MULTIDATABASE CONCURRENCY CONTROL

By

GLENN RAY THOMPSON

Bachelor of Science
Oklahoma State University
Stillwater, Oklahoma
1974

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1976

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
December, 1987

MULTIDATABASE CONCURRENCY CONTROL

Thesis Approved:

Thesis Adviser

Dean of the Graduate College

PREFACE

I would like to express my sincere appreciation to my research advisor, Dr. Yuri Breitbart, for all of his encouragement and guidance. My gratitude also goes to the other members of my committee, Dr. George Hedrick, Dr. Michael Folk, and Dr. Daryl Nord, for their valuable comments during the preparation of the manuscript.

I would like to thank Amoco Production Company for allowing me to use the resources of the Tulsa Research Center to complete my course work and dissertation. Thanks goes to the members of the Database Research Group, Pete Olson, Bing Vassaur, George Thomas, Steve Lee, Tom Reyes, and Hector Morales for many productive brainstorming sessions. Thanks also to the members of the PROFS Group for putting up with my many questions about document preparation.

A special thanks to Dr. Avi Silberschatz for valuable discussions early in my research.

Finally, a heart felt thank you goes to my wife, Vicki, for years of love and support, and to my son, Jeffrey, for allowing me to work at home when I know he had playing football on his mind.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

Heterogeneous Database Problem

Data accessibility is important for the successful operation of any corporation. Historically, however, it has been difficult for individuals to locate and access data within different departments of their own organizations. In many cases, data within an organization is controlled by different database management systems (DBMSs). Some DBMSs are better suited for scientific and engineering applications, while other DBMSs are better suited for business applications. Also, some DBMSs are used simply because of personal preference. Therefore, accessing data from different sources within a corporation usually represents a difficult and specialized task. For these reasons, multidatabase research has increased in the past several years (Breitbart and Tieman, 1985; Ferrier and Stangret, 1983; Landers and Rosenberg, 1982; Litwin et al., 1982; Pu, 1986; Staniszkis et al., 1985; Yu et al., 1985).

Ideally, a multidatabase system (MDBS) supports access to preexisting heterogeneous distributed databases using a

global database model and a global query language (Thompson and Breitbart, 1986). A global database model does not require the user to know the location or understand the characteristics of the physical data. A global query language allows users to access multiple preexisting databases in a single query. Knowledge of the intricacies of the DBMSs supporting the physical data is not required. Several prototype multidatabase retrieval systems have been developed and some of these prototypes have been deployed successfully in industry (Breitbart and Tieman, 1985; Landers and Rosenberg, 1982). A more detailed explanation of a multidatabase system and appropriate terminology is provided in Chapter II.

The multidatabase approach is not the only technique for solving the problems described above. Another solution involves the migration of data to a common DBMS, thereby reducing the complexity of accessing the data (Staniszkis et al.,,1985). If data movement from one geographic location to another is required, the problem again becomes a complex one. Therefore, conversion to a homogeneous distributed DBMS (DDBMS) seems to be the perfect solution to the problem. However, migration of data and application programs to a DDBMS may represent an enormous expenditure of resources, which most organizations are unwilling to allocate. Migration to a DDBMS also defeats the arguments, mentioned previously, for maintaining different DBMSs.

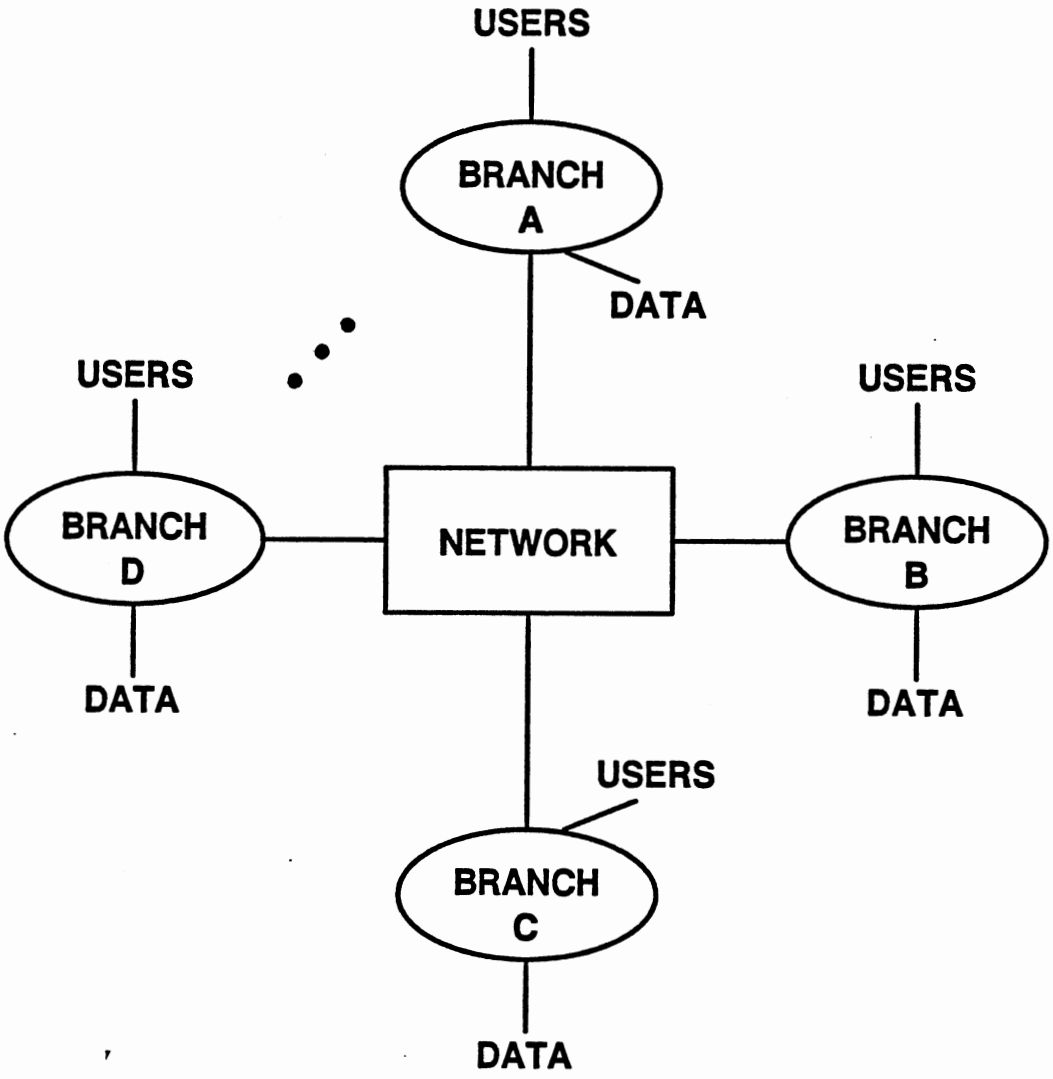


Figure 1. Geographic Distribution of Data Within a Corporation

In many cases, data is distributed geographically throughout different branches of a corporation (see Figure 1). This situation complicates matters when it is

necessary to retrieve the data. However, if physical communication paths are established among the sites involved, the data can usually be maneuvered to the appropriate locations for manipulation. If similar requirements exist for users at several locations, the deployment of a distributed multidatabase system may be appropriate (Thompson and Breitbart, 1986).

A distributed multidatabase system has several advantages over multiple autonomous multidatabase systems: (1) reduces data transmission with enhanced query optimization and localized processing of intermediate query results, (2) establishes a vehicle for data access synchronization, (3) provides better distribution of data processing loads and costs, and (4) eliminates unnecessary direct communication paths to data sources. Chapter IV contains a discussion of several design aspects of distributed multidatabase systems.

Taxonomy of Distributed Database Systems

In general, DBMSs utilize different data models, such as the hierarchical (Tsichritzis and Lochovsky, 1976), relational (Codd, 1970), and network (Bachman and Williams, 1964) models. A variety of different local area and wide area networks may also be utilized to access distributed databases from remote locations (Cole, 1987; Thompson, 1984). However, DDBMSs usually fall into one of three

classes based on the methods of data management and distribution that are utilized (Thompson and Breitbart, 1986). A DDBMS classification scheme is described below.

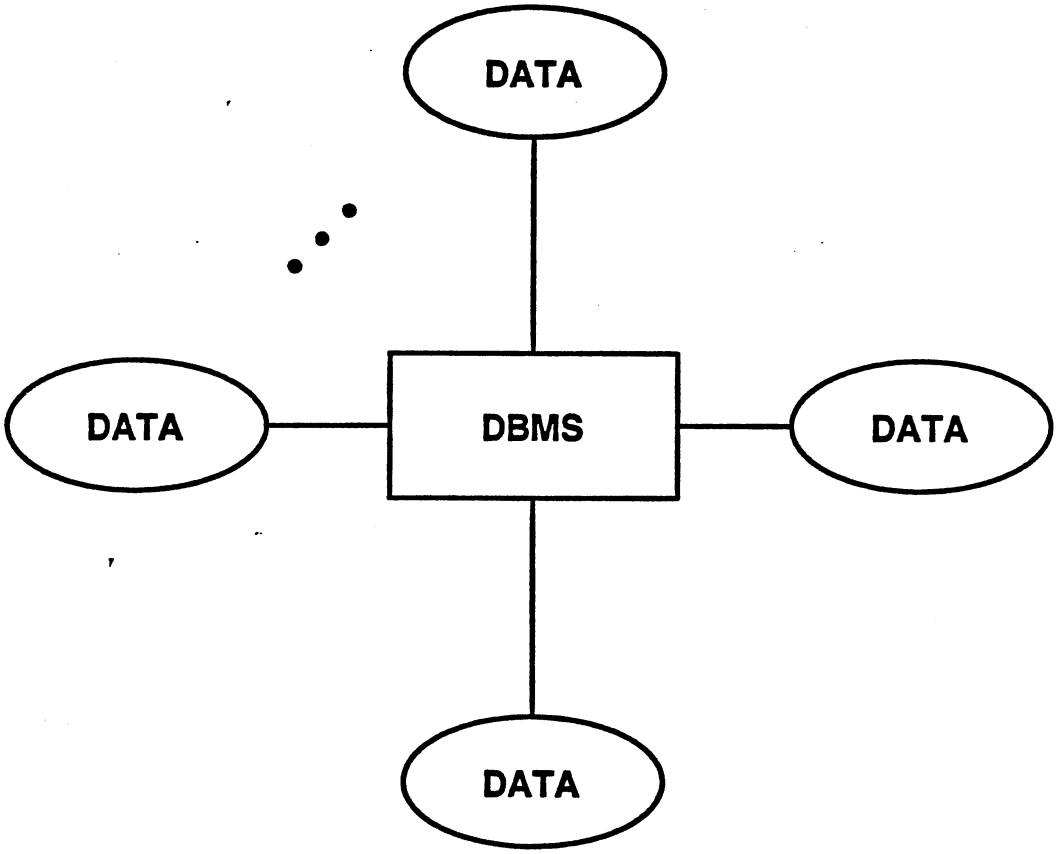


Figure 2. Logical Database Distribution

Class 1, logical database distribution (Figure 2), is characterized by a centralized DBMS managing data distributed among several locations. Each location contains support software for data access by the central DBMS.

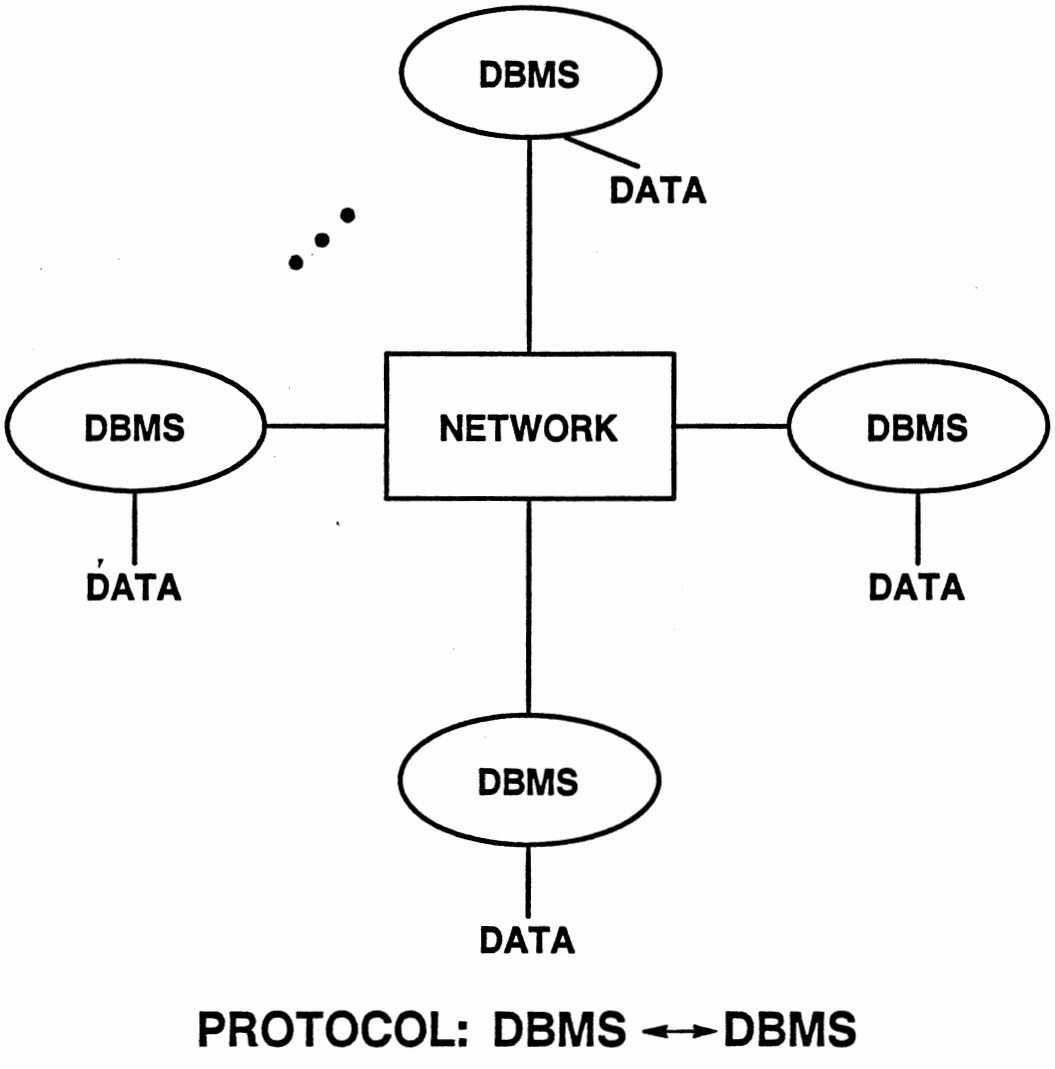


Figure 3. Physical Database Distribution

The logical database distribution model requires the development of data access communication protocols. Most multidatabase systems (e.g. ADDS (Breitbart and Tieman, 1985) and MULTIBASE (Landers and Rosenberg, 1982; Smith et al., 1981)) fall into this category.

