the following sequence of local operations for each transaction:

$$T(G_1): r_1(a_1)\, w_1(b_1)\, w_1(b_3)$$
$$T(G_2): r_2(b_3)\, w_2(c_2)\, w_2(c_3).$$

The site graph of $G_1$ and $G_2$ for this site selection is shown in Figure 2a.

The GTM could also generate the following system of local operations for each transaction:

$$T(G_1): r_1(a_1)\, w_1(b_1)\, w_1(b_3)$$
$$T(G_2): r_2(b_1)\, w_2(c_2)\, w_2(c_3).$$

The site graph of $G_1$ and $G_2$ for the latter application of $T$ is shown in Figure 2b and it does contain a cycle.

The existence of a cycle in the site graph of a system of global transactions may cause global database inconsistency during the execution of read and write operations of global and local transactions that are either read-only transactions or local write transactions. On the other hand, the absence of cycles in the site graph will guarantee the correct execution of any mix of global transactions and local transactions described in Theorem 3, as the next theorem indicates.

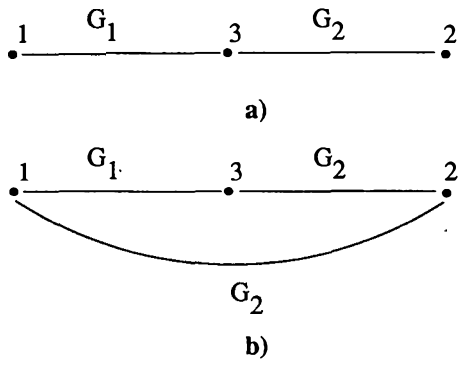


Figure 2. Site Graph

**Theorem 4:** Let $G$ and $L$ be a set of global and local transactions, respectively. The consistency of a global database is assured if the following conditions hold:

1. Any local transaction from $L$ is a read-only transaction or can write only into non-replicated data items.
2. There exists at least one application of operator $T$ to each transaction from $G$ such that the corresponding site graph for $G$ is acyclic. □

## 6. Multidatabase Global Deadlock

If a local DBMS uses some sort of locking mechanism to insure serializability, then the potential problem of a global deadlock can arise. We will assume that any local schedule is deadlock-free, meaning that any local wait-for graph [see KORT86] does not contain a cycle. This, however, does not assure that global deadlock will not occur.

Example 4

Consider a multidatabase system where data items $a$ and $b$ are at site 1 and $c$ and $d$ are at site 2. The following global transactions are submitted to the MDBS:

$$G_1: r_1(a)\, w_1(a)\, r_1(c)\, w_1(c)$$
$$G_2: r_2(b)\, w_2(b)\, r_2(d)\, w_2(d).$$

Let $L_1$ and $L_2$ be two local transactions that are submitted at sites 1 and 2, respectively:

$$L_1: r_3(a)\, r_3(b)$$
$$L_2: r_4(c)\, r_4(d).$$

Let $S1$ and $S2$ be local schedules at sites 1 and 2, respectively.

$$S1: r_3(a)\, r_1(a)\, w_1(a)\, r_2(b)\, w_2(b)\, r_3(b)$$
$$S2: r_4(c)\, r_1(c)\, w_1(c)\, r_2(d)\, w_2(d)\, r_4(d).$$

Both schedules are serializable and their combined execution retains a global database consistency by virtue of Theorem 3. A global deadlock, however, may occur during the execution of $G_1$ and $G_2$ as follows.

At site 1, $G_1$ waits for the data item $a$ that is locked by $L_1$. $L_1$ waits for the data item $b$ that is locked by $G_2$. $G_2$, in turn, is waiting for a message from the MDBS that it may proceed. The MDBS, however, is waiting to receive a message from $G_1$ at site 2 that is has completed, since the MDBS is trying to synchronize the execution of $G_1$ and $G_2$ at the global level. Also, at site 2, $G_1$ waits for the data item $c$ that is locked by $L_2$. $L_2$ waits for the data item $d$ that is locked by $G_2$ and $G_2$ is waiting for a message from the MDBS that it may proceed at site 2. Figure 3 illustrates the deadlock situation described above. □

One possible reason for a global deadlock is as follows. The MDBS is trying to execute a system of global transactions in one order, while local DBMSs in the presence of local transactions will locally change the order without MDBS's knowledge. This change would not destroy global database consistency. However, it creates a deadlock situation that should be resolved by methods other than simply maintaining the same order of global transactions at each site.

**Theorem 5:** If there is an application of the operator $T$ to a set of global transactions $G$ such that:

a) A site graph of $G$ is acyclic.

b) Any local wait-for-graph is also acyclic.

then no global deadlock may occur. □

## 7. Site Selection Complexity

The observant reader should have noted that there are many different ways to chose a data item copy for a read operation. Theorems 4 and 5 indicate that to ensure global database consistency and avoid global deadlocks, one should try to select reading sites in such a way that a site graph for a set of global transactions will be acyclic. This problem is NP-complete as the next theorem indicates.

