CONCURRENCY CONTROL IN MULTIDATABASES

By

KALPANA HALLEGERE CHIKKANNA

Bachelor of engineering

in Computer Science and Engineering

Mysore University, Mandya

Karnataka, India

1988

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfilment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1994

OKLAHOMA STATE UNIVERSITY

CONCURRENCY CONTROL IN MULTIDATABASES

Thesis Approved:

Huizhu Lu

Thesis Adviser

Mitchell L Neb

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my main adviser, Dr. Huizhu Lu for her supervision, constructive guidance, inspiration, and friendship. My sincere appreciation extends to my other committee members Dr. K.M. George, and Dr. Mitch Neilsen, whose guidance, assistance, suggestions, and encouragement are invaluable.

I would like to give my special appreciation to my parents for their strong mental support, and encouragement at times of difficulty.

Finally, I would like to thank the staff members in the University Computer Center for helping me throughout my study.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

Multidatabase Architecture 1
Scope. 6
Objective. 7

II. LITERATURE SURVEY 8

Definitions of major terminologies 8
Past work. 10

III. ALGORITHMS FOR CC IN MULTIDATABASE 27

Method 27
Assumptions made 27
MDBS model with integrated scheduler 28
Requirements 29
Characteristics of the proposed method 29
CC schemes used by the proposed method 31
Timestamp Ordering Rule (TO Rule) 31
Thomas's Write Rule (TWR) 31
Pure Integrated Scheduler 32
Outline of transaction processing. 32
Pseudocode for the GTM 34
Pseudocode for the LTM 38
Algorithm for integrated scheduler 40
Pseudocode for transaction processing in LDBMS . .48

IV. PROOF OF CORRECTNESS. 53

V. SUMMARY AND FUTURE WORK. 58

A List of major approaches to CC in MDBS 58
Summary. 59
Future work. 60

BIBLIOGRAPHY. 61

LIST OF TABLES

Table Page

1. A list of major approaches to CC in MDBS 58

LIST OF FIGURES

Figure Page

1. Types of heterogeneities.....2

2. The overall architecture of an MDBS.....3

3. MDBS model with integrated scheduler.....28

CHAPTER I

INTRODUCTION

Multidatabase Architecture

A *MULTIDATABASE (MDBS)* is a system which provides uniform, integrated interface for retrieving data from preexisting, heterogeneous, distributed databases without violating their local autonomy. It allows users to manipulate data contained in various databases without modifying current database application and without migrating data to a new database. A database is considered to be *distributed* if it provides access to data located at multiple (local) sites in a network. It is considered to be *heterogeneous* if the local nodes are based on different technologies. The types of heterogeneities in the database systems can be classified according to the differences in DBMSs or differences in semantics of data (Figure 1) [SL90].

A general model of transaction processing in MDBSs shown in Figure 2 [Kim93] is characterized by the following features.

1. Interactions with a global or local database is conducted by user programs called *transactions.*
2. There are two types of transactions, *global* and *local.*
3. Global transactions are controlled by MDBS.
4. Local transactions are performed outside MDBS's control.

5. The MDBMS is built on top of Local Database Systems (LDBSs) to appear as an application of LDBSs. It consists of *Multidatabase Kernel (MDBK)* and *Local Transaction Managers (LTMs)* for each participant site.

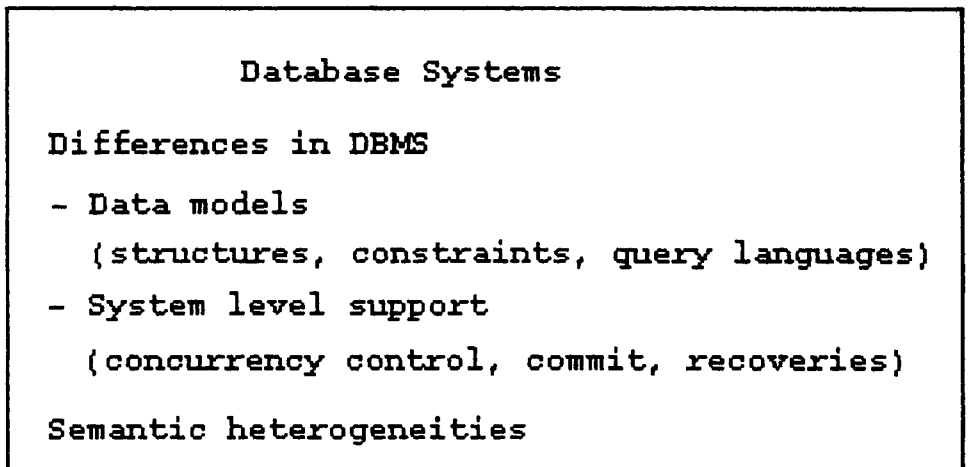

Figure 1. Types Of Heterogeneities [SL90]

6. The Global Data Manager (GDM) of MDBK is used to determine the location or locations of the data referenced by global transactions. It manages the multidatabase schema which is