Class 2, physical database distribution (Figure 3), is characterized by multiple distributed DBMSs, each controlling access to localized data. Each DBMS operates in a peer-to-peer relationship with all other DBMSs in the network. This technique requires the development of DBMS-to-DBMS communication protocols. System R* (Lohman et al., 1985) and SDD-1 (Bernstein et al., 1981) are examples of homogeneous DBMSs that utilize the physical database distribution technique for accessing distributed data. Heterogeneous SIRIUS-DELTA (Ferrier and Stangret, 1983; Litwin et al., 1982) also uses this technique for data access in a local area network.

Class 3, integrated database distribution (Figure 4), is characterized by multiple distributed DBMSs, each controlling access to a mixture of local and geographically distributed data. Access to geographically distributed data in this category may not be available through the primary network used for DBMS-to-DBMS communication. As with physical database distribution, each DBMS operates in a peer-to-peer relationship with all other DBMSs in the network. This model requires the development of DBMS-to-DBMS and data

access communication protocols. The architecture of the ADDSNET system (Thompson and Breitbart, 1986) is an example of a multidatabase system that utilizes this technique for accessing distributed data. Some of the components of the ADDSNET system are discussed in Chapter IV.

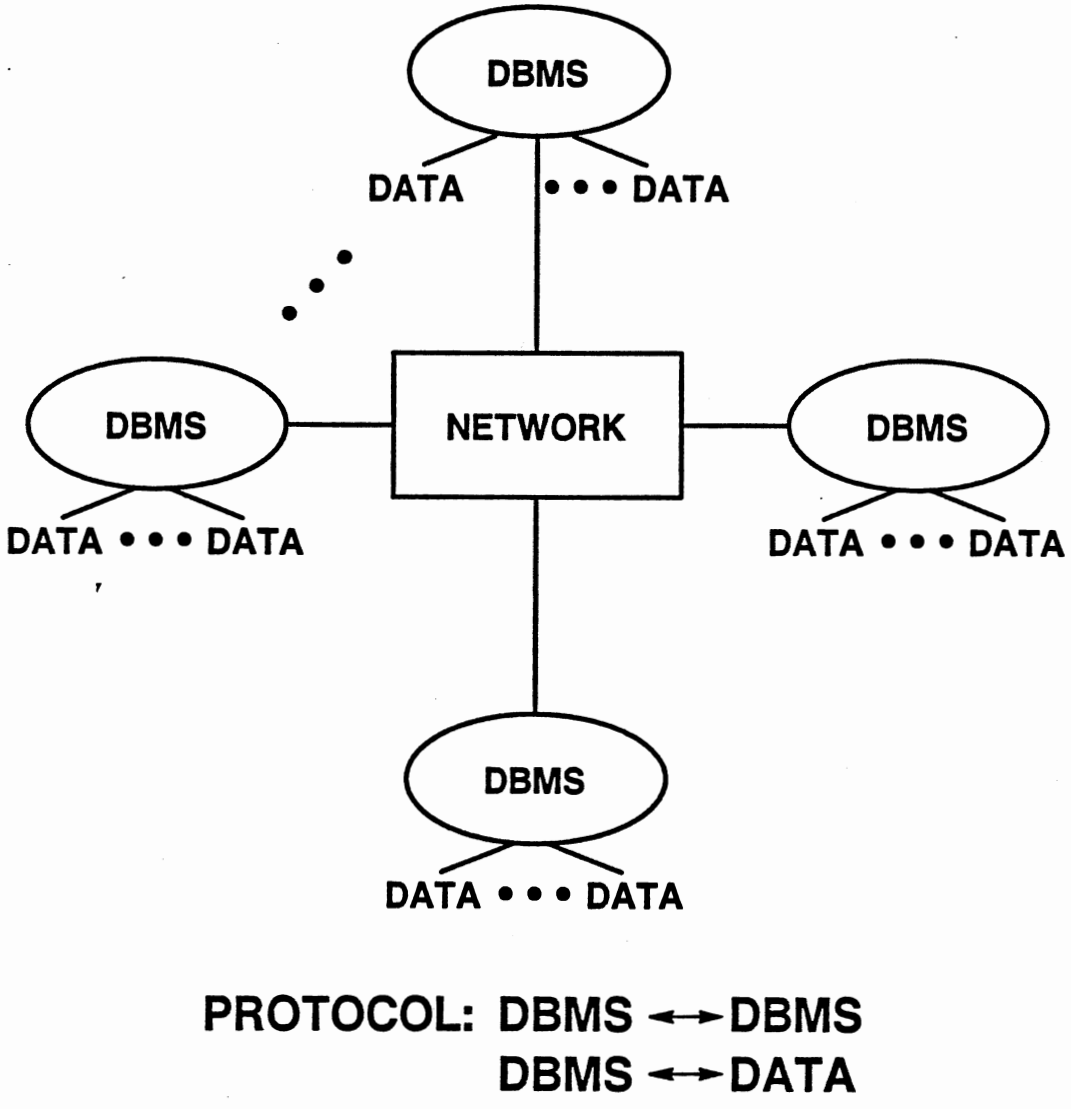


Figure 4. Integrated Database Distribution

A number of multidatabase systems exist that utilize a variety of database models and design techniques. Katz (1981) applies a taxonomical classification scheme to multidatabase systems. In his paper, Katz refers to a "local database" as a physical database from which data is to be retrieved and manipulated. He also refers to a "global data model" as the data model applied to a logical database which consists of one or more fragments of the local databases. His classification of multidatabase systems is based upon the level of freedom of the global data model (i.e. Does the system support one or multiple global data models?), the level of integration of the local databases (i.e. Are the local databases integrated into a single global database?), and the degree of overlap supported for integrated local databases (i.e. Does the system support data fragmentation and replication?).

Comparison of Existing Systems

In this section, the characteristics of several multidatabase systems are examined. These systems include ADDS (Breitbart et al., 1984; Breitbart and Tieman, 1985; Breitbart et al., 1986), DDTS (Devor and Weeldreyer, 1980), MERMAID (Templeton et al., 1983; Yu et al., 1985), MULTIBASE (Landers and Rosenberg, 1982; Smith et al., 1981), NDMS (Staniszkis et al., 1985), POLYPHEME (Adiba et al., 1980; Decitre and Andre, 1980), and Heterogeneous SIRIUS-DELTA (Ferrier and Stangret, 1983; Litwin et al., 1982).

Features	MULTIBASE	DDTS	SIRIUS-DELTA	MERMAID
1. Data transparency supported?	Y	Y	Y	Y
2. Global data model	Daplex	ECR	Rel	Rel
3. Local data models supported	Network Rel Hierarch	Rel	Rel	Rel
4. Preexisting databases supported?	Y	N	N	Y
5. User interfaces: - Interactive? - Programming language?	 Y N	 Y N	 Y N	 Y N
6. Views supported?	Y	Y	Y	Y
7. Query processing: - Separation of global and local optimization? - Take only transmission costs into account? - Use semijoins?	 Y Y Y	 Y Y Y	 Y Y N	 Y Y Y
8. Updates supported?	N	Y	Y	N
9. Two-phase locking?	N	Y	Y	N
10. Deadlocks avoided (A), detected (D)? How?	-	A prevent	A prevent	-
11. Replicated data supported? - Update method for copies	Y -	Y write-all	Y write-all	Y -
12. Two-phase commit?	N	Y	Y	N
13. Distributed multidatabase system?	N	N	Y	N
14. Type of networks supported	LAN and WAN	LAN and WAN	LAN	LAN

Figure 5a. Comparison of Multidatabase Systems, Part A

Features	POLYPHEME	NDMS	ADDS
1. Data transparency supported?	Y	Y	Y
2. Global data model	Rel	Rel	Rel
3. Local data models supported	Rel	Network Rel Hierarch	Network Rel Hierarch
4. Preexisting databases supported?	Y	Y	Y
5. User interfaces: - Interactive? - Programming language?	 Y Y	 Y Y	 Y Y
6. Views supported?	N	Y	Y
7. Query processing: - Separation of global and local optimization? - Take only transmission costs into account? - Use semijoins?	 Y Y N	 Y N Y	 Y Y Y
8. Updates supported?	N	Y	N
9. Two-phase locking?	N	N	N
10. Deadlocks avoided (A), detected (D)? How?	-	D time-out	-
11. Replicated data supported? - Update method for copies	Y -	Y -	Y -
12. Two-phase commit?	N	Y	N
13. Distributed multidatabase system?	N	N	N
14. Type of networks supported	Cyclades LAN	X.25	LAN and WAN

Figure 5b. Comparison of Multidatabase Systems, Part B

Several multidatabase features are of primary concern. Among these are preexisting database support, concurrency control, and distributed system support. Figure 5 contains a tabular comparison of the multidatabase systems mentioned above. The comparisons made are based upon the most recent information available to the author for these systems. Also, some information for the comparisons is taken from Ceri and Pelagatti (1984, pp. 361-385).

Figure 5 indicates that three of the systems examined support update transactions: NDMS, DDTS, and Heterogeneous SIRIUS-DELTA. However, there are significant differences between these systems and the multidatabase model and transaction processing algorithm discussed in Chapters II and III.

The architectural philosophy of NDMS differs greatly from the multidatabase model discussed in Chapter II. NDMS provides distributed processing primitives, such as SEND and RECEIVE, for distributed transaction processing. The logical components (or subtransactions) of a distributed transaction must be written in the language of the local DBMS and the components must communicate using the supported primitives. NDMS does not provide a distributed concurrency control mechanism for maintaining the consistency of a global database. Transaction recovery is managed manually by restoring the global database from a historical journal. NDMS is implemented as a centralized multidatabase system,

supporting access to databases on the MVS[1] and VMS[2] systems.

Heterogeneous SIRIUS-DELTA requires that modifications be made to the local DBMSs to bring the functionality of the local DBMS up to the level of a common "pivot system". Also, local transactions are not permitted outside of the control of the multidatabase system. The "pivot system" on each of the local DBMSs supports transaction commit and recovery algorithms. Heterogeneous SIRIUS-DELTA is implemented as a distributed multidatabase system, supporting access to databases on Honeywell[3] computing systems and various microcomputers.

The DDTS system uses information contained in the concurrency control mechanisms of the local DBMSs to construct its global two-phase locking concurrency control scheme. The local DBMSs are required to support a two-phase locking protocol and modifications to each local DBMS are required to supply appropriate lock information to the global concurrency control mechanism. DDTS is implemented as a centralized multidatabase system, supporting access to databases on Honeywell[3] computing systems.

Objectives and Scope

As stated previously, a large amount of research recently has been conducted in the area of multidatabase

---

[1]MVS is a product of the IBM Corporation.
[2]VMS is a product of the Digital Equipment Corporation.
[3]Honeywell, Incorporated.

systems. However, the problem of updating semantically related data located in preexisting databases has not been addressed sufficiently (Breitbart et al., 1987b). Pu (1986) requires that the multidatabase system be made aware of local transaction execution. Making the MDBS aware of local transaction execution requires changes to the local concurrency control mechanisms to enable the local DBMSs to report local transaction execution information to the MDBS. The MDBS uses this information for local and global transaction 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 homogeneous distributed database management systems.

Another assumption that is frequently made is that retrieve-only multidatabase systems do not require a concurrency control mechanism, since the probability of inconsistent retrievals in the presence of local transactions is quite low (Landers and Rosenburg, 1982). In (Breitbart et al., 1987a), it is shown that while the probability of inconsistent retrieval may be low, it is still possible.

Gligor and Popescu-Zeletin (1985) discuss several inherent difficulties of the update problem in a multidatabase system. These problems are outlined below.

1. Generating and executing subtransactions based on the global transactions submitted to the MDBS,

2. Maintaining global transaction and subtransaction atomicity,

3. Preserving global database consistency, and

4. Avoiding global deadlocks.

The two main objectives of this study are (1) the design of an algorithm for performing database updates in a centralized multidatabase system that solves the problems listed above and (2) the design of a general architecture for distributed multidatabase systems. The multidatabase update problem involves (1) determining the restrictions, if any, that must be applied to global and local transactions and (2) providing an algorithm that performs consistent execution of global and local transactions. The distributed multidatabase system components discussed in this study are the data dictionary and the interconnection network.

The characteristics of transaction execution in a multidatabase system are examined in this study. The model of a multidatabase system outlined in (Breitbart et al., 1987b; Thompson and Breitbart, 1987) and formalized in (Breitbart and Silberschatz, 1987c) is provided in Chapter II. The model serves as a basis for our results presented in Chapters III and IV, and is similar to the database model

described in (Bernstein and Goodman, 1984; Bernstein and Goodman, 1985) The multidatabase update model characterizes the types of transactions permitted and describes the conditions for preserving global database consistency. The problem of global deadlock is also discussed in Chapter II.