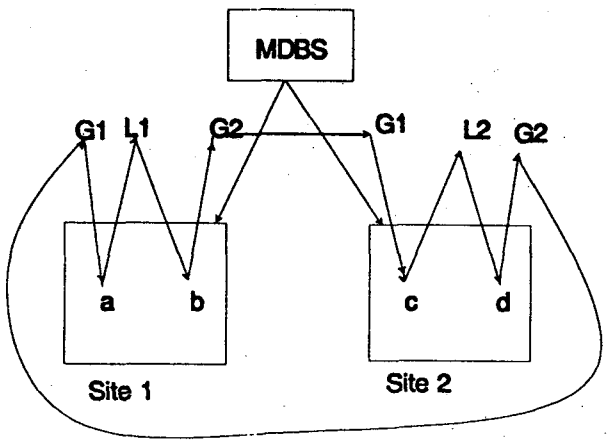


Figure 3. Global Deadlock

**Theorem 6:** For a given set of global transactions $G$, the problem of finding an application of operator $T$ such that a site graph of $G$ is acyclic is NP-complete. □

**Corollary:** The problem to find a site graph with minimal number of sites is NP-complete. □

The NP-completeness of the problem of generating an acyclic site graph seems to dash any hopes of obtaining a fast algorithm to generate an acyclic site graph. In a practical environment, however, the number of local sites is not very large; it further seems that even in exponential numbers of sites, algorithms do not perform too badly in generating acyclic site graphs. In [BREI87c] we propose a simple site selection algorithm and integrate it with a complete set of algorithms to implement a concurrency control schema in a multidatabase environment. This algorithm permits concurrent execution of global and local transactions. However, the level of concurrency for global transactions in this environment may be less than the level of concurrency for global transactions in the absence of local transactions. A performance evaluation of the algorithm is currently being conducted.

**Acknowledgements**

The authors are deeply grateful to Amoco Production Company for its generous support of this research. We are also grateful to G. R. Thompson for numerous inspiring discussions during the preparation of this paper and to M. Truszczynski for pointing out that the set splitting problem can be reduced to the problem of finding an acyclic site graph.

## 8. References

[ALON87]
R. Alonso, H. Garcia-Molina, and K. Salem, "Concurrency Control and Recovery for Global Procedures in Federated Database Systems," IEEE, Data Engineering, September, 1987.

[BERN81]
P. Bernstein and N. Goodman, "Concurrency Control in Distributed Systems," ACM Computing Surveys, Vol. 12, No. 2, 1981.

[BERN85]
P. Bernstein and N. Goodman, "Serializability Theory for Database," Journal of Computer and System Sciences, 31, 1985.

[BERN87]
P. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery inDatabase Systems," Addison-Wesley, 1987.

[BREI85]
Y. Breitbart and L. Tieman, "ADDS - Heterogeneous Distributed Database System," Distributed Data Sharing Systems, Eds. F. Schreiber and W. Litwin, North-Holland, 1985, 7-24.

[BREI87a]
Y. Breitbart, A. Silberschatz, and G. Thompson, "An Approach to the Update Problem in Multidatabase Systems," IEEE Data Engineering, September, 1987.

[BREI87b]
Y. Breitbart and H. Morales, "Multidatabase Performance Evaluation," (in preparation).

[BREI87c]
Y. Breitbart, A. Silberschatz, G. Thompson, "An Update System in a Multidatabase Environment," (in preparation).

[ELMA87]
A. Elmagarmid and Y. Leu, "An Optimistic Concurrency Control Algorithm for Heterogeneous Distributed Database Systems," IEEE Data Engineering, September, 1987.

[FERR83]
A. Ferrier and C. Stangret, "Heterogeneity in the Distributed Database Management System SIRIUS-DELTA," Eight VLDB, Mexico City, 1982.

[GARE79]
R. Garey and S. Johnson, "Computers and Intractability," Freeman & Co., San Francisco, 1979.

[GLIG85]
V. Gligor and R. Popescu-Zeletin, "Concurrency Control Issues in Distributed Heterogeneous Database Management Systems," Distributed Data Sharing Systems, Eds. F. Schreiber and W. Litwin, North-Holland, 1985, 43-56.

[HAMM80]
M. Hammer, and McLeod, "On Database Management System Architecture," Infotech State of the Art Report vol. 8: Data Design, Pergamon Infotech Limited, 1980.

[KORT86]
H. Korth and A. Silberschatz, "Database System Concepts," McGraw-Hill Book Co., 1986.

[LAND82]
T. Landers, and R. Rosenberg, "An Overview of Multibase," Distributed Data Systems, Ed. H. Schneider, North-Holland, 1982, 153-184.

[LITW82]
W. Litwin, J. Boudenant, C. Esculier, A. Ferrier, A. Glorieux, J. La Chimia, K. Kabbaj, C. Moulinoux, P. Rolin, and C. Stangret, "SIRIUS Systems for Distributed Data Management," Distributed Data Bases, Ed. H.J. Schneider, North-Holland, 1982, 311-366.

[MOTR81]
A. Motro and P. Buneman, "Constructing Superviews,"

Proceedings of ACM-SIGMOD International Conference on Management of Data, 1981.

[PAPA79]
C. H. Papadimitriou, "Serializability of Concurrent Updates," J. ACM 26, 4, 1979.

[PU86]
C. Pu,"Superdatabases for Composition of Heterogeneous Databases," Columbia University Technical Report No. CUCS-243-86, 1986.

[PU87]
C. Pu, "Superdatabases: Transactions Across Database Boundaries," IEEE Data Engineering, September 1987.

[TEMP83]
M. Templeton, D. Brill, A. Hwang, I. Kameny, and E. Lund, "An Overview of the MERMAID System - a Frontend to Heterogeneous Databases," Proceedings of EASCON, 1983, IEEE/Computer Society 1983.