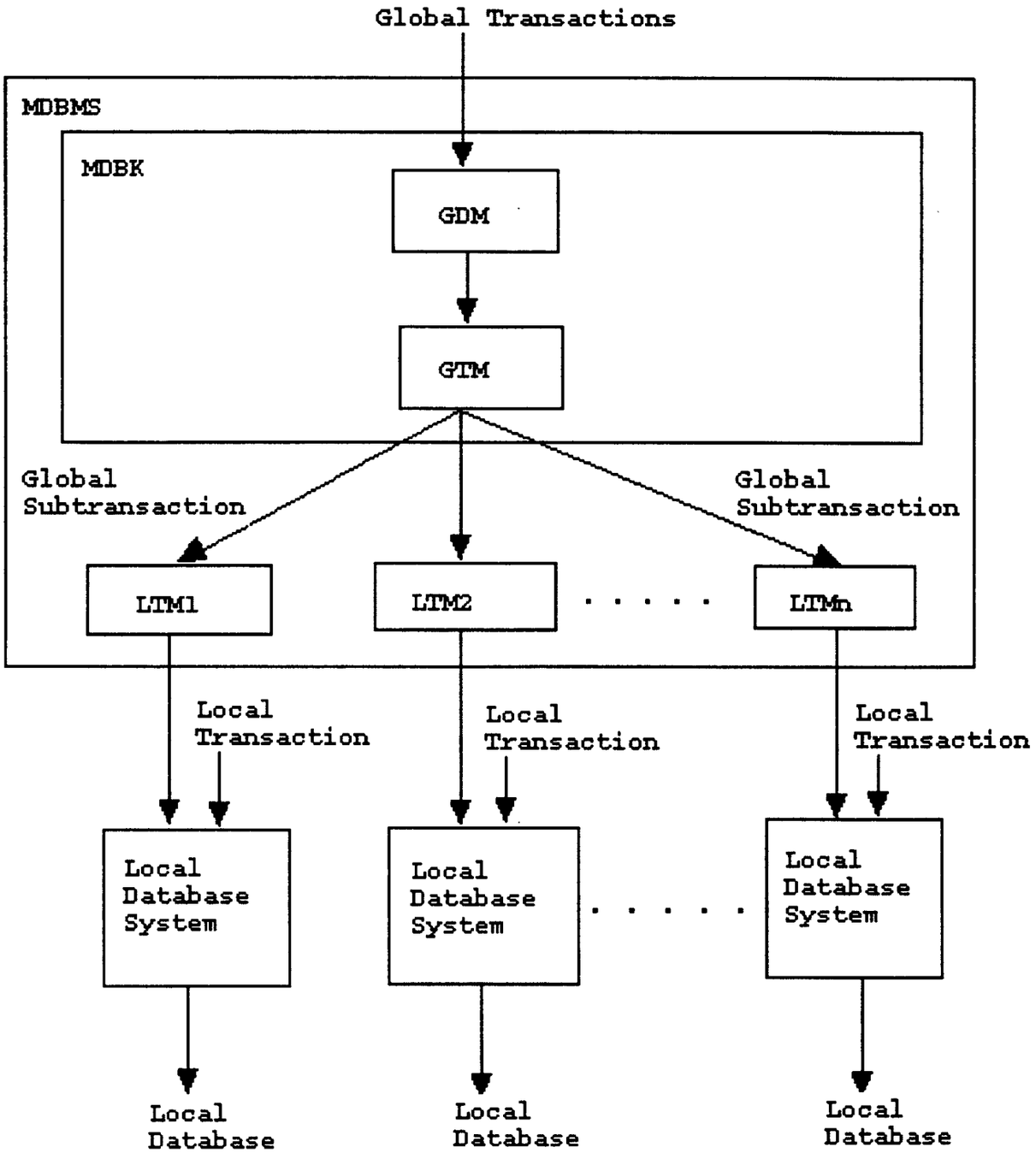

Figure 2. The Overall Architecture Of an MDBS [Kim93]

integrated from each local database schema, and decomposes global transactions into global subtransactions to distribute to appropriate LDBS for execution. All operations of a subtransaction access data managed by one LDBS; i.e., MDBK

does not directly manage any data other than the multidatabase schema.

7. Functions of *GTM* is two-fold: Concurrency control (or scheduling) to guarantee serialized execution of transactions by controlling the execution of subtransactions, commitment and recovery to achieve atomicity and durability of global transactions in the presence of failures. It allocates one *Local Transaction Manager (LTM)* for each of the sites referenced by the global transaction.

8. The LTM is the remote component of the MDBMS which runs directly on top of each LDBS. It receives operations of subtransactions from the GTM, submits them to the LDBMS, and sends the results to the GTM. Once an LTM is allocated, it is not deallocated until the transaction commits or aborts. An LTM has several responsibilities with respect to the execution of a global subtransaction.

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

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

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

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

9. Local DBMSs are not aware of each other, and if a local transaction is submitted to a local DBMS then no other local site is aware of that transaction. Local DBMSs behave as if

does not directly manage any data other than the multidatabase schema.

7. Functions of *GTM* is two-fold: Concurrency control (or scheduling) to guarantee serialized execution of transactions by controlling the execution of subtransactions, commitment and recovery to achieve atomicity and durability of global transactions in the presence of failures. It allocates one *Local Transaction Manager (LTM)* for each of the sites referenced by the global transaction.

8. The LTM is the remote component of the MDBMS which runs directly on top of each LDBS. It receives operations of subtransactions from the GTM, submits them to the LDBMS, and sends the results to the GTM. Once an LTM is allocated, it is not deallocated until the transaction commits or aborts. An LTM has several responsibilities with respect to the execution of a global subtransaction.

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

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

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

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

9. Local DBMSs are not aware of each other, and if a local transaction is submitted to a local DBMS then no other local site is aware of that transaction. Local DBMSs behave as if

MDBS does not exist according to the concept of local autonomy.

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

11. When a global transaction completes execution, the GTM instructs the LTMs allocated to the transaction, to commit the updates to the local databases. The MDBS uses a two-phase commit protocol in communication with the LTMs 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 is to be aborted, GTM instructs the LTMs to rollback the updates to the local databases.

Following are the criteria followed by the MDBS model for concurrency control.

1. The MDBS concurrency control mechanism guarantees a serializable global transaction execution.

2. The *Local Concurrency Control (LCC)* mechanism(s) guarantees a serializable local transaction execution.
3. No modifications are allowed to a local DBMS's software in order to deal with a MDBS.
4. No direct communications exist between local DBMSs.

Scope

Information is a key source in the daily operations of business, government, and academic organizations. Today, organizational information is frequently represented in computer databases. Due to the growing number of sophisticated users and organizations, the sharing of information resources increases. However, a multitude of systems usually means multiple access methods and user paradigms. Multidatabase systems give users a common interface to multiple databases, while minimizing the impact on existing operations. To improve the response of such system by allowing considerably large number of users access the databases concurrently without affecting the autonomy of component DBMSs is an important area of current research, as evidenced by the number of projects in both academia and industry. The trade press has also documented the need for user friendly global information sharing. The next level of computerization will be a distributed global systems that can share information from all participating sites. MDBSs are a key component of this advancing technology.

Objective

The objective of this thesis is to propose a method for concurrency control in MDBSs to retain the autonomy of component DBMSs, and serializability with an improved degree of concurrency.

This thesis has been organized as follows. Definitions of major terminologies used in the literature and literature survey are presented in Chapter II. Algorithms for concurrency control at various levels in the MDBS are presented in Chapter III. Later, in Chapter IV proof for the correctness of proposed method for concurrency control is presented. Finally, we present the conclusion in Chapter V.

CHAPTER II

LITERATURE SURVEY

Definitions Of major terminologies

Transaction: User program which allows interaction with the database [Ber87].

Scheduler: Program or a collection of programs that controls the concurrent execution of transactions [Ber87].

Transaction Manager (TM): Responsible for the interaction between the user and the transaction. It receives database and transaction operations issued by transactions and forwards them to the scheduler [Ber87].

Data Manager (DM): Responsible for the execution of various transaction operations to be serializable and recoverable [Ber87].

Serial execution: For every pair of transactions, all of the operations of one transaction execute before any of the operations of the other [Ber87].

Conflicting operations: Two operations are said to be conflicting if they both operate on the same data item and at least one of them is a write. Two operations might conflict each other directly or by another operation executed in between, indirectly [Ber87].

Serializability: An execution is serializable if it produces the same output and has the same effect on the database as some serial execution of the same transactions [Ber87].

Atomicity: Either all operations of the transaction are properly reflected or none are [SKS91].

Isolation: Each transaction assumes that it is executed alone in the system and the local DBMS guarantees that intermediate transaction results are hidden from other concurrently executed transactions [SKS91].

Consistency: Execution of transaction in isolation preserves the consistency of the database [SKS91].

Durability: The values changed by the transaction must persist after the transaction successfully completes [SKS91].

Execution Order: The order that each operation is executed [LE90].

Serialization order: Partial order of all operations in the execution [LE90].