The main results of this study are presented in Chapters III and IV. Chapter III describes an algorithm that permits a multidatabase system to update semantically related data items while retaining global database consistency in the presence of global and permitted local transactions. Chapter IV proposes an architecture for a distributed multidatabase system and also proposes solutions to the problems of distributed data dictionary and network management. Chapter V provides a summary of the study and recommendations for further research. The Appendix describes the abbreviations and acronyms used throughout the study.

CHAPTER II

MULTIDATABASE MODEL AND DEFINITIONS

Model

A multidatabase system consists of two or more databases, possibly distributed, which are controlled by one or more DBMSs (Breitbart et al., 1987b; Breitbart and Silberschatz, 1987c; Thompson and Breitbart, 1987). An MDBS allows users to manipulate data contained in the databases without modifying current database applications and without migrating the data to a new database. An MDBS also creates the illusion of logical database integration without requiring physical integration of the databases. For simplicity, the intricacies of the DBMSs and data access methods are transparent to the user.

To provide a facility that is acceptable to the end users, as well as the application programmers, an MDBS should adhere to the following principles.

1. No modifications to the local DBMS software to accommodate the MDBS are permitted.

2. The autonomy of the local databases are maintained.

Preventing changes to the DBMS software is an important issue. Modifying the DBMSs to interact with the MDBS puts a heavy burden on the MDBS developers when support for a new DBMS is added. These changes may also create difficult problems, both in maintaining current applications and in maintaining the DBMS software.

The concept of local autonomy requires that existing local transactions be allowed to execute as if the MDBS were not present. Local autonomy also requires that DBMS maintenance and performance tuning be allowed to continue as usual.

Since changes to the local database software are not permitted, the DBMSs treat the global subtransactions and the local transactions equally. Also, the local DBMSs should perform their operations without the knowledge of other DBMSs and the MDBS. Therefore, local autonomy requirements make the update problem in a multidatabase system significantly different from the update problem in a homogeneous DDBMS.

In a homogeneous DDBMS, when global transactions are submitted, the sites involved in the execution of the transactions communicate to guarantee consistent execution of the transactions. However, this is not the case for the execution of global transactions in a multidatabase system. We assume that the only communication between the MDBS and the local DBMSs is in the form of query submission from the

MDBS to the DBMSs and data transmission from the DBMSs to the MDBS.

In a multidatabase system where only global transactions are permitted, homogeneous DDBMS concurrency control techniques may be employed. However, from a practical standpoint, this restriction significantly diminishes the usefulness of the system. Users should be allowed to submit transactions outside of the control of the MDBS. Permitting the execution of some types of local transactions may create semantic inconsistencies in the global database. An example of such a local transaction is described below.

If a global database contains replicated data, the copies of the data should not be updated by local transactions. Consider, for example, a global database that contains global data item x which has a copy at site A and site B. If a local transaction is submitted at site A that changes the value of x, the global database becomes inconsistent, since the value of x is no longer the same at both sites. Therefore, for a multidatabase system, it is clear that the execution of local transactions that modify local copies of replicated data items must be prohibited.

In this chapter, the mathematical model for performing updates in a multidatabase system constructed in (Breitbart and Silberschatz, 1987c) is described. The model consists of global and local components and uses the notion of one-copy serializability (Attar et al., 1982) to define global

database consistency. Figure 6 illustrates the relationships among the major components of the model.

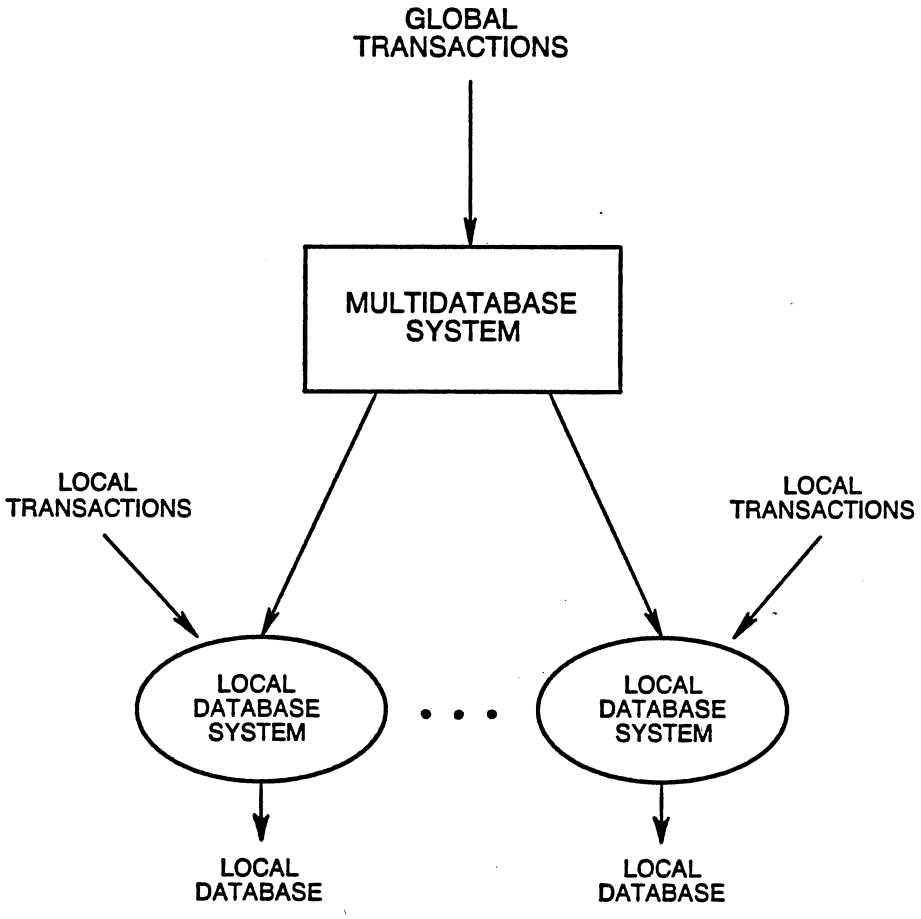


Figure 6. Multidatabase Model

The following assumptions are made about a multidatabase system.

1. The MDBS guarantees serializable global transaction execution.

2. The local DBMSs guarantee serializable local transaction execution.

3. Modifications to the local DBMS software are not permitted.

4. No communication exists among the local DBMSs.

The remainder of this chapter formalizes a multidatabase model based on these assumptions.

Global Components

The global database consists of a set of data items, denoted by x, y, z, ... We treat the data items as abstractions. In practice, the data items may be files, relations, or records (Berstein and Goodman, 1981; Bernstein and Goodman, 1984). The global operations defined on the global database are read(x), which returns the current value of some data item x, and write(x), which assigns a new value to data item x. Read(x) is denoted by r[x] and write(x) is denoted by w[x]. The read and write operations may also be referred to as atomic actions.

The global database system (or MDBS) processes read and write operations on data items in the global database. Interaction with the global database is performed by user

programs called global transactions. A global transaction consists of a sequence of read and write operations on items in the global database. Example 1 provides samples of global transactions. The notation $r_i[x]$ and $w_i[x]$ is used to associate a read or write operation with a global transaction T_i, where x is the data item referenced by T_i.

Example 1

Global Database: x, y, and z
Global Transaction T_1: $w_1[x]$ $r_1[y]$ $w_1[y]$
Global Transaction T_2: $w_2[x]$ $r_2[z]$ $w_2[z]$
Global Transaction T_3: $r_3[x]$ $r_3[y]$
□

In this discussion, a global transaction is considered to be a processing program. We also assume that global transactions are independent and do not communicate with each other.

The read set of a transaction is defined as the set of database items read by the transaction and the write set of a transaction is the set of database items written by the transaction. The write set of a read transaction is empty and the write set of a write transaction contains at least one item. In Example 1, T_1 and T_2 are write transactions and T_3 is a read transaction.

Each global transaction executes as a unit. That is, either a transaction runs to completion or it does not run at all. We further assume if transaction T_i reads and writes x, then operation $r_i[x]$ occurs before operation $w_i[x]$ in the execution of the transaction. We also assume that no transaction reads or writes a data item more than once.

The isolated execution of a global transaction submitted to the MDBS maintains global database consistency. Given a set of global transactions, T_1, T_2, ..., the MDBS may execute them serially, that is, for every pair of transactions T_i and T_j either all of T_i's operations precede all of T_j's or vice versa. Serial execution of only global transactions guarantees global database consistency.

The interleaved execution of atomic actions from different global transactions makes it necessary to introduce the notions of equivalent global schedules and serializable global schedules. Let T_1, T_2, ... be a set of global transactions. We define a global schedule over T_1, T_2, ... as a sequence S of atomic actions of the transactions such that the relative order of the atomic actions for each transaction T_i is retained in S. Example 2 illustrates what is meant by a global schedule.

Example 2

The following sequence of operations is a global schedule for the set of global transactions in Example 1.

$$S = w_1[x]\ r_3[x]\ r_1[y]\ w_2[x]\ w_1[y]\ r_2[z]\ r_3[y]\ w_2[z]$$

□

Let S be a global schedule over global transactions T_1, T_2, ... Transaction T_i reads-x-from T_j in S if (Attar et al., 1982; Bernstein and Goodman, 1985):

1. $W_j[x]$ and $r_i[x]$ are global operations in S,
2. $W_j[x]$ precedes $r_i[x]$ in S,
3. No $w_k[x]$ falls between operations $w_j[x]$ and $r_i[x]$.

In Example 2, T_3 reads-x-from T_1 and T_3 reads-y-from T_1.

Two global schedules over transactions T_1, T_2, ... are equivalent if for all i, j, and x, T_i reads-x-from T_j in one schedule, if and only if, T_i reads-x-from T_j in the other schedule. A global schedule is serializable if it is equivalent to some serial global schedule.

The schedule illustrated in Example 2 is equivalent to the serial schedule $T_1\ T_3\ T_2$. We use serializability as the correctness criteria for the global and local concurrency control mechanisms.

A global serialization graph (Bernstein and Goodman, 1985) for a global schedule S is a directed graph whose nodes represent global transactions and whose arcs are $\{T_i \rightarrow T_j \mid$ there exists operation O_i in T_i and operation O_j in T_j, such that O_i conflicts with O_j and O_i occurs before O_j in S$\}$.

Theorem 1 (Bernstein and Goodman, 1985)

If a global serialization graph for a schedule S is acyclic then S is serializable.

Local Components

The sites of the multidatabase system are a collection of data locations, possibly distributed, denoted by a, b, c, ... A replicated data item has two or more copies, denoted by x_a, x_b, x_c, ..., at sites a, b, c, ...

A local database is a set of data items, denoted by x_a, y_a, z_a, ..., located at the same site a. The local operations defined on the local databases are read(x_a) which returns the current value of some data item x_a and write(x_a) which assigns a new value to data item x_a. Read(x_a) is denoted by r[x_a] and write(x_a) is denoted by w[x_a].

A local database system (or DBMS) processes read and write operations on data items in a local database. A local transaction is a sequence of local read and write operations on items in a single local database. A singular transaction is a local transaction that operates only on non-replicated data items. Read transactions and singular write transactions are the only local transactions permitted. This restriction on local transactions is required to maintain the consistency of replicated data items and is discussed in greater detail in Chapter III.

The definition of local schedule, serializable local schedule, equivalent local schedules, and the correctness of local database concurrency control are the same as the corresponding global definitions, given in the previous section of this chapter, with the replacement of the word "local" for the word "global".

Global Subtransactions

Let x be a replicated data item with copies located at sites a, b, and c. When a global transaction executes operation r[x], it is sufficient to read the value of x at a single site. When global operation w[x] is executed, the copies of x at all sites, a, b, and c, must be written.

To formalize the relationship between global and local operations, we define a translation function. The translation function F(t) maps each global operation r[x] in t onto local operation $r[x_a]$ for some copy x_a of x, and each global operation w[x] in t into local operations $w[x_{a1}]$, ..., $w[x_{am}]$ for all copies x_{a1}, ..., x_{am} of x. Applying function F to a global transaction generates a sequence of operations to be performed on the local databases.

Example 3

Consider the global database of Example 1 consisting of data item x located at site 1, data item y located at sites 2 and 4, and data item z located at site 3. Applying trans-

lation function F to schedule S in Example 2 may yield the following schedule.

$$F(S) = w_1[x_1]\ r_3[x_1]\ r_1[y_2]\ w_2[x_1]\ w_1[y_2]\ w_1[y_4]$$
$$r_2[z_3]\ r_3[y_2]\ w_2[z_3]$$

Notice that operation $r_1[y_2]$ or $r_1[y_4]$ may be chosen to replace operation $r_1[y]$ in schedule S.

□

A subtransaction is a sequence of local operations for a single site which have been derived from a single global transaction. Only one subtransaction is generated for each site from a single global transaction. To assist in generating the subtransactions for a global transaction, we define a projection function. The projection function P(s) groups the local read and write operations in s into sequences of operations, one operation sequence for each site. The result is a set of local operation sequences, where each local operation sequence contains read and write operations for a single site. P(F(T)) yields all subtransactions of global transaction T. The projection of a schedule s on a site i, $P_i(s)$, yields a sequence of local operations containing read and write operations only for site i.

Example 4

Applying projection function P to the result of Example 3 yields the following set of schedules.

$$P(F(S)) = \{w_1[x_1]\ r_3[x_1]\ w_2[x_1],\ r_1[y_2]\ w_1[y_2]\ r_3[y_2],\ r_2[z_3]\ w_2[z_3],\ w_1[y_4]\}$$

□

Global Database Consistency

The following discussion of global database consistency is taken from (Breitbart and Silberschatz, 1987c).

Given a set of global transactions, the execution of these global transactions in the absence of any other transactions at the local sites, is equivalent to the execution of a set of global subtransactions that are generated using the translation function F and the projection function P, discussed in the previous section. Preservation of global database consistency is based on the assumption that the MDBS produces serializable schedules of global transaction execution. This definition of global database consistency can be extended to include the execution of local transactions at multiple sites.

Let G be a set of global transactions and let L be a set of local transactions executed at one or more sites. Breitbart and Silberschatz define an operator Q on the operations of the local transactions, such that

$$Q(r_j(x_i)) = r_j(x) \text{ and}$$
$$Q(w_j(x_i)) = w_j(x),$$

where i represents the site containing data item x and j identifies a local transaction.

Example 5

Applying operator Q to the subtransactions of global transaction T_2 in Example 4 yields the global operations of the original transaction.

$$Q(w_2[x_1]\ r_2[z_3]\ w_2[z_3]) = w_2[x]\ r_2[z]\ w_2[z]$$

□

Applying operator Q to each operation of the local transactions in L yields a system G' of global transactions. Breitbart and Silberschatz argue that the execution of the transactions in G and L retain global database consistency if, and only if, there is a serializable global schedule for {G union G'} that is computationally equivalent to the execution of G by the MDBS and L by the local DBMSs.

If the execution of the global transactions in both G and G' are controlled by the MDBS, then generating a serializable global schedule for a multidatabase system is equivalent to generating a serializable global schedule for a homogeneous DDBMS. However, the MDBS does not control the execution of the transactions in G', based on the assumptions of the model. The problem remains that the MDBS con-

currency control algorithm must produce a serializable schedule for the execution of the global transactions in {G union G'} without any knowledge of G'. Under these conditions, inconsistencies can be introduced into the global database if the transactions in G' contain operations that conflict with operations of the transactions in G.

Breitbart and Silberschatz state that, in a multidatabase system, there are two cases where the execution of local transactions may violate global database consistency.

1. A local transaction changes the value of a replicated data item at only one site.

2. Local transactions contain operations that conflict with the operations of global transactions.

Theorem 2 specifies the conditions for global database consistency in a multidatabase system. The correctness of the theorem depends on the assumption that the local DBMSs produce serializable schedules of local transaction execution.

Theorem 2 (Breitbart and Silberschatz, 1987c)

Let G be a set of global transactions and L be a set of local transactions executed at sites 1, ..., k. The execution of the transactions in the set {G union L} retains global database consistency if the following conditions hold.

1. The local transactions from L are either read transactions or singular write transactions.

2. There exists a total ordering of transactions from G, such that for each pair of atomic operations O_i and O_j from different local transactions ST_i and ST_j that are projections of global transactions G_i and G_j, operation O_i precedes operation O_j in any local schedule if, and only if, G_i precedes G_j in the total ordering.

Global Deadlock

In order to formalize the concept of global deadlock (Gligor and Popescu-Zeletin, 1985), it is necessary to introduce the notion of a global resource allocation graph and a global wait-for graph.

A resource allocation graph consists of a pair RAG = (V,E), where V is a set of vertices and E is a set of edges (Holt, 1971; Holt, 1972; Peterson and Silberschatz, 1983). The set of vertices consists of local data items and transactions that are either waiting for or holding locks on local data items. The set of edges consists of all edges $T_i \rightarrow x$, where transaction T_i has requested a lock on data item x which is already locked by another transaction, and all edges $x \rightarrow T_i$, where T_i owns a lock on data item x. Conflicts among both shared and exclusive locks are considered

here. We assume that a shared lock (read lock) is obtained prior to reading a data item and an exclusive lock (write lock) is obtained prior to writing a data item.

A wait-for graph (WFG) is a simplification of the resource allocation graph, where the set of vertices consists only of transactions that are either waiting for or holding locks on local data items (Holt, 1971; Holt, 1972; Korth and Silberschatz, 1986; Peterson and Silberschatz, 1983). The set of edges consists of all edges $T_i \rightarrow T_j$, where transaction T_i has requested a lock on a data item which is already locked by T_j. An edge of the form $T_i \rightarrow w \rightarrow T_j$ in a resource allocation graph is represented by the edge $T_i \rightarrow T_j$ in a wait-for graph.

When transactions are allowed to hold one or more locks on data items while requesting additional locks, the possibility of transaction deadlock exists. Intuitively, a deadlock condition occurs when a set of transactions exists such that every transaction in the set is waiting for another transaction in the set (Korth and Silberschatz, 1986). A cycle in a resource allocation graph or a wait-for graph indicates the presence of a deadlock condition. If there are no cycles in the graph, then a deadlock condition does not exist. When a deadlock is discovered, at least one of the transactions involved in the cycle must be killed and the effects of the transactions on the database must be canceled.

A local resource allocation graph (LRAG) consists of vertices and edges representing the data item allocation conditions among local transactions and global subtransactions at a single site. Accordingly, a local wait-for graph (LWFG) consists of only local transactions and global subtransactions at a single site (Ceri and Pelagatti, 1984, pp. 219-225). These graphs are maintained by the local DBMSs and are unavailable to the MDBS. Figure 7 contains sample local resource allocation graphs for sites 1 and 2, global transactions T_1 and T_2, and local transactions L_3 and L_4. The notation $T_{i.j}$ defines the subtransaction of global transaction T_i executing at site j.

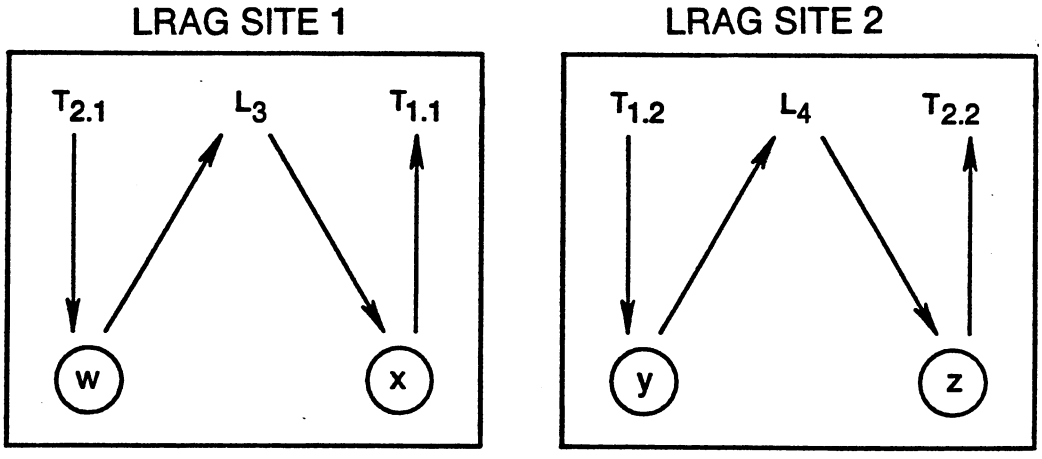
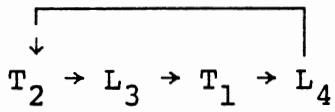


Figure 7. Local Resource Allocation Graphs

A global resource allocation graph (GRAG) is constructed by merging the LRAGs for the set of local sites, where each vertex in the GRAG appears only once. Vertices of the type $T_{i.k}$ and $T_{i.j}$, representing different subtransactions of the same global transaction, are merged into the single vertex T_i. A global wait-for graph (GWFG) is constructed by merging the LWFGs for the set of local sites in much the same way as a GRAG is constructed (Ceri and Pelagatti, 1984, pp. 219-225). LWFGs can be constructed from Figure 7 by "collapsing" the edges of the LRAGs as described in the definition of a wait-for graph. A GWFG can then be constructed from the LWFGs. Notice that the GWFG for the LRAGs in Figure 7 contains the following cycle.



A WFG, and accordingly a RAG, may also contain another type of edge between global subtransactions executing at different sites. This new edge appears in a graph when a subtransaction at one site waits for the execution of a subtransaction at another site due to synchronization enforced by the multidatabase system.

An example of this type of synchronization is the allocation of server resources to global transactions, when only a limited number of server resources are available. A server is an MDBS process that executes a subtransaction for

one global transaction at a time. When a global transaction requires a new server to continue execution and none are available, the transaction must wait until another global transaction releases a server.

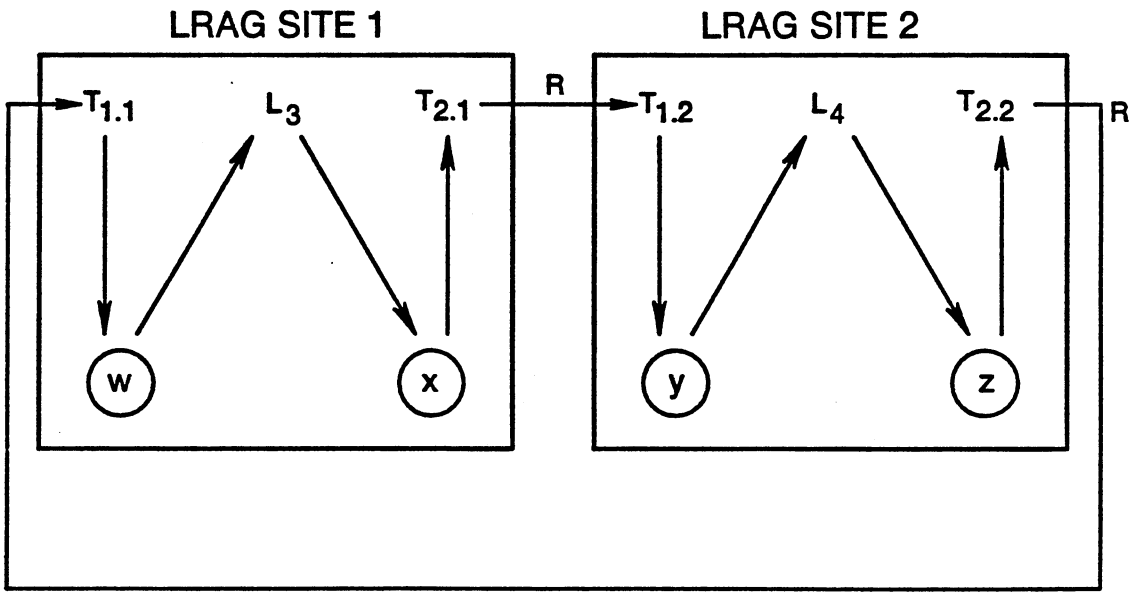
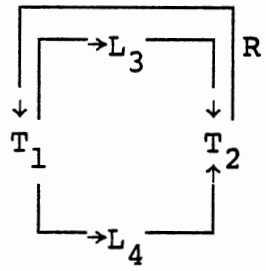


Figure 8. Local Resource Allocation Graphs with MDBS Synchronization

Figure 8 contains two local resource allocation graphs which are joined by arcs, labeled with an R, between sub-transactions of T_2 and T_1. These arcs indicate that the MDBS is attempting to synchronize the execution of global

transactions T_1 and T_2, in that order. The GWFG for the LRAGs in Figure 8 contains the following cycles.



A global deadlock is defined to be a cycle in a global resource allocation graph or a global wait-for graph. The deadlock is undetectable because the local transactions are not included in the graphs maintained by the MDBS and the local database systems have no knowledge of transactions executing at other sites. The reason for the undetectable deadlock is that the MDBS is attempting to synchronize the execution of global transactions, at the local sites, and the local DBMSs have no knowledge of the global synchronization strategy. In fact, any type of synchronization enforced by the MDBS may cause a deadlock. Therefore, an undetectable deadlock must be resolved by some method other than simply maintaining the execution order of the global subtransactions at the local sites.

Gligor and Popescu-Zeletin (1985) describe an undetectable global deadlock in terms of global transactions that reference replicated data items. However, Figures 7 and 9 illustrate a situation where an undetectable deadlock exists

involving two global transactions that reference only non-replicated data items.

Global Transaction T_1	Subtransaction $T_{1.1}$	Subtransaction $T_{1.2}$
× rlock[x]	× rlock[x]	
× r[x]	× r[x]	
□ rlock[y]		□ rlock[y]
□ r[y]		□ r[y]
□ unlock[x,y]	□ unlock[x]	□ unlock[y]

Global Transaction T_2	Subtransaction $T_{2.1}$	Subtransaction $T_{2.2}$
× rlock[z]		× rlock[z]
× r[z]		× r[z]
□ rlock[w]	□ rlock[w]	
□ r[w]	□ r[w]	
□ unlock[w,z]	□ unlock[w]	□ unlock[z]

Local Transaction L_3	Local Transaction L_4
× rlock[w]	× rlock[y]
× r[w]	× r[y]
× rlock[x]	× rlock[z]
× r[x]	× r[z]
× wlock[w]	× wlock[y]
× w[w]	× w[y]
□ wlock[x]	□ wlock[z]
□ w[x]	□ w[z]
□ unlock[w,x]	□ unlock[y,z]

Figure 9. Deadlocked Transactions of Figure 7

In Figure 9, the symbol "×" adjacent to an atomic action indicates that the operation has been completed. The symbol "□" adjacent to an atomic action indicates that the operation has been requested but not completed.

Global transaction T_1 reads data items x and y, and global transaction T_2 reads data items w and z. Local transaction L_3 reads and writes data items w and x, and local transaction L_4 reads and writes data items y and z. In this example, a read lock (rlock) is obtained before reading a data item and a write lock (wlock) is obtained before writing a data item. All read and write locks are released when the transactions are committed.

The following local schedules represent the execution order of the atomic operations of global transactions T_1 and T_2, and local transactions L_3 and L_4. These schedules produce the deadlock situation illustrated in Figures 7 and 9. The site indications for each data item and the read and write lock indications have been eliminated from the schedules for readability. The set of operations preceding the symbol "/" in each schedule are completed. The set of operations following the symbol "/" in each schedule cannot be executed due to the global deadlock.