Lock point: Time at which the transaction aquires locks for all data it needs [LE90].

Serialization point: A distinguished action that determines the execution order of the transaction in the schedule [LE90].

Prepared State: State of a transaction in which the subtransaction finishes all of its read and computation operations and has all of its updates stored in a stable storage (such transaction is ready to commit or abort according to a global decision) [LE90].

Past Work

Concurrency control in heterogeneous database systems has been studied in the context of an organization in which different departments are controlled by different DBMSs. Concurrency control requirements in such DBMSs is different from those in conventional homogeneous (distributed) database systems. Because, in the former case component DBMSs involved might be using different concurrency control techniques and they are often autonomous. This autonomy of component DBMSs is the key factor to be retained in maintaining global serializability which is the correctness criterion adopted in concurrency control techniques.

The possibility of having heterogeneous database system was thought of in mid 1980's and various problems involved are addressed in later years. Data accessibility is important for the successful operation of any corporation. Historically, however it has been difficult for individuals to locate & access data within different departments of their own organization. In many cases, data within different departments of their own organization is controlled by different DBMSs. Some DBMSs are better suited for scientific and engineering 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, research in the field to develop efficient method to access existing heterogeneous databases, integrating heterogeneous databases [Stan87], concurrency control in distributed databases [Moon87] increased.

MULTIDATABASE (MDBS) is one of those systems which provide a uniform, integrated interface for retrieving data from preexisting, heterogeneous, distributed databases. This allows the user to access data in multiple databases quickly and easily. A MDBS is a distributed system that acts as a front end to multiple, local DBMSs. The global system provides full database functionality and interacts with local DBMSs at their external user interface. The local DBMSs are autonomous (site autonomy). Each site independently determines what information it will share with the global system, what global requests it will service, when it will join MDBS and when it will stop participating in it. This places a large burden on global DBAs. Because of independence and the possibly large number of participating sites, global requirements and desirable global optimization are likely to conflict with local ones. The traditional concept of a transaction as short lived and atomic is unsuited to the MDBS environment. Because of local autonomy, global control does not include control of the actual data items. This is the factor which generates problems in concurrency control. MDBS transactions are relatively long-lived and often non-atomic.

When a DBMS involves DBSs which are heterogeneous, transaction processing among component DBSs takes vital importance. To ensure consistency each transaction must effectively run in isolation. This isolation can be achieved by some means of concurrency control.

Serializability is the most widely used correctness criterion in concurrency control. Autonomy has considerable effects on global serializability. Concurrency control schedules data access of concurrency control to be serializable. However, this requires the knowledge of all currently active transactions and the ability to control access to data items which is not normally possible with standard DBMSs. Moreover different local concurrency control schemes are adopted by different DBMSs. The global system has enough information to provide concurrency control for global transactions, but it does not have information about local transactions. Therefore, it can not provide total concurrency. Various types of autonomy affected by global transactions executed at local nodes are

. *Design autonomy*, refers to the ability of a component DBMS to choose its own design criteria (main cause for heterogeneity).

. *Communication autonomy*, refers to the ability of a component DBMS to decide whether to communicate with other component DBMSs.

. *Execution autonomy*, refers to the ability of a component DBMS to execute local operations without

interference from external operations ,and decide the order in which to execute external operations.

. *Association autonomy,* refers to the ability of a component DBMS to decide whether and how much to share its functionality and resources with others.

Therefore, MDBS must provide correctness at global level. For such approaches, the expectation is that the applications that use an MDBS environment will provide sufficient information to accurately specify the restrictions that need to be placed on the global transactions so that they can safely interact with concurrent autonomous local executions.

The MDBS approach [LMR90] assumes that the user needs to access multiple databases without the benefit of a global schema. An autonomous database should have data definition autonomy, including name independence, data duplication autonomy, data restructuring autonomy at the logical and physical level, and value type autonomy. This leads to a situation wherein, data in different databases may be redundant, and there might be discrepancies such as names, data structures and value types. Due to the heterogeneity involved, transaction processing among component database systems takes a vital importance. Desired autonomy and consistency are to be maintained when component DBMSs are forced to share a global database schema. This can be achieved by some means of concurrency control.

Problems of heterogeneous database integration, the principal user requirements together with implementation requirements were identified. A general model of MDBS based on a relational model was introduced [Stan87]. Effects of restarting the transactions on Concurrency Control (CC) after they are aborted due to invalid subtransactions were identified [Moon87]. The concept of serializability as the correctness criterion for CC was introduced [Wolf87]. The behavior and performance of two different CC algorithms *Two-Phase Locking (2PL) and Commit Timed Version (CTV)* were analyzed. 2PL is a pessimistic approach whereas CTV is an optimistic approach [NHE86]. Feasibility of HDDBMSs was analyzed by comparing its features with other Distributed Databases (DDBs) and a typical architecture for HDDBMS was suggested [Oxb87].

The proliferation of different DBMSs and advances in computer networking and communication led to increasing HDDBMS scenarios. The possibility of providing integrated access to the users in heterogeneous, distributed environment was addressed and a model *THE HD-DBMS* was proposed. The major approaches to data sharing and accessing from the primitive commercial file and database load/unload, PC download to common interfaces on top of existing DBMS were cited [Card87].

To maintain serializability, an order is imposed on the execution of transactions. The complexity of strict serializability was revisited [Ket87]. Mermaid, a front end

to distributed, heterogeneous databases was introduced. This provides an integrated access to systems which differ in technologies such as operating systems, networks etc. This model was found to be less powerful in finding the source(s) of errors and suggesting potential cures [Temp87].

The *Amoco Distributed Database System (ADDS)* is an MDBS model. The CC requirements for the model were discussed and several solutions were proposed [Thom87]. The algorithm proposed was found to reduce concurrency in multiple transaction execution. Several CC mechanisms proposed in next few years were based on roll back and blocking operations. The performance of these two operations of the concurrency control mechanisms was analyzed and found that neither of them is consistently better for all workloads, rather they are workload sensitive [Kum87]. The feasibility of HDDBMSs and the concepts of serializability and local autonomy are discussed in the context of CC and the extent of strictness for both concepts was suggested [ELHMRS87].

Research was conducted on the concept of locking to introduce an improved method for CC and 2PL was found to be the suitable locking technique for HDDBMSs [CM87]. Special type of scheduler called cautious scheduler, which never resorts to rollbacks for the purpose of CC was investigated. This is based on the assumption that transactions predeclare their read and write sets on their arrival [IKK88]. Algorithms for reducing rollbacks, tolerating higher degree of conflict among transaction and allowing more concurrency

at the update phase are presented [BKh87]. These algorithms were found to be unsuitable for MDBS environment since rollback is not completely prevented and due to the requirement of the pre declaration of read and write sets by transactions.

A formal model of data update in MDBS environment was developed and a theory of concurrency control in such environment was presented [BS88a]. The author formulated a correctness condition for CC mechanism and proposed a protocol that allows concurrent execution of a set of global transactions in presence of local ones. This protocol ensures the consistency of MDBS and deadlock freedom and was proved to be correct. But this does not exploit maximum concurrency in MDBS environment.