Site 1: $r_1[x]$ $r_3[w]$ $r_3[x]$ $w_3[w]$ / $r_2[w]$ $w_3[x]$

Site 2: $r_2[z]$ $r_4[y]$ $r_4[z]$ $w_4[y]$ / $r_1[y]$ $w_4[z]$

The transaction processing algorithm described in Chapter III prevents the chain of events that lead to the creation of a global deadlock.

CHAPTER III

A MULTIDATABASE SYSTEM

In Chapter II, a model of transaction execution in a centralized multidatabase system is discussed. In this chapter, we propose an algorithm for maintaining global database consistency in the presence of global and permitted local transactions using the framework of the transaction execution model. The correctness of the algorithm is discussed in the last section of the chapter. The ADDS system (Breitbart and Tieman, 1985; Breitbart et al., 1987b) utilizes the techniques described in this chapter for global transaction execution.

MDBS General Architecture

A multidatabase system provides uniform access to preexisting heterogeneous distributed databases. The ADDS multidatabase system uses a relational data model and an extended relational algebra query language to provide access to distributed data. The local database schemas are mapped into a relational global database schema as described in (Breitbart et al., 1986) and the mappings are stored in the data dictionary. The data dictionary also contains the

physical characteristics and location of the local data. The only communication between the MDBS and the local DBMSs is in the form of query submission and data retrieval. We require that each of the local DBMSs utilize a concurrency control mechanism that maintains local database consistency.

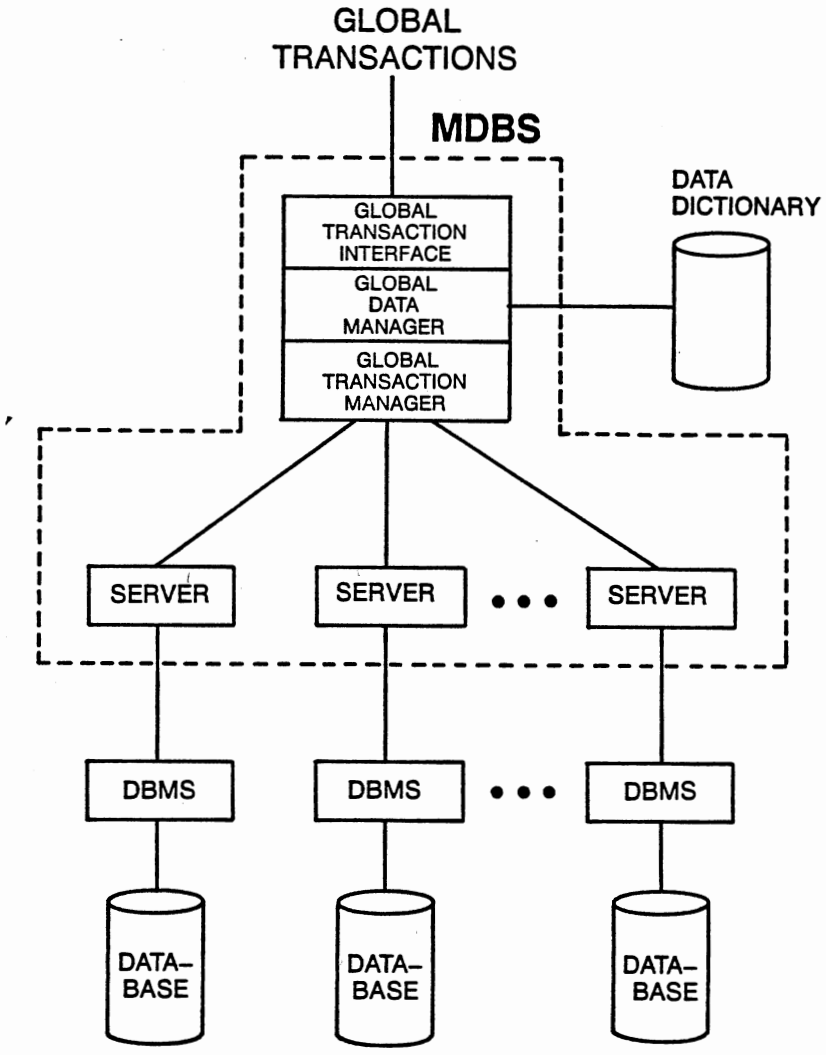


Figure 10. MDBS Architecture

Figure 10 illustrates the proposed layered architecture of a multidatabase system and provides a more detailed look at the multidatabase model contained in Figure 6 (Breitbart et al., 1987b). Global transactions are considered to be processing programs. The global transaction interface (GTI) receives user transactions, ensures their syntactical correctness and generates a global execution plan.

The global data manager (GDM) uses the data dictionary to determine the location or locations of the data referenced by global transactions. The GDM is also responsible for managing all intermediate data that is received from the global transaction manager during transaction execution.

The global transaction manager (GTM) manages the execution of the global transactions. The GTM allocates a server to a global transaction to process read and write operations for data controlled by a single DBMS.

A server is a process that executes a single global subtransaction. In this study, we assume that the MDBS has an unlimited number of servers available for allocation. A server has several responsibilities with respect to global subtransaction execution.

1. Initialize the execution of a global subtransaction at a local site.

2. Translate the global operations into the language of the local DBMS for execution.

3. Manage data transfer between the local DBMS and the GTM.

4. Interface local DBMS commit and abort processing with MDBS commit and abort processing.

The GTM allocates one server to a global transaction for each of the sites referenced by the transaction. A server allocated to a transaction is not released until the transaction has completed execution at each site and the results of the transaction have been committed or aborted by the MDBS.

As operations are received from the global transactions, the GTM sends the global operations to the appropriate servers. If a server is not allocated to a global transaction for a particular site, the GTM allocates a server to the transaction and passes the global operation to the appropriate servers for execution.

When a global transaction completes execution, the GTM instructs the servers allocated to the transaction, to commit the updates to the local databases. The MDBS uses a two-phase commit protocol in communication with the servers to commit the results of a global transaction. The MDBS does not require any specific commit protocol to be supported by the local DBMSs and assumes that any local DBMS is capable of properly committing the results of local transactions. If a global transaction must be aborted, the GTM

instructs the servers to rollback the updates to the local databases.

The layered MDBS architecture discussed here is employed by the ADDS multidatabase system. The current version of ADDS is implemented under the VM/SP[1] operating system. The local databases supported include IMS[2], SQL/DS[3], Inquire[4], RIM[5], and Focus.[6] Communication with the local DBMSs is accomplished using the SNA[7] and Ethernet[8] networks. The current ADDS network nodes include geographically distributed mainframes containing complete ADDS systems, as well as, workstations (e.g., Sun[9] and Apollo[10]) containing ADDS user interface software and connected by local area networks.

Update Algorithm

The notion of a site graph (Thompson and Breitbart, 1987; Breitbart and Silberschatz, 1987c) is central to our discussion of the MDBS update algorithm. We create a site graph of a global transaction T by first determining the

---

[1]VM/SP is a product of the IBM Corporation.
[2]IMS is a product of the IBM Corporation.
[3]SQL/DS is a product of the IBM Corporation.
[4]Inquire is a product of Infodata Systems, Inc.
[5]RIM is a product of the Boeing Computer Services Co.
[6]Focus is a product of Information Builders, Inc.
[7]SNA is a product of the IBM Corporation.
[8]Ethernet is a local area network specification by the Xerox, Intel, and Digital Equipment Corporations.
[9]Sun Workstation is a registered trademark of Sun Microsystems, Inc.
[10]Apollo Computer, Inc.

sites that contain copies of the global data items referenced by T and connecting them as nodes in a graph that has exactly one path between any two nodes. The nodes of the graph are connected by undirected edges labeled with the transaction name T. The data structures for the graph are not specified in this study. The reader is directed to (Aho et al., 1974) for implementation details of undirected graphs.

Given a set of global transactions, if we combine a site graph for each of the transactions into a single graph, we obtain a site graph for the system of global transactions. The next example illustrates the notion of a site graph.

Example 6 (Breitbart et al., 1987b)

Consider a global database that contains data item x at sites 1 and 2, data item y at sites 1 and 3, and data item z at sites 2 and 3. Global transactions T_1 and T_2 are defined in the following way.

$$T_1: w_1(y)\ r_1(x) \qquad T_2: w_2(z)\ r_2(y)$$

The GTM may generate one of the following sequences of local operations for each transaction.

$$T_1: w_1(y_1)\ w_1(y_3)\ r_1(x_1) \qquad T_2: w_2(z_2)\ w_2(z_3)\ r_2(y_3)$$

or

$$T_1: w_1(y_1)\ w_1(y_3)\ r_1(x_1) \qquad T_2: w_2(z_2)\ w_2(z_3)\ r_2(y_1)$$

The site graphs of T_1 and T_2 for each site selection are shown in Figures 11a and 11b, respectively.

□

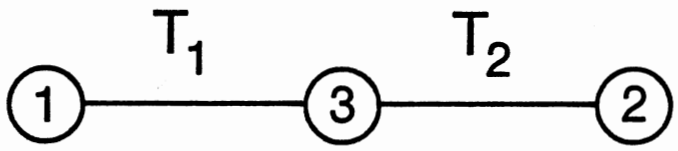


Figure 11a. Site Graph With No Cycles

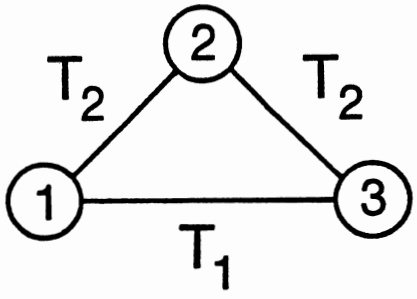


Figure 11b. Site Graph With A Cycle

```
procedure PROCESS_TRANSACTION(T, op, x, Q, n, S):
     begin
     comment The PROCESS_TRANSACTION algorithm synchro-
          nizes the execution of global transactions in a
          manner that maintains global database consistency.
     let T be the global transaction from which the
          global operation is derived, op be the global
          operation to be performed, x be the global data
          item referenced by op, Q be the set of sites
          that contain a copy of x, n be the number of
          elements in Q, and S be the sites currently
          participating in the execution of T.
     if op = 'COMMIT' then
          begin
          invoke COMMIT_TRANS(T)
          return
          end
     if op = 'ABORT' then
          begin
          invoke ROLLBACK_TRANS(T)
          return
          end
     if op = 'READ' then
          invoke SITE_SELECTION(T, Q, n, S)
     else
          begin
          comment op = 'WRITE'
          invoke SITE_GRAPH(T, Q, n, S)
          end
     if the site graph contains a cycle then
          begin
          let T' be the global transaction selected for
               rollback.
          invoke ROLLBACK_TRANS(T')
          if T = T' then
               return
          end
     if Q is not a subset of S then
          begin
          comment Allocate the appropriate number of servers
               to process the current operation.
          allocate the number of servers defined by n minus
               the number of elements in {Q intersect S}
          end
     send the global operation (T, op, x) to the servers
          allocated to transaction T for the sites in Q
     end
```

Figure 12. Transaction Processing Algorithm

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 permitted local transactions. On the other hand, in the absence of cycles in the site graph, the MDBS guarantees correct execution of any mix of global transactions and permitted local transactions and also guarantees the absence of global deadlocks (Breitbart and Silberschatz, 1987c).

The technique used by the MDBS to process read and write operations for a system of global transactions is described below. For all read and write operations, the GDM uses the data dictionary to determine the sites that contain a copy of the referenced data item.

During the initialization of a global transaction, the transaction sends a BEGIN_TRANSACTION message to the GTI process and the GTI process assigns a unique identifier to the transaction. The transaction is now free to submit global read and write operations to the MDBS for execution. Figure 12 describes the algorithm used by the GTM to process the global operations of the global transactions. The components of the transaction processing algorithm are described below.

The PROCESS_TRANSACTION algorithm in Figure 12 describes the global transaction execution process for the MDBS. The steps performed as each global operation (i.e. read, write, commit, or abort) enters the system from the

global transactions is described. Servers are allocated to process the read and write operations, as appropriate.

Upon receiving a read operation, the GTM invokes the SITE_SELECTION algorithm which selects a site that contains a copy of the data item to be read and that does not create a cycle in the site graph. If a site is selected that meets this criteria, the read operation proceeds. Upon receiving a write operation, the GTM adds all the sites that contain a copy of the data item to the site graph. If the addition of these sites to the site graph does not create a cycle, the write operation proceeds.

If any of the sites that contain the data item do not have servers allocated to the transaction, the GTM allocates the required servers and sends the global operation to the servers for execution. For example, if the transaction executes a w[x] operation and global data item x is replicated at sites 1 and 2, the GTM sends the w[x] operation to the servers responsible for accessing data items at sites 1 and 2 for the transaction.

After a server completes a read operation, the server sends the data to the GTM process, which in turn sends the data to the global transaction. After a server completes a write operation, the server simply sends an acknowledgement to the GTM process that the operation is complete and the GTM sends an acknowledgement to the global transaction.

If the addition of the sites where the global operation is to be performed creates a cycle in the site graph, one of the transactions involved in the cycle is rolled back and later restarted. Any one of several transaction restart strategies may be used to select the global transaction to rollback. The ADDS system rolls back the youngest global transaction to break a cycle in the site graph. After a transaction has committed or aborted, all edges labeled with the transaction are removed from the site graph.

The GTM employs a two-phase server allocation strategy which requires that once a server has been deallocated from a transaction, the transaction may not request any additional servers. Also, servers are not deallocated from a transaction until the effects of the global transaction have been committed to the local databases or until the global transaction is aborted. This condition is very important since the failure of a global subtransaction at one or more sites requires that the effects of all other subtransactions for the same global transaction be rolled back.