The notion of heterogeneous databases has been characterized as the inevitable consequence of replacing the traditional data processing practice with modern database management. The current problems and future issues connected with the great proliferation and overwhelming use of HDBSs and their DBMSs were articulated. A taxonomy of DBMS solutions to the problems and issues of heterogeneous databases was presented and future research work needed was discussed based on the taxonomy [HK89].

Local autonomy of component database systems in HDDBMSs has considerable effect on the Global Concurrency Control. In order to provide a correct environment for global updates, in concurrent execution of global transactions, *Global*

Concurrency Controller (GCC) must be provided. Several CC protocols have been proposed [AGMS87] [Su87] [BS88b] [EH88] [Pu88]. None of these protocols estimated the depth of the difficulty of problems. The difficulties in maintaining serializability of global executions within the HDDBMS which were found to be resulting from the differences between the serialization orders and execution orders and autonomies of local databases were considered. The difficulties in designing GCC algorithms and the unsuitability of serializability as the correctness criterion for GCC were discussed. Also it was concluded that it is impossible to design a good GCC algorithm which has a high degree of concurrency and does not violate local autonomy, as long as serializability is used as correctness criterion [DELO89].

The GCC algorithm used in superdatabases [Pu88] was found to be good for the hierarchical composition of HDDBSs and also it provides high degree of concurrency. But the autonomity of component DBMSs is not ensured. Distributed cycle detection algorithm proposed [Su87] provides high degree of concurrency for global transactions. But it violates local autonomy and execution autonomy.

A method for integrated CC and recovery, applicable to heterogeneous multidatabase systems was proposed. A prototype system called HERMES which is a MDBS to run global transactions distributed over SYBASE and INGRES was proposed. They assume that each LDBS adopts strict 2PL and The responsibility for two-phase local commitment and recovery of

the *prepared site* at participants is taken over by an entity called *2PC agent*. A 2PC agent maintains a log in a separate stable storage to monitor the status of subtransactions and simulate 2PC in presence of LDBS failures. Main importance of this method is in preserving global serializability in presence of certain class of participant-related failures [WV90].

Hierarchical CC has been proposed as one possible approach for MDBSs. A new GCC algorithm based on this approach was presented. Global serializability is used as correctness criterion for GCC. However, to apply this approach some restrictions have to be imposed on the LCC algorithms. In a hierarchical CC approach, LCCs control the execution of local transactions and global subtransactions to retain the serializability of local histories, while GCC controls the execution of the global subtransactions to maintain the compatibility of the subtransaction serialization orders. Based on this property, the hierarchical CC approach was formalized and its correctness was proved [LE90]. However it was identified that the hierarchical approach is not suitable for all MDBS environments due to the restrictions imposed on LCCs. MDBSs are also given the name interoperable database systems. Interoperability of MDBSs was discussed to explain the need for a centralized control of DBMS in heterogeneous environment [LMR90]. To overcome the difficulties caused by

the differences due to the heterogeneity of component DBMSs, a centralized control of DBMS was managed giving applications the illusion of being the sole user of the data while providing overall consistency, privacy and efficiency. Later arose the need for shared access of data across these multiple, autonomous databases. This needed an extension in the techniques for CC to retain local autonomy of component DBMSs when multiple transactions need to update same data in the global schema representing all database systems.

Several HDDBMS models were introduced to provide an effective means of sharing data in an organization with diverse data systems. *DATAPLEX* is one of those HDDBMSs developed by General Motors Research Laboratories. 2PL is adopted as an approach to CC in this model. Subsequent versions of this model were expected to provide the capabilities of distributed update, multiple copy synchronization etc. which are not available in the current version [Chung90].

A method for reliable transaction management in MDBSs was suggested and a scheduler algorithm which assures the global database consistency was proposed. The scheduler uses 2PL method for CC. The scheduler has got no control over local locks. However, keeps track of global transaction's requests for local locks through the use of global lock mechanism. Each global data item has a global lock associated with it. This method has been proved to be correct in maintaining consistency and local autonomy at the cost

of degree of concurrency [BST90].

A simulated 2PC and recovery algorithm based on MDB-serializability as the correctness criterion was proposed [Ba90]. In this method, the MDBK maintains a global log which consists of *GT termination log, GT active log, GST completion log, Intermediate GST result log, and GST ready to commit log.* A STUB (which corresponds to a server) does not maintain any log. It is assumed that an LDBS produces serializable and strict histories. Since there is no log associated with STUB, a STUB does not have any knowledge of subtransactions when its underlying LDBS fails. It must connect to the MDBK to receive operations of the subtransactions to be resubmitted. If the MDBK has failed at this point, the LDBS must wait until the MDBK recovers even though there may be no subtransactions that must be resubmitted.

HYDRO is another model of HDDBMS proposed [PRR91]. This adopted serializability as the correctness criterion for CC. It was shown that global serializability and atomic commit can be attained in a HDDBMS in which full local autonomy is provided to the local DBMSs. This has been modeled for a network environment. A local HYDRO server for each autonomous LDBS was introduced to support 2PC. Local transactions are routed to a HYDRO server rather than submitted to an LDBS directly. In general, however it is very difficult to accomplish this because the server must provide the same LDBS interface to all existing applications. They achieve atomic commitment by simulating 2PC. A HYDRO server must keep the

before values of each update operations of a global subtransaction in a log, which may not be easily attainable when the operations are expressed in SQL.

Since preserving local autonomy is a crucial factor in MDBSs, possibilities of violations of various types of autonomies are discussed in developing a failure-resilient transaction management system in MDBSs [SKS91].It is however, not possible to guarantee serializability in a MDBS using conventional approach. Several seemingly different solutions have been proposed using nested transactions paradigm. A simple model which is used to develop a number of new MDBS schedules using existing theories and concepts has been proposed [Deac91].

Heterogeneous databases and serializable schedules are contradictory in practice. Mechanisms that guarantee fully serializable schedules impose strong constraints and they are an overkill. Much simpler and unrestricted mechanisms can provide the correctness that is needed [GM91].

Interdatabase dependencies in MDBS environment play an important role in updating interdependent data. A new correctness criterion, quasi-serializability for maintaining transaction consistency in MDBSs was introduced [DEK91]. But it was identified that not all aspects of transaction consistency are ensured by this approach.

CC requirements in advanced database applications are different from those in conventional database applications. They need non-serializable support among the group of users

whose transactions are long lived and integrate CC. This led to the relaxation of serializability [BKa91]. This increased the efficiency of 2PL which is identified as the suitable CC technique for HDDBMSs.

The notion of serializability has been traditionally accepted as the correctness criterion in database systems. However, in HDDBMS environment ensuring serializability is a difficult task due to the desire of preserving the local autonomy of the participating local DBSs. A new correctness criterion, two-level serializability (2LSR) introduced was found to ensure serializability, but degree of concurrency is reduced due to excessive constraints on concurrent execution [MRBKS91].