When a server process is allocated to a global transaction, the appropriate local database processing software is loaded and the server is prepared for local database access. The server is responsible for translating the global operations received into the language of the local DBMS for execution and appears as a single transaction processing program to the local DBMS.

```
procedure SITE_SELECTION(T, Q, n, S):
     begin
     comment The SITE_SELECTION algorithm selects an
          appropriate site for reading the value of a repli-
          cated data item.
     let T be the global transaction from which the
          global operation is derived, Q be the set of
          sites that contain a copy of the data item, n be
          the number of elements in Q, and S be the
          sites currently participating in the execution of
          T.
     if {Q intersect S} is not empty then
          begin
          select a site s from {Q intersect S}
          n ← 1
          Q ← s
          end
     else
          begin
          for i ← 1 step 1 while i <= n do
               begin
               select site Q(i)
               invoke SITE_GRAPH(T, Q(i), 1, S)
               if the site graph is acyclic then
                    begin
                    n ← 1
                    Q ← Q(i)
                    return
                    end
               if i < n then
                    delete the edge just added to the site
                         graph
               end
          select site Q(n)
          Q ← Q(n)
          n ← 1
          end
     end
```

Figure 13. Site Selection Algorithm

The SITE_SELECTION algorithm in Figure 13 selects an appropriate site for reading the value of a replicated data item. Site selection can prevent the unnecessary rollback of a global transaction due to a cycle in the site graph.

If a transaction has already processed data at a site that contains a copy of the data item to be read, the site is selected to perform the read operation. However, if the transaction has not processed any data at a site that contains a copy of the data item, all sites that contain a copy of the data item must be examined individually. If there exists a site that contains a copy of the data item and the addition of the site to the site graph does not create a cycle, the site is selected for the execution of the read operation. A new server for this site is allocated to the transaction. This server then processes all data items that are located at the specified site for the transaction. If no site containing the data item may be added to the site graph without creating a cycle, one of the transactions involved in the cycle is rolled back and later restarted.

The SITE_GRAPH algorithm in Figure 14 adds the specified sites for a global transaction to the site graph. Maintenance of the site graph prevents the possibility of a global deadlock and global database inconsistency, which is shown later in this chapter. Cycles in the site graph are detected by performing a depth first traversal of the graph, producing a spanning tree for the graph, and checking for the existence of any "back edges" (Aho et al., 1974). This

technique provides an efficient method for locating cycles in the site graph.

```
procedure SITE_GRAPH(T, Q, n, S):
     begin
     comment The SITE_GRAPH algorithm adds the specified
          sites for a global transaction to the site graph.
     let T be the global transaction from which the
          global operation is derived, Q be the set of
          sites that contain a copy of the data item, n be
          the number of elements in Q, and S be the
          sites currently participating in the execution of
          T.
     delete the sites from Q that are also in S and adjust
          n accordingly
     if n = 0 then
          return
     comment Add the sites in Q to the site graph for
          transaction T.
     if n > 1 then
          add an edge with label T between each pair of sites
               Q(i) and Q(j), where i = 1, ..., n - 1 and
               j = i + 1
     if set S is not empty then
          begin
          let S(i) be the last site in S that was added to
               the site graph for transaction T.
          add edge (S(i), Q(1)) to the site graph for
               transaction T
          end
     merge the site graph for transaction T with the site
          graph for all global transactions
     end
```

Figure 14. Site Graph Algorithm

The COMMIT_TRANS algorithm in Figure 15 commits the updates performed by the global transactions to the global database. The transaction processing algorithm assumes that

the local DBMSs support a commit protocol for updating the local databases. A COMMIT message is sent to all servers allocated to the specified transaction, indicating that the updates to the local databases must be applied. After the servers complete commit processing with the local DBMSs, the servers are deallocated from the global transaction and are returned to the pool of available servers. After a global transaction is committed, the edges of the site graph associated with the transaction are deleted.

```
procedure COMMIT_TRANS(T):
    begin
     comment The COMMIT_TRANS algorithm commits the
          updates performed by the specified transaction to
          the global database.
     let T be the global transaction to be committed.
     comment The servers use the commit protocol of the
          local DBMSs to apply the updates to each of the
          local databases.
     send a 'COMMIT' message to all servers currently
          allocated to T and receive a 'COMMITTED' response
          from each of the servers
     deallocate the servers from transaction T
     delete the edges for transaction T from the site graph
     end
```

Figure 15. Transaction Commit Algorithm

```
procedure ROLLBACK_TRANS(T):
     begin
     comment The ROLLBACK_TRANS algorithm rolls back the
          specified transaction and performs the necessary
          processing to cancel the effects of the trans-
          action on the global database.
     let T be the global transaction to be rolled back.
     comment The servers cancel all updates to the local
          databases for transaction T and then terminate.
     send an 'ABORT' message to all servers currently
          allocated to T and receive an 'ABORTED' response
          from each of the servers
     deallocate the servers from transaction T
     delete the edges for transaction T from the site graph
     place transaction T on the restart queue
     end
```

Figure 16. Transaction Rollback Algorithm

The ROLLBACK_TRANS algorithm in Figure 16 cancels the affects of the specified global transaction on the global database. An ABORT message is sent to the servers allocated to the global transaction, indicating that the updates to the local databases must be canceled. After the servers have completed abort processing, the servers are deallocated from the global transaction and returned to the pool of available servers. All edges in the site graph associated with the global transaction are deleted and the transaction is placed on the restart queue.

Global transaction recovery protocols are only briefly discussed in this study. Extreme caution must be exercised for global transaction recovery since the MDBS cannot control the recovery actions of the local DBMSs. Additional investigation is suggested in this area.

Correctness of the Update Algorithm

The critical component of the transaction processing algorithm presented in this chapter is the notion of a site graph. Therefore, it is necessary to prove that the maintenance of a site graph for global transactions, together with the execution of only permitted local transactions, maintains global database consistency and prevents global deadlock.

Theorem 3 (Breitbart and Silberschatz, 1987c)

Let G be a set of global transactions and L be a set of local transactions for sites 1, ..., k. The execution of the transactions in the set {G union L} retains a global database consistency if the following conditions hold:

1. All local transactions in L are either read transactions or singular write transactions.

2. The site graph for the global transactions in G is acyclic for at least one application of translation function F, defined in Chapter II, to each transaction in G.

Theorem 4 (Breitbart and Silberschatz, 1987c)

If the conditions of Theorem 3 hold, no global deadlocks can occur.

Careful analysis shows that the algorithm proposed in this chapter solves all of the multidatabase concurrency control problems mentioned in the last section of Chapter I. Global subtransactions are generated using the translation and projection functions described in Chapter II, and a two-phase server allocation strategy is used for the execution of global subtransactions. The allocation of a server to process all of the global operations at a single site for a global transaction, together with a two-phase commit protocol for global transactions, maintains global transaction and subtransaction atomicity. The DBMSs guarantee local database consistency and freedom from local deadlocks, and the site graph algorithm guarantees global database consistency and freedom from global deadlocks. The MDBS, by careful distribution of global operations to the local sites, ensures global database consistency without any information from the local DBMS concurrency control mechanisms.

It should be noted that the algorithm is independent of the types of concurrency control mechanisms used by the local DBMSs. Also, the site graph algorithm is not dependent on the technique used for site specification. That is,

a predeclaration technique for the sites could be employed by the MDBS, instead of the dynamic site specification technique used in the algorithm.

The 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. In our view, the reduction in the level of concurrency is a small price to be paid for retaining local autonomy in a multidatabase system.

A comprehensive study of multidatabase update transaction processing is currently being performed (Breitbart and Morales, 1987e). Initial results indicate that in certain environments (e.g. a fully replicated global database) excessive transaction restarts occur. Excessive transaction restarts for fully replicated global databases appear to be the main deficiency of the proposed algorithm. An alternative to the proposed algorithm is needed for such environments. In practice, however, it is unlikely that a multidatabase system consisting of preexisting databases would be fully replicated. Therefore, we feel that the proposed algorithm has substantial value for practical multidatabase systems.

CHAPTER IV

A DISTRIBUTED MULTIDATABASE SYSTEM

In Chapter III, we propose an algorithm for the consistent execution of global read and write transactions in a multidatabase system consisting of a centralized MDBS process. We mentioned in Chapter I, however, that when users at several locations have the need for distributed data access, the notion of a distributed multidatabase system becomes an important issue. A distributed multidatabase system has several potential benefits: (1) reduces data transmission with enhanced query optimization and localized processing of intermediate query results, (2) establishes a vehicle for data access synchronization, (3) provides better distribution of data processing loads and costs, and (4) eliminates unnecessary direct communication paths to data sources.

In this chapter, we discuss the general architecture of a distributed multidatabase system and relate this architecture to the distributed version of the ADDS multidatabase system (ADDSNET) (Thompson and Breitbart, 1986). We also examine the major components of a multidatabase system which are impacted by the migration to a distributed system. For

convenience, we limit our discussion to that of a retrieve-only distributed multidatabase system. However, the concepts discussed apply to a system supporting update transactions as well.

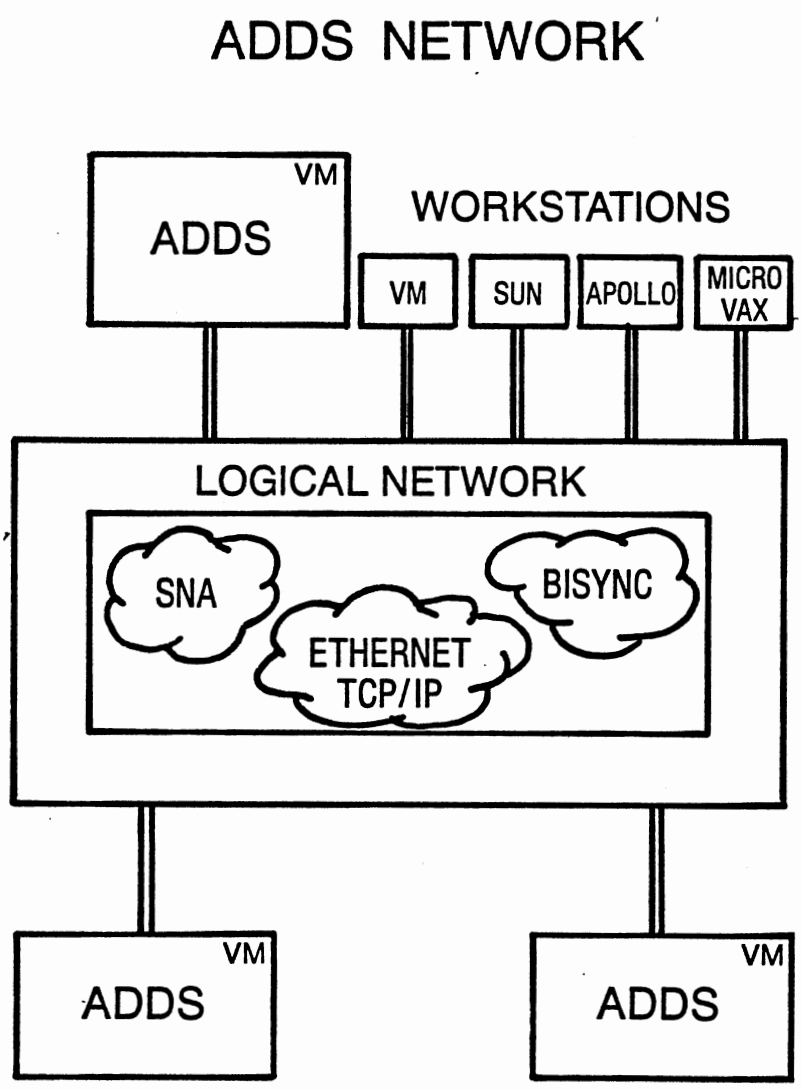


Figure 17. ADDSNET Architecture

ADDSNET General Architecture

The ADDSNET system is an example of a multidatabase system that uses integrated database distribution for accessing distributed databases, as described in Chapter I. Figure 17 illustrates the general architecture of the ADDSNET system.

User queries may be submitted from any ADDSNET node and the queries may reference distributed databases accessible to any ADDSNET node. The ADDSNET system requires that each network node containing data sources for distributed queries, also contain the ADDSNET software. The single exception to this requirement is in a local area network (LAN) environment where multiple workstations are interconnected. In this case, only one of the workstations in the network is required to contain the ADDSNET software. This exception is made because of the limited capacity of most workstations and the great transmission speeds of most LANs.

A key component of the ADDSNET system, as well as any database system, is the data dictionary (Breitbart et al., 1986; Breitbart et al., 1987d). The data dictionary is resident on each ADDSNET node. The next section discusses the architecture of the ADDSNET data dictionary.

Data Dictionary

As mentioned in Chapter III, the local database schemas are mapped into a relational global database schema, called a composite database (CDB), and the mappings are stored in the ADDS data dictionary. The CDBs also contain the physical characteristics and location of the local data. Figure 18 illustrates the major components of the CDB definitions.