Although many researchers in distributed database area perceive that the only practical way to construct a distributed database system from already existing heterogeneous database system is to integrate them by guaranteeing local site autonomy, an efficient update synchronization scheme has not been developed so far. A new concurrency control scheme that guarantees both serializability of concurrent execution and site autonomy for HDDBMSs has been proposed [KM91]. The proposed CC scheme was found to reduce the level of concurrency.

The problems in Multidatabase recovery were addressed. To assure that multidatabase recovery preserves the consistency of a multidatabase system, a multidatabase recoverability requirement was introduced. i.e., only if each

LDBS produces strict and serializable history and the MDBK can have an exclusive access to the LDBS after restart, but before it becomes available for local users. Also, a recovery mechanism that takes advantage of the local recovery in participating database systems by minimizing the replication of recovery tasks was described [Geo91].

In an MDBS environment, the traditional transaction model has been found to be too restrictive. An extended transaction model, in which some of the requirements of transactions, such as isolation, atomicity etc. are relaxed was proposed. To provide access to multiple heterogeneous hardware or software systems, distributed operations language (DOL) was used. This approach is based on providing common communication and data exchange protocol and uses local access manager to protect the autonomy of member software systems [ARNS91]. Rigorous transaction management schemes are introduced to achieve global serializability in MDBSs. These schemes seem to preserve local autonomy and assure global serializability, but the degree of concurrency is low, since no two global transactions can be executed in the same two LDBSs concurrently [Briet91].

As a number of diverse, heterogeneous types of DBMSs are employed within single organization, the need to integrate those systems is stringent to have an efficient and transparent access to remote sites. The integrated access control placed at the global data manager level in each site is used as security enforcement in an HDDBMS [KM92].

An approach to schema integration in a HDDBMS design was described in context of a prototype MDAS which acts as a front-end to multiple local DBMSs which continue to perform all local data management and processing [DP92].

Algorithms for scheduling of distributed transactions in a heterogeneous multidatabase were presented. The algorithms of prepare certification and commit certification protect against serialization errors called global view distortion and local view distortion. The main advantage of this method as compared to other known solutions is that it is totally decentralized [VW92].

The problems in ensuring atomicity of global transactions in multidatabase environment were addressed. It was shown that the atomicity requirement of global transactions and the autonomy constraints on the design of the MDBS software are mutually conflicting goals. A global commit protocol that ensures atomicity without violating autonomy of the local DBMSs was developed. The protocol needs restrictions to be placed on the data items accessed by global transactions, the execution of global transactions and the requirements of local schedulers such that local schedules are serializable in the MDBS view if the global commit protocol is used to ensure atomicity of global transactions [MRBKS92b].

Ticket based transaction management in Multidatabases was introduced. Several different methods based on this approach were proposed. In Optimistic Ticket Method (OTM),

direct conflicts between multidatabase transactions are created at each LDBS to determine the relative serialization order of their subtransactions. Conservative Ticket Method (CTM) does not require global serialization testing and eliminates global restarts due to failed validation. This might allow a higher throughput than OTM. Implicit Ticket Method (ITM) works only if the participating LDBSs disallow schedules in which transaction execution and serialization orders are not analogous. ITM can process any number of transactions concurrently, even if they have conflicting and concurrent subtransactions at multiple sites [GRS92].

A fully decentralized global concurrency control method in which the concurrency control decision for controlling global transactions can be made at each site, based on the information that is locally available. This method uses a top down approach to enforce the same serialization order at all sites a global transaction is executed. This method uses forced local conflicts to prevent unacceptable local schedules while assuming deadlock free execution [BRG92].

An extension of traditional 2PL protocol for CC, a deadlock detection technique based on the use of potential-wait-for graph to detect deadlock among multidatabase transactions, and a commitment protocol called a resubmit log method for recovery from multidatabase transaction failures and system failures were presented. Also, it was proved that all these methods can be implemented without requiring any changes to existing database systems [Kim93].

A framework for constructing analytical performance models of concurrent B-tree algorithms. The models can predict the response time and maximum throughput. Variety of locking algorithms including naive lock-coupling, optimistic descent, 2PL, and link-type (Lehman-Yao) algorithms are analyzed. The analyses are validated by simulations of the algorithms on actual B-trees, as well as by simulations done by other researchers. Link-type algorithm was found to be the best algorithm for B-tree concurrency control, allowing levels of concurrency that are significantly higher than is possible with the other algorithms. Such high levels of concurrency are allowed because the only modified nodes that are actually W-locked (write-lock) are the one that are modified and because the locks are held for as short time as possible. Also, recovery algorithms for a concurrent B-tree used in a database are proposed. Naive recovery algorithm holds locks on all nodes that are modified until the transaction that issued the operation either commits or aborts. The leaf-only recovery algorithm holds locks on the leaf nodes and releases locks on the upper level nodes. The analysis shows that leaf-only recovery algorithm is significantly better than naive recovery algorithm. The suitability of B-tree algorithms for MDBS environment is not yet discussed [JS93].

CHAPTER III

ALGORITHMS FOR CC IN MDBS

Method

An additional level is introduced on the top of each local DBMS in the basic MDBS model (Figure 2) so that all transactions (local and global) are scheduled at this level. The scheduling algorithm used at this level is based on a pure integrated scheduler which is a combination of Thomas's Write Rule (TWR) and Timestamp ordering (TO). Timestamps obtained by transactions reflect their serialization orders. Figure 3. shows the MDBS model with an additional level (Integrated Scheduler).

Assumptions Made

1. No data in any LDB is replicated in any other LDB.
2. Single subtransaction per global transaction on an LDBS.
3. A transaction enters its Prepared State when it completes the execution of its database operations and leaves this state when it is committed or aborted.
4. Since total autonomy means lack of cooperation and communication and hence total isolation, some less extreme notions of LDBSs autonomy are proposed in the literature.

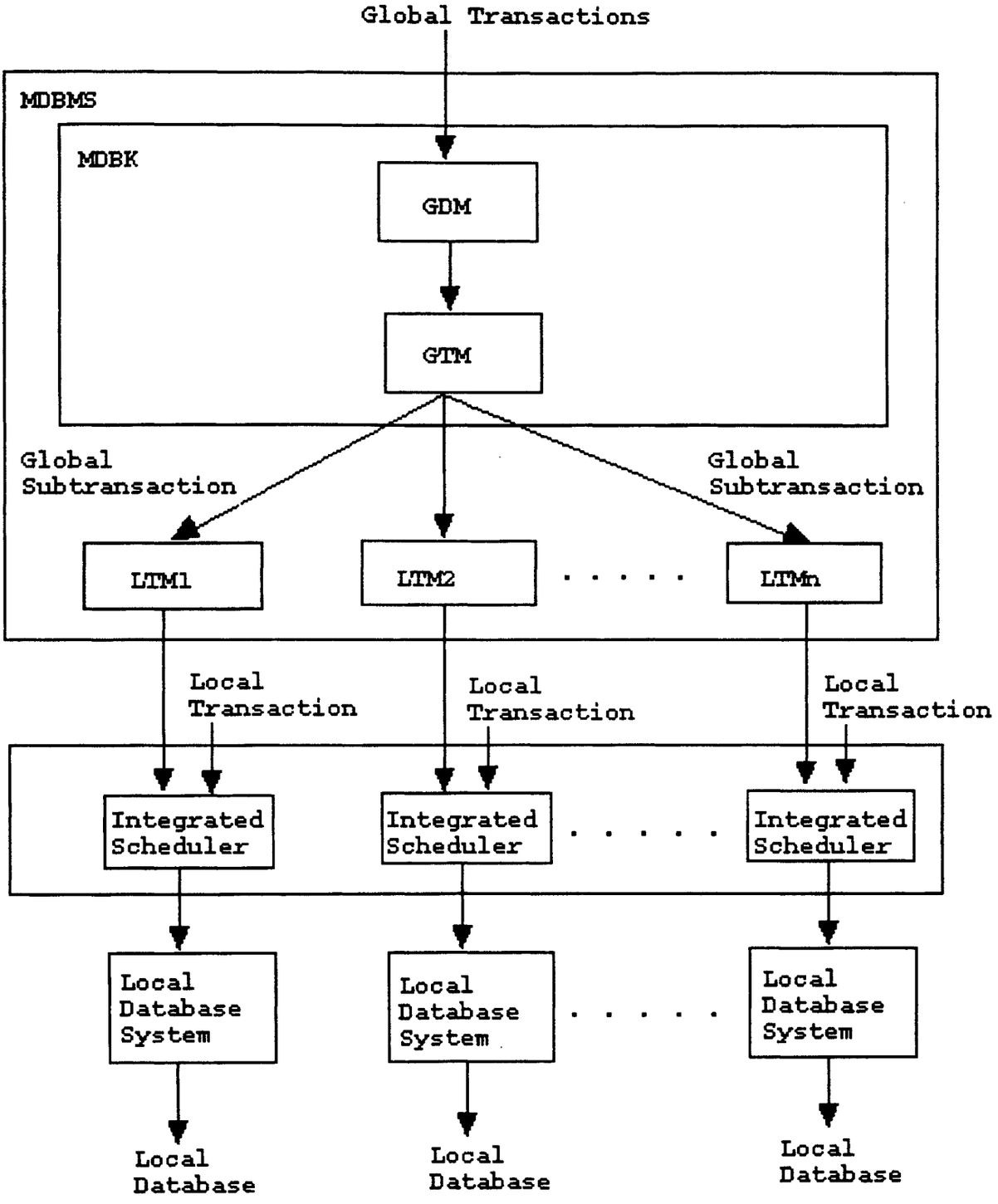

Figure 3. MDBS model with Integrated Scheduler

Requirements

Following are the requirements to be satisfied by the proposed algorithm.

1. Autonomy of LDBSs

Implementation of an MDBS must not require any changes to existing database, applications, and LDBS itself.

2. ACID property

The traditional properties of transactions, namely, atomicity, consistency, isolation and durability must be preserved.

3. Deadlock

The algorithm should be free from deadlock.

4. Serializability

The algorithm should produce serializable schedules.

Characteristics of the proposed method for CC in MDBS

1. By scheduling the global transactions such that their execution order and the serialization order at each site are identical, global serializability is assured.
{Serialization order and Execution order of two global transactions executing at the same site might differ due to

an indirect serialization order introduced between the global transactions by the local transactions at that site [Ber87].}
Global Concurrency Controller (GCC) can maintain a certain serialization order by controlling the execution order of global operations. Local Concurrency Controllers (LCCs) in each LDBMS maintain local serializability at the site, thus guaranteeing identical serialization and execution orders.
2. By scheduling the operations belonging to local and global transactions at the integrated level, unnecessary delay in scheduling the transactions and unnecessary abortions of transactions due to improper scheduling of local and global transactions which results in nonserializable execution of transactions are eliminated. Operations are either immediately scheduled or rejected, thus improving the degree of concurrency. The Number of aborted transactions is reduced due to the scheduling mechanism used at the integrated level. The operation is rejected only if it is a write (read) and some other read (write) operation with greater timestamp has already been scheduled.
3. Local autonomy of individual DBMSs is assured since the addition of a integrated scheduler will not affect the structure of existing DBMSs.

CC schemes used by the proposed method

Timestamp Ordering Rule (TO rule) [Ber87]

A TO scheduler orders conflicting operations according to their timestamps. i.e., if $p_i[X]$ and $q_j[X]$ are two conflicting operations on data item X, belonging to transactions T_i and T_j, then the DM processes $p_i[X]$ before $q_j[X]$ if and only if timestamp(T_i) < timestamp(T_j). If $p_i[X]$ arrives too late, it is rejected and T_i is aborted. When T_i is resubmitted, it must be assigned a large timestamp, large enough so that its operations are less likely to be rejected during its second execution.

Thomas's Write Rule (TWR) [Ber87]

It is basically a Write-Write (WW) synchronizer. It never rejects or delays any operations. When it receives a write that has arrived too late insofar as the TO rule is concerned, it simply ignores the write, but reports its successful completion to the transaction manager. The assumption made when this method is introduced is that, processing a sequence of writes in timestamp order produces the same result as processing the single write with maximum timestamp, thus late operations can be ignored.

Pure Integrated Scheduler [Ber87]

A pure integrated scheduler is a combination of timestamp Ordering (TO) for Read-Write (RW) synchronization and Thomas's Write Rule (TWR) for Write-Write (WW) synchronization. The main importance of this scheduler is that, it avoids unnecessary rejection of writes. If T_i and T_j are two transactions with conflicting operations $r_i[x]$ and $w_j[X]$ or $w_i[X]$ and $r_j[X]$ or $w_i[X]$ and $w_j[X]$ and $ts(T_i)$ and $ts(T_j)$ are the timestamps assigned to transactions T_i and T_j respectively, the scheduling is performed as follows.

1. It schedules $r_i[X]$ provided that for all $w_j[X]$ that have already scheduled, the condition $ts(T_i)>ts(T_j)$ is true. Otherwise, it rejects $r_i[X]$.
2. It rejects $w_i[X]$ if it has already scheduled some $r_j[X]$ with $ts(T_j)>ts(T_i)$. Otherwise, if it has scheduled some $w_j[X]$ with $ts(T_j)>ts(T_i)$, it ignores $w_i[X]$ (according to TWR). Otherwise, it processes $w_i[X]$ normally.

Outline of transaction processing

1. The GTM decomposes every global transaction submitted into as many global subtransactions as the number of sites in which the transaction has to be executed, each of which accesses only one LDB. The GTM maintains a global log to record information about global transactions. The global log is used in case of failure.

2. The GCC determines an order among the global transactions so that their serialization orders are compatible in all local sites they are executed, and allocates LTM to each subtransaction in that order.

3. The LTM converts the global read/writes to the language understandable by the local DBMS at that site and forwards them to the integrated scheduler. Also, the LTM keeps a log to record information about each subtransaction submitted, along with the result of subtransaction execution which is passed on to the GTM once the transaction in globally committed after failure.