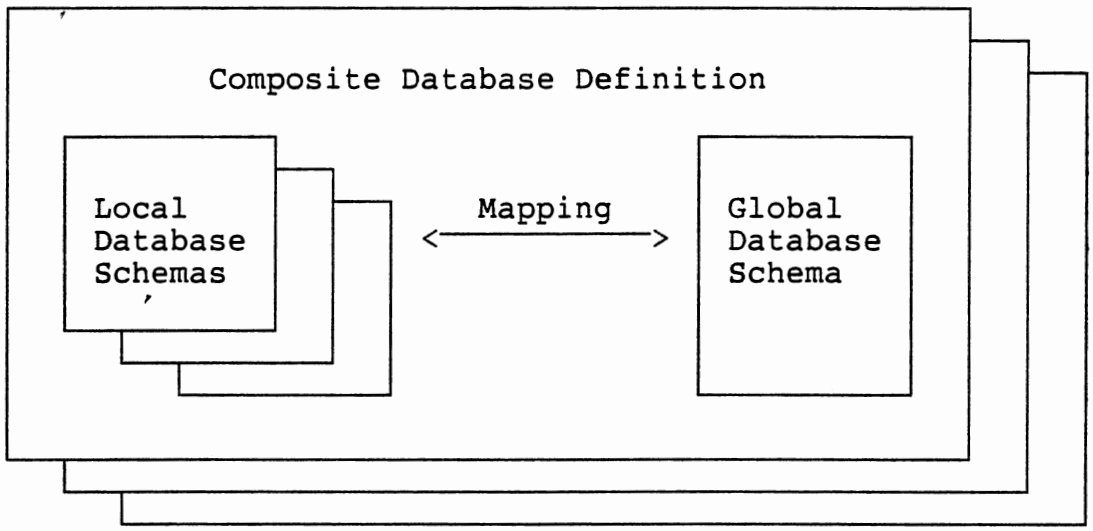


Figure 18. ADDS CDB Definitions

The ADDS query compiler requires data dictionary information to resolve references made to logical relations and attributes. The query optimizer requires data dictionary

information to create a runtime schedule for a query. The database servers require data dictionary information to translate an ADDS user subquery to the language of the local DBMSs.

The data dictionary at each ADDSNET location is logically divided into a "test" and a "production" dictionary. The test dictionary contains composite database definitions that are being developed and tested. A CDB in the test dictionary may reference databases at any of the ADDSNET nodes, however, the definition of the CDB exists only at the local node. An update to a test CDB affects only the local test data dictionary.

The production dictionary is replicated at all ADDSNET locations to enhance the performance of query compilation and execution. A CDB definition is moved to the production dictionary only after the CDB definition has been completely tested. An update to any copy of the production dictionary is propagated to the other copies of the dictionary at the other ADDSNET locations.

The ADDS data dictionary consists of a set of relational tables that contain the components of the CDB definitions. There are at least two techniques for managing the production data dictionary for the ADDSNET system: (1) store the set of tables under the control of a homogeneous DDBMS and let the DDBMS perform the updates to the copies of the data dictionary at the ADDSNET nodes, or (2) store the

set of tables under the control of separate non-distributed DBMSs at the ADDSNET nodes and define a CDB for the tables and let ADDSNET perform the updates to the copies of the data dictionary at the different nodes. Option (2) relies on the availability of a distributed version of the site protocol of Chapter III for the ADDSNET system.

When a user query against a production CDB is initiated, the subqueries are transmitted to the appropriate ADDSNET locations for execution. Since the production CDB is replicated at all nodes, no CDB information needs to accompany the subqueries sent to the ADDSNET nodes. However, when a user query against a test CDB is initiated, the subqueries and necessary components of the test CDB are transmitted to the appropriate ADDSNET locations for execution. Test CDB information is required by the servers at the ADDSNET locations where the subqueries are ultimately executed. An ADDSNET node receives a subquery from another location only when the node supports access to the databases referenced by the query.

This technique provides a flexible and efficient mechanism for managing a distributed MDBS data dictionary. Only a minimum amount of global schema information is transmitted between sites to accomplish query execution. Also, minimal overhead is incurred managing the production dictionary, since the production global schema definitions are relatively static.

Logical Network

Users of a distributed multidatabase system may require access to a wide variety of different hardware and software systems. Therefore, providing a uniform interface to these systems is important for MDBS development and maintenance. Also, the flexibility of the network architecture is of primary concern. Listed below are the major requirements of the ADDSNET network architecture (Lee et al., 1987; Thompson, 1984).

1. The network must be flexible enough to support virtually any physical network utilized. In some cases, several wide area and local area networks may be utilized by a single network node.

2. The network must be capable of migrating to a replacement physical network with minimal modifications.

3. The network protocols must be flexible enough to support complex communication, such as remote sessions, as well as less complex message and broadcast communication.

4. The network must not compromise established security procedures for data access.

The open systems interconnection (OSI) reference model (ISO, 1982) provides a flexible and consistent framework for

network specification. The layered approach of the OSI model provides the generality necessary for the interconnection of very diverse systems. This is the main reason for utilizing an extended OSI model for the ADDSNET network architecture.

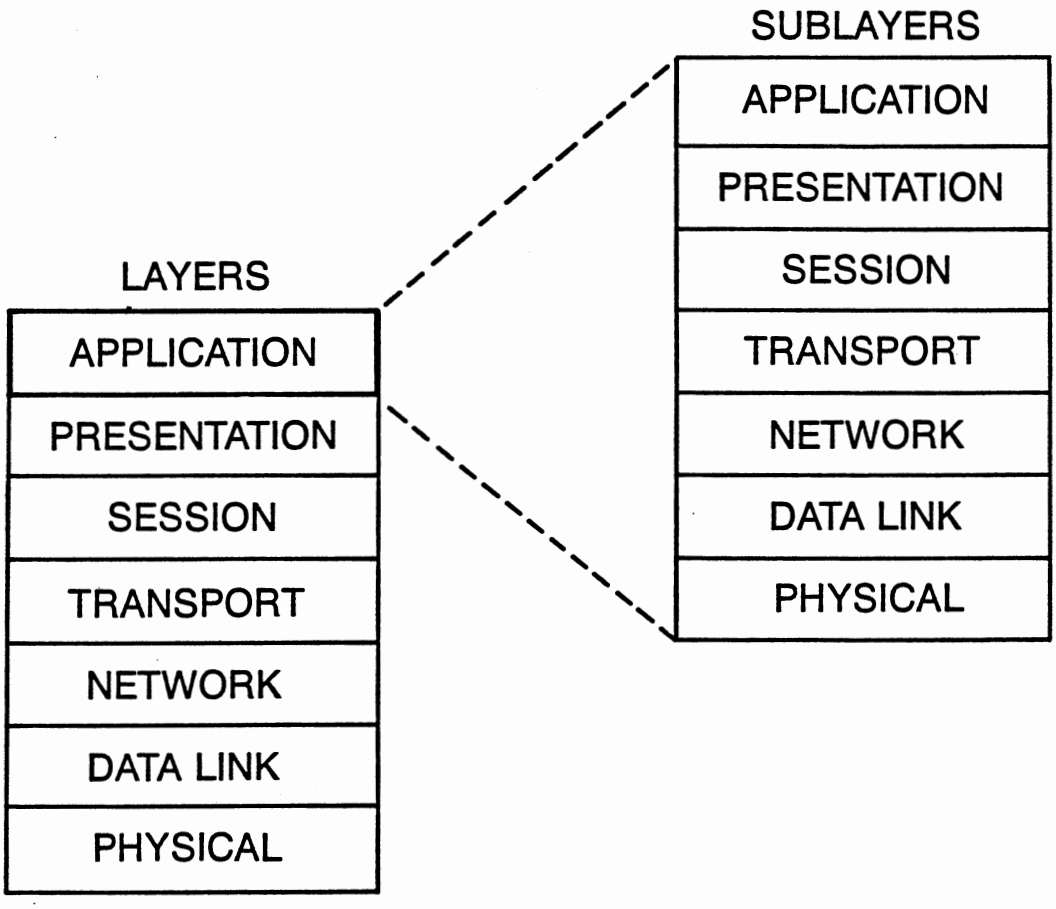


Figure 19. ADDSNET Logical Network Model
(Extended OSI Model)

The ADDSNET network architecture extends the layered approach used in the OSI model one step further into the OSI application layer. The ADDSNET network architecture divides the OSI application layer into sublayers that describe a "logical" network architecture (Lee et al., 1987; Thompson, 1984). Figure 19 illustrates the ADDSNET logical network model.

Logical Network Components

Network Controller. The network controller (NC) process is the primary component of the ADDSNET logical network. A single NC process that performs session and logical network management is located at each network node. The NC process embodies the presentation, transport, and session sublayers of the logical network model. Communication with the NC process is accomplished using a set of procedure calls. These procedure calls define the application sublayer interface to the logical network. The various communication protocols supported by the logical network are discussed later in this section. The NC process is actually composed of four asynchronous processes. These processes perform logical network control, timing, interprocess communication, and operator command processing.

Network Interface. The network interface (NI) is the only process within the logical network that is physical network dependent. One NI process is allocated for each

physical network supported by a logical network node. An NI process is responsible for direct communication with a specific physical network and embodies the network, data link, and physical sublayers of the logical network model. The NC process transmits logical network packets to the NI processes destined for other network nodes. The NI processes transfer the logical packets to the physical networks using the established communication procedures for the physical networks. The process is reversed for incoming network traffic.

A logical network packet may consist of one or more physical packets, depending on limitations that may be imposed on the length of physical packets or messages. An NI process is responsible for the appropriate segmentation and reconstruction of logical packets as they are transmitted. It may also be necessary for a NI process to map logical network node names to physical node names, depending on the naming conventions established for the physical network.

Each NI process maintains a single session with the NI processes at other nodes in the same physical network. The NC process, together with the NI process for a specific network, multiplexes many logical network sessions over a single physical network session. This technique helps to eliminate session limitations associated with some physical network implementations.

Network Definition. The network definition language (NDL) defines a logical network node and its interconnection with one or more physical networks. The network definition file contains the NDL statements which are interpreted at NC process initialization. Each logical network node has its own set of NDL statements. The NDL is general in design to prevent the exclusion of any physical network because of its nonconformity to predefined logical network specifications. System dependent information, such as the character set used by a logical network node, is contained in the NDL statements to allow the NI processes to perform appropriate system dependent functions.

Logical Network Protocols

Three logical network protocols are currently implemented: (1) a session, (2) a message, and (3) a broadcast protocol. Each protocol provides a unique function within the logical network. The session protocol is used for process-to-process communication when a high degree of integrity for a logical connection is required. The message and broadcast protocols are less complex protocols that provide communication among distributed processes. An application process may communicate with any number of other application processes without regard for the physical connections or network software required to accomplish the communication. Figure 20 illustrates this feature of the logical network.

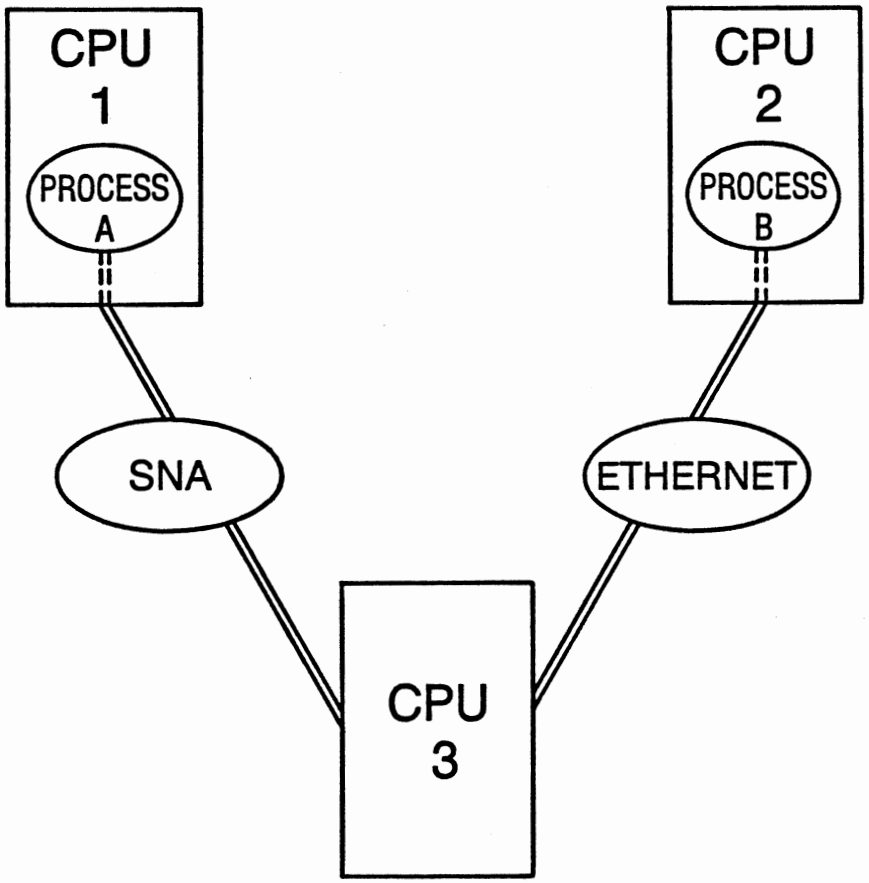


Figure 20. Sample Logical Network Session

Target processes for all of the protocols are identified by a logical network function name. A source process communicates with a target process using the function name of the target process. When an application process is initialized and prepared to receive communication, the process declares its function name to the local NC process. A

source process is required to identify the logical network node name and the function name of the target process whenever any of the protocols are used. The same function name may be active at many different network nodes. However, at a single node, a function may be associated with only one application process. Also, a single application process may perform many functions.

Session Protocol. The session protocol provides distributed applications with a reliable logical connection between application processes that may span heterogeneous hardware and software. A session is initiated by requesting session services from the local NC process. The NC process manages all logical network sessions and provides support for session error recovery across the physical networks.

Message Protocol. The message protocol is designed to provide a less complex communication path between distributed application processes. This protocol provides a vehicle for the implementation of higher level protocols among application processes. A message is transmitted by requesting message services from the local NC process. The NC process manages the transmission of the specified message to the remote network node.

Broadcast Protocol. The broadcast protocol supports the transmission of "global" messages to all logical network nodes. The source process defines the function name of the target process, possibly located at all network nodes, to

receive the specified message. The source process then requests broadcast services from the local NC process. The NC process manages the communication of the specified message to all network nodes defined.