4. The integrated scheduler assigns unique timestamps to the transactions submitted, schedules the transaction operations based on the timestamp ordering, and forwards them to local DBMS for processing.

5. The LCC at each local DBMS ensures local serializability.

6. A local transaction is allowed to commit in the normal fashion once its last operation is executed and its effects are made permanent.

7. A global subtransaction executed at the local site is allowed to enter *prepared-to-commit* state after receiving READY instruction from the GTM which acts as the coordinator and remains in this state till the coordinator issues COMMIT or ABORT instruction for global commit or abort. Communication between the coordinator and local sites is accomplished through the LTMs.

8. If an LDBS stops functioning, then all uncommitted transactions at that site are aborted, MDBK is notified of the site failure and the LDBS is restarted by getting a list of global transactions whose subtransactions have entered *prepared-to-commit* state, and not yet committed. All the transactions in the list are committed and effects are made permanent.

9. If the MDBK stops functioning all uncommitted global transactions are marked as 'aborted', and 'ABORT' message is sent to all servers allocated to global transactions to cancel the effects of transaction execution in the LDBS.

Pseudocode for the GTM

```
do forever
begin
     On receiving a global transaction do
     begin
          Assign a timestamp;
          Decompose into subtransactions;
          Allocate one LTM for each subtransaction if
          available
          else Add to the queue of subtransactions waiting
          for LTM;
          Record information about each subtransaction in the
          log;
     end
```

```
On receiving a message from one or more LTM(s) allocated
to a transaction do
begin
     If message = 'SERIALIZED' then {Subtransaction
     execution completed.}
     begin
          Check if the relative serialization order is
          the same in all participating sites;
          If yes, then
          begin
               Send 'READY' message to all LTMs
               allocated to the transaction;
               Record the message in the log;
               Go to WAIT;
          end
          else
          begin
               Send 'ABORT' message to all LTMs
               allocated to the transaction;
               Record the message in the log;
               Go to WAIT;
          end
     end
     IF message = 'YES' from all LTMs allocated to the
     transaction then {Transaction committed in all
     participating sites within the time out period.}
```

```
begin
     Send 'COMMIT' message to all LTMs allocated to
     the transaction;
     Go to WAIT;
end
else
If message='NO' from at least one LTM allocated to
the global transaction then {Transaction not yet
committed in at least one participating site within
a specified time period.}
begin
     Send 'ABORT' message to all LTMs allocated to
     the transaction;
     Record the message in the log;
     Go to WAIT;
end
If message = 'REJECT' then
begin
     Send 'ABORT' message to all LTMs allocated to
     the transaction;
     Record the message in the log;
     Go to WAIT;
end
WAIT: Wait for the message from LTMs
If message='COMMITTED' from all LTMs then
```

```
begin
     Make the effects of transaction execution
     permanent in the global database;
     Record the message in the log;
     Deallocate all LTMs allocated to the
     transaction;
     Allocate LTMs to the subtransactions waiting
     on LTM;
end
If message = 'ABORTED' from at least one LTM then
begin
     Deallocate all LTMs allocated to the
     transaction;
     Add the transaction to the restart queue;
     Allocate LTMs to the subtransactions waiting
     on LTM;
     Record information about new subtransactions
     in the log;
end
If message = 'SITE FAILURE' from LDBS(s) then
begin
For all global transactions
     Mark the subtransactions which have not voted
     'YES' or 'NO' as aborted in the log;
     Receive a list of transactions from the log
     maintained by the LTMs;
```

```
                Mark the transactions as 'committed' or
                'aborted' in the log;
                Build a new list of transactions to be
                resubmitted;
                Send the new list to the servers allocated;
          end
          If the MDBK stops functioning then
          begin
          For each global transaction Gi which did not
          terminate
                Mark the transaction as aborted in the log;
                Send 'ABORT' message to all LTMs allocated;
          end
     end(do)
end(do forever)
```

Pseudocode for the LTM

```
do forever
begin
     On receiving a subtransaction do
     begin
          Decompose the subtransaction into atomic operations
          in the language understandable by the LDBMSs;
          Enqueue the atomic operations to the queue of
          operations to be scheduled at the integrated level;
     end
```

```
On receiving a message from the LDBMSs or the integrated
scheduler or the GTM do
begin
     If message = ('READY' or 'COMMIT' or 'ABORT') from
     the GTM then
     begin
          Record the message in the log;
          Forward the message to the LDBMS;
     end
     If message = ('SERIALIZED' or 'YES' or 'NO' or
     'ABORTED' or 'REJECT') from the LDBMS or integrated
     scheduler then
     begin
          Record the message in the log;
          Forward the message to the GTM;
     end
     else
     begin
          If message = 'COMMITTED' then
          begin
               Record message and the result of
               execution in the log;
               Forward the message and result of
               transaction execution to the GTM;
          end
     end
     else
```

```
        begin
            If message = 'COMPLETE' from the integrated
            scheduler then
            begin
                Mark the operation of corresponding
                subtransaction as completed;
            end
        end
    end(do)
end(do forever)
```

Algorithm for Integrated Scheduler

Data Structures used by the integrated scheduler

ARR_QUEUE[]: Queue of operations submitted to the integrated level by the LTM. Each element in the queue is a structure of two components TRANS_ID and OP.

TSARRAY[]: Array of timestamps assigned to various transactions whose index is TRANS_ID.

UN_SCHED_QUEUE[]: Queue of operations belonging to various transactions to be scheduled to execute at that site. Each element in this queue is a structure of three components, TRANS_ID, OP, and TS.

SCHED_QUEUE[]: Queue of operations belonging to various transactions that are scheduled on the order of their timestamps. Each element in this queue is a structure of three components, TRANS_ID, OP, and TS.

OPER: Temporary structure variable with the same structure as the elements of SCHED_QUEUE[].

REAR: Index to SCHED_QUEUE[], points to the rear end of the queue.

MAX_SCHED_WR[X]: Maximum of all timestamps of the transactions that are scheduled so far to write data item X.

MAX_SCHED_RD[X]: Maximum of all timestamps of the transactions that are scheduled so far to read data item X.

RW_STATUS: Flag used for synchronization between two synchronizer.

```
Procedure IntegratedSchedule(ARR_QUEUE[])
{The procedure IntegratedSchedule assigns a unique timestamp
to each of the transactions in the ARR_QUEUE[] when it
receives transaction's 'BEGIN' operation and schedules the
transaction operations such that their timestamp ordering
reflects their serialization order.}
```

Input Set of operations belonging to local transaction and global subtransactions to be executed at that site.