CHAPTER V

SUMMARY AND RECOMMENDATIONS

Summary

The problem of managing heterogeneous distributed databases is becoming an increasingly difficult problem due to an ever increasing number of different DBMSs utilized in many corporations. Many retrieve-only multidatabase systems have been developed that attempt to provide a tool for managing heterogeneous distributed data sources. However, most of these systems have not progressed beyond early prototype stages.

The problem of updating semantically related data located in preexisting heterogeneous databases has not been addressed in sufficient depth. Some multidatabase systems have been developed that perform updates to different local databases. However, most of these systems either ignore global database consistency or require changes to the local DBMS software to accommodate the multidatabase system.

We feel that both of these options are unacceptable. Maintaining global database consistency and local database autonomy are of critical importance for user acceptance of a multidatabase system.

A multidatabase transaction processing algorithm, based on a "site protocol" concurrency control mechanism, is proposed as a solution to the problem of updating heterogeneous distributed databases. The proposed algorithm maintains global database consistency in the presence of global and permitted local transactions, and eliminates the possibility of global transaction deadlock. A model of a centralized multidatabase system is described in this study to provide a foundation for the transaction processing algorithm.

An architecture for a distributed multidatabase system is presented and solutions to the problems of distributed data dictionary and network management are proposed. The notion of a test/production dictionary is proposed as a flexible and efficient means for dictionary management in a distributed multidatabse system. The concept of a logical network provides an effective means of integrating heterogeneous networks.

Recommendations for Further Research

In Chapter III, a transaction processing algorithm for a multidatabase system is described. The algorithm is designed to maintain global database consistency for a centralized multidatabase system in the presence of global and permitted local transactions. Further investigation is required to extend the algorithm to support transaction processing in a distributed multidatabase system, such as

ADDSNET. In particular, a distributed site graph algorithm must be constructed.

The transaction processing algorithm makes the assumption that an inexhaustible pool of server resources are available for global subtransaction execution. This assumption may not be practical for some systems, where only a small number of server resources may be available for allocation. This situation may produce global deadlocks when a global transaction waits for server resources held by one or more global transactions. Therefore, the finite server allocation problem should be investigated further.

The concurrency control component (site protocol) of the transaction processing algorithm in Chapter III is substantially different than conventional concurrency control schemes, such as two-phase locking and optimistic concurrency control. Therefore, a thorough performance analysis of the algorithm should be performed to determine the levels of transaction concurrency provided under varying conditions. For example, the ratio of the number of read to write operations in a global transaction and the number of replicated data items can be varied to determine their effect on global transaction throughput. As stated in Chapter III, Breitbart and Morales (1987e) are currently investigating the performance characteristics of the proposed algorithm.

From the initial results obtained by Breitbart and Morales (1987e), excessive global transaction restarts appear to be the main deficiency of the proposed algorithm. If a technique, such as site preallocation, is utilized for all global transactions, transaction restarts due to cycles in the site graph would be eliminated. However, global transaction restarts initiated by the local DBMSs are still possible. The performance characteristics of the site preallocation technique should be evaluated.

For the model described in Chapter II, we assume that local read and singular write transactions are permitted. However, in an environment where only local read transactions are permitted, it may be possible to increase transaction concurrency by modifying the site protocol, taking into consideration the limitation on local transactions.

Global transaction commit and recovery protocols are not discussed in detail in this study. Extreme caution must be exercised for global transaction recovery since the multidatabase system does not control the recovery actions of the local DBMSs. Additional investigation is required in this area.

SELECTED BIBLIOGRAPHY

Adiba, M., J. M. Andrade, P. Decitre, F. Fernandez, and N. G. Toan. "Polypheme: An Experience in Distributed Database System Design and Implementation." Distributed Data Bases. Eds. C. Delobel and W. Litwin. North-Holland, 1980, 67-84.

Aho, A. V., J. E. Hopcroft, and J. D. Ullman. The Design and Analysis of Computer Algorithms. Addison-Wesley, 1974, 172-179.

Attar, R., P. A. Bernstein, and N. Goodman. "Site Initialization, Recovery, and Backup in a Distributed Database System." Proceedings of the Sixth Berkeley Workshop on Distributed Data Management and Computer Networks, (Feb. 1982), 185-202.

Bachman, C. W. and S. S. Williams. "A General Purpose Programming System for Random Access Memories." Proceedings of the Fall Joint Computer Conference, 26, AFIPS Press, 1964, 411-422.

Bernstein, P. A., N. Goodman, E. Wong, C. L. Reeve, and J. B. Rothnie, Jr. "Query Processing in a System for Distributed Databases (SDD-1)." ACM Transactions on Database Systems, 6, 4 (Dec. 1981), 602-625.

Bernstein, P. A. and N. Goodman. "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases." ACM Transactions on Database Systems, 9, 4 (Dec. 1984), 596-615.

Bernstein, P. A. and N. Goodman. "Serializability Theory for Databases." Journal of Computer and System Sciences, 31, 1985, 355-374.

Breitbart, Y. J., L. F. Kemp, Jr., G. R. Thompson, and A. Silberschatz. "Performance Evaluation of a Simulation Model for Data Retrieval in a Heterogeneous Database Environment." Proceedings of the Trends and Applications Conference, (May 1984), 190-197.

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

Breitbart, Y. J., P. L. Olson, and G. R. Thompson. "Database Integration in a Distributed Heterogeneous Database System." Proceedings of the Second International Conference on Data Engineering, (Feb. 1986), 301-310.

Breitbart, Y. J., A. Silberschatz, and G. R. Thompson. An Approach to the Update Problem in Multidatabase Systems. Research Technical Report No. F87-C-11, Amoco Production Co., Tulsa, OK, 1987a.

Breitbart, Y. J., A. Silberschatz, and G. R. Thompson. "An Update Mechanism for Multidatabase Systems." Data Engineering, 10, 3 (Sept. 1987b), 12-18.

Breitbart, Y. J. and A. Silberschatz. "Multidatabase Update Issues," 1987c (in preparation).

Breitbart, Y. J., W. F. Lee, P. L. Olson, Y. Y. Sung, G. F. Thomas, and G. R. Thompson. "Catalog Management in a Heterogeneous Distributed Database System," 1987d (in preparation).

Breitbart, Y. J. and H. A. Morales. "Performance Evaluation of a Multidatabase Update Algorithm," 1987e (in preparation).

Ceri, S. and G. Pelagatti. Distributed Databases Principles and Systems. McGraw-Hill, Inc., 1984.

Codd, E. F. "A Relational Model for Large Shared Data Banks." Communications of the ACM, 13, 6 (June 1970), 377-387.

Cole, R. "A Method for Interconnecting Heterogeneous Computer Networks." Software-Practice and Experience, 17, 6 (June 1987), 387-397.

Decitre, P. and E. Andre. "Polypheme Project: The DEM Distributed Execution Monitor." Distributed Data Bases. Eds. C. Delobel and W. Litwin. North-Holland, 1980, 85-98.

Devor, C. and J. Weeldreyer. DDTS: A Testbed for Distributed Database Research. Honeywell Report HR-80-268, Honeywell Corporate Computer Science Center, Bloomington, Minnesota, 1980.

Ferrier, A. and C. Stangret. "Heterogeneity in the Distributed Database Management System Sirius-Delta." Proceedings of the Eighth VLDB Conference, 1983, 45-53.

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

Holt, R. C. "Comments on Prevention of System Deadlocks." Communications of the ACM, 14, 1 (Jan. 1971), 36-38.

Holt, R. C. "Some Deadlock Properties of Computer Systems." ACM Computing Surveys, 4, 3 (Sept. 1972), 179-196.

ISO International Standards Organization, ISO/TC97: Information Processing Systems. Open Systems Interconnection - Basic Reference Model. Draft International Standard ISO/DIS 7489, (April 1982).

Katz, R. "Software Architectures for Heterogeneous Database Management." Proceedings of the COMPSAC Conference, 1981, 33-42.

Korth, H. F. and A. Silberschatz. Database System Concepts. McGraw-Hill, Inc., 1986, 390-402.

Landers, T. and R. L. Rosenberg. "An Overview of Multibase." Distributed Data Bases. Ed. H. J. Schneider. North-Holland, 1982, 153-184.

Lee, W. F., P. L. Olson, G. F. Thomas, and G. R. Thompson. A Remote User Interface for the ADDS Multidatabase System. Research Technical Report No. F87-C-12, Amoco Production Co., Tulsa, OK, 1987.

Litwin, W., J. Boudenant, C. Esculier, A. Ferrier, A. M. 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.

Lohman, G. M., C. Mohan, L. M. Haas, D. Daniels, B. G. Lindsay, P. G. Selinger, and P. F. Wilms. "Query Processing in R*." Query Processing in Database Systems. Eds. W. Kim, D. S. Reiner, and D. S. Batory. Springer-Verlag, 1985, 31-47.

Peterson, J. and A. Silberschatz. Operating System Concepts. Addison-Wesley, Inc., 1983, 257-286.

Pu, C. Superdatabases for Composition of Heterogeneous Databases. Technical Report No. CUCS-243-86, Columbia University, New York, NY, 1986.

Smith, J. M., P. A. Bernstein, U. Dayal, N. Goodman, T. A. Landers, K. W. T. Lin, and E. Wong. "Multibase: Integrating Heterogeneous Distributed Database Systems." Proceedings of the AFIPS National Computer Conference, 1981, 487-499.

Staniszkis, W., W. Kaminski, M. Kowalewski, K. Krajewski, S. Mezyk, and G. Turco. "Architecture of the Network Data Management System." Distributed Data Sharing Systems. Eds. F. A. Schreiber and W. Litwin. North-Holland, 1985, 57-75.

Templeton, M., D. Brill, A. Hwang, I. Kameny, and E. Lund. "An Overview of the Mermaid System - A Frontend to Heterogeneous Databases." Proceedings of the EASCON Conference, (Sept. 1983), 387-402.

Thompson, G. R. A Network Independent Architecture for Communicating Network Systems. Research Technical Report No. F84-C-24, Amoco Production Co., Tulsa, OK, 1984.

Thompson, G. R. and Y. J. Breitbart. "Design Issues in Distributed Multidatabase Systems." Proceedings of the Workshop on Applied Computing, Stillwater, OK, (Oct. 1986), 38-46.

Thompson, G. R. and Y. J. Breitbart. A Method for Consistent Multidatabase Transaction Processing. Patent Application Serial No. 078129, Amoco Production Co., Tulsa, OK, 1987.

Tsichritzis, D. C. and F. H. Lochovsky. "Hierarchical Data-base Management: A Survey." ACM Computing Surveys, 8, 1 (March 1976), 67-103.

Yu, C. T., C. C. Chang, M. Templeton, D. Brill, and E. Lund. "Query Processing in a Fragmented Relational Distributed System: Mermaid." Transactions on Software Engineering, SE-11, 8 (Aug. 1985), 795-810.

APPENDIX

ABBREVIATIONS AND ACRONYMS

ADDS	Amoco Distributed Database System
ADDSNET	Amoco Distributed Database System Network
CDB	Composite DataBase
CPU	Central Processing Unit
Cyclades	local area network used by the Polypheme system
Daplex	data definition language and model used by the Multibase system
DBMS	DataBase Management System
DDBMS	Distributed DataBase Management System
DDTS	Distributed Database Test System
ECR	Entity Category Relationship data model
GDM	Global Data Manager
GRAG	Global Resource Allocation Graph
GTI	Global Transaction Interface
GTM	Global Transaction Manager
GWFG	Global Wait-For Graph
Hierarch	Hierarchical data model
IBM	International Business Machines corporation
IMS	Information Management System
ISO	International Standards Organization
LAN	Local Area Network

LRAG	Local Resource Allocation Graph
LWFG	Local Wait-For Graph
MDBS	MultiDataBase System
MVS	Multiple Virtual Spaces operating system
NC	Network Controller process
NDL	Network Definition Language
NDMS	Network Data Management System
NI	Network Interface process
OSI	Open Systems Interconnection
r	read atomic operation
RAG	Resource Allocation Graph
Rel	Relational data model
RIM	Relational Information Management system
rlock	read (shared) lock
SDD-1	System for Distributed Databases
SNA	Systems Network Architecture
SQL/DS	Structured Query Language/Data System
VM/SP	Virtual Machine/System Product operating system
VMS	Virtual Memory System operating system
w	write atomic operation
WAN	Wide Area Network
WFG	Wait-For Graph
wlock	write (exclusive) lock
X.25	packet switching network protocol standard

VITA

GLENN RAY THOMPSON

Candidate for the Degree of

Doctor of Philosophy

Thesis: MULTIDATABASE CONCURRENCY CONTROL

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Fort Smith, Arkansas, July 16, 1952, the son of Wendell R. and Margaret G. Thompson. Married to Vicki L. Green on August 3, 1974. Son, Jeffrey B. Thompson, born on November 9, 1982.

Education: Graduated from Mount Saint Mary High School, Oklahoma City, Oklahoma, in May, 1970; received Bachelor of Science Degree in Mathematics from Oklahoma State University in May, 1974; received Master of Science Degree in Computing and Information Sciences from Oklahoma State University in May, 1976; completed requirements for the Doctor of Philosophy Degree at Oklahoma State University in December, 1987.

Professional Experience: Teaching Assistant, Department of Computing and Information Sciences, Oklahoma State University, August, 1974, to May, 1976; Programmer Analyst, Standard Oil Company (Indiana), Tulsa, Oklahoma, June, 1976, to July, 1977; Senior Systems Programmer, Bank of Oklahoma, Tulsa, Oklahoma, July, 1977, to November, 1978; Systems Engineer, Cities Service Company, Technology Center, Tulsa, Oklahoma, November, 1978, to June, 1982; Research Scientist and Research Supervisor, Amoco Production Company, Research Center, Tulsa, Oklahoma, June, 1982, to present; Member and past President of the student chapter of the Association for Computing Machinery.