Output Integrated schedule containing interleaved operations belonging to local transaction and global subtransactions.

step 0

```
{Initialize data structures.}
I <- 0; J <- 0; MAX_SCHED_RD[X] <- 0; MAX_SCHED_WR[X] <-
0; SCHED_QUEUE <-ø; UN_SCHED_QUEUE <-ø; TSARRAY[] <- ø;
REAR <- 0;
```

step 1

```
{Dequeue transaction operations, assign timestamp, and
enqueue them for scheduling.}
while notempty(ARR_QUEUE[])
begin
     If OP(ARR_QUEUE[J]) = 'BEGIN' then
          TSARRAY(TRANS_ID(ARR_QUEUE[J])) <- timestamp;
     OP(OPER) <- OP(ARR_QUEUE[J];
     TRANS_ID(OPER) <- TRANS_ID(ARR_QUEUE[J]);
     TS(OPER) <- TSARRAY[TRANS_ID(ARR_QUEUE[J]];
     UN_SCHED_QUEUE[I] <- OPER;
     I <- I+1;
     J <- J+1;
end(while)
```

step 2

```
{Take a transaction operation from the head of the
queue, read the timestamp, and schedule accordingly.}

I <- 0;
while notempty(UN_SCHED_QUEUE[])
begin
     OPER <- UN_SCHED_QUEUE[I];
     If OP(OPER) = 'WR[X]' then
     begin
          WW_SYNCHRONIZER(OPER);
     end
     else
     begin
          If OP(OPER) = 'RD[X]' then
          begin
               RW_SYNCHRONIZER(OPER);
          end
          else
          begin
               If (OP(OPER) = 'BEGIN') OR (OP(OPER) =
               'END') OR (OP(OPER) = 'COMMIT') OR
               (OP(OPER) = 'ABORT') then
               begin
                    SCHEDULE(OPER);
               end
          end
```

```
        end
    I <- I+1;
    end (while)
```

```
Procedure WW_SYNCHRONIZER(OPER)[Ber87]
{call Read_Write synchronizer to decide whether the operation
has to be rejected, ignored or scheduled to maintain
consistency.}
    RW_STATUS <- RW_SYNCHRONIZER(OPER)
    If not(RW_STATUS) then
    begin
        REJECT(OPER);
    end
    else
    begin
        If (TS(OPER) < MAX_SCHED_WR[X]) then
        begin
            IGNORE(OPER);
        end
        else
        begin
            SCHEDULE(OPER);
        end
    end
```

```
Procedure RW_SYNCHRONIZER(OPER)[Ber87]
{If the timestamp of the write operation to be scheduled is
greater than the maximum of all timestamps of operations
scheduled so far to read data item X, then set RW_STATUS to
TRUE, otherwise set to FALSE.}

     If (OP(OPER) = 'WR[X]') then
     begin
          If TS(OPER) > MAX_SCHED_RD[X] then
          begin
               RW_STATUS <- TRUE;
          end
          else
          begin
               RW_STATUS <- FALSE;
          end
          RETURN(RW_STATUS);
     end
     else
     begin
          {The read operation is scheduled if its timestamp
          is greater than the maximum of all time stamps of
          transactions scheduled so far to write data item X,
          otherwise it is rejected.}
          If TS(OPER) > MAX_SCHED_WR[X] then
          begin
               SCHEDULE(OPER);
```

```
        end
        else
        begin
            REJECT(OPER);
        end
    end

PROCEDURE SCHEDULE(OPER)
{The procedure SCHEDULE(OPER) schedules the database and
transaction operation and enqueues to SCHED_QUEUE[].}

    IF OP(OPER) == 'RD[X]' then
    begin
        MAX_SCHED_RD[X] <- TS(OPER);

    end
    else
    begin
        IF OP(OPER) == 'WR[X]' then
        begin
            MAX_SCHED_WR[X] <- TS(OPER);
        end
    end
    SCHED_QUEUE[REAR] <- OPER;
    REAR <- REAR + 1;
```

```
PROCEDURE REJECT(OPER)
{The procedure REJECT(OPER) is called when an operation
arrives late. This procedure cancels the effects of the
transaction execution and places the transaction on the
restart queue for re submission.}

     IF the operation belongs to local transaction the
     begin
          Send 'REJECT' message to the LDBMS to cancel the
          effects of the operations belonging to that
          transaction;
          Mark the transaction TRANS_ID(OPER) as aborted;
     end
     else
     begin
          Send 'REJECT' message to the LTM  allocated to
          TRANS_ID(OPER) and to the LDBMS at the site;
     end

PROCEDURE IGNORE(OPER)
{The procedure IGNORE(OPER) is called when an operation is
arrived late and it is a WRITE. This procedure notifies the
completion of the operation to the concerned transaction
manager without actually performing the operation.}
     IF the operation belongs to a local transaction then
     begin
```

```
        Notify the LDBMS about the completion of operation
        without actually forwarding it to the data manager;
    end
    else
    begin
        Send 'COMPLETE' message to the LTM;
        {Notify the LTM allocated to that transaction about
        the completion of late operation.}
    end
```

Pseudocode for transaction processing in LDBMS

```
do forever
begin
    On receiving an integrated schedule from the integrated
    scheduler do
    begin
        Pick the operation at the head of the queue
        containing operations scheduled for processing;
        Check if there is a conflicting operation that has
        already been dispatched for processing and not yet
        completed;
        If yes, wait for the acknowledgment from the DM for
        the completion of conflicting operation;
        Dispatch the operation to the DM for processing;
    end
```

```
On processing the last operation of a transaction do
begin
     If the operation belongs to a local transaction
     then
     begin
          Make the effects of transaction execution
          permanent in the local database;
          Commit the transaction and report the
          transaction manager at the site about the
          completion of transaction;
     end
     else
     begin
          Allow the transaction to enter the 'prepared-
          to-commit' state;
          Send 'SERIALIZED' message to the GTM through
          LTM;
          Go to WAIT;
     end
end
On receiving a message from the LTM or the integrated
scheduler do
begin
     If message = 'REJECT' then
     begin
          If operation rejected belongs to local
          transaction then
```

```
        begin
              Cancel the effects of transaction
              execution;
              Add the transaction to the restart queue
              at the site;
        end
        else
        begin
              Cancel the effects of subtransaction
              execution;
              Mark the transaction as aborted;

        end
end
If message = 'READY' then
begin
        Check if the subtransaction has entered the
        'prepared-to-commit' state;
        If yes, then
        begin
              Send 'YES' message to the LTM;
              Go to WAIT;
        end
        else
        begin
              Send 'NO' message to the LTM;
              Go to WAIT;
```

```
        end
    end
    WAIT: Wait for the message from the GTM;
    If message = 'COMMIT' then
    begin
        Make the effects of transaction execution
        permanent in the local database;
        Forward the results of transaction execution
        and the message 'COMMITTED' to the LTM;
    end
    If message = 'ABORT' then
    begin
        Cancel the effects of subtransaction execution
        in the local database;
        Send 'ABORTED' message to the LTM;
    end
end(do)
If LDBS fails then
begin
    Abort the uncommitted local transactions and
    subtransactions;
    Notify MDBK of the failure;
    Send a list of transactions which have voted 'YES'
    and not yet committed to the MDBK;
    Get a list of transactions to be resubmitted from
    the MDBK;
    Commit the transactions in the list received;
```

```
            Send 'COMMITTED' message to the GTM along with the
            result of execution;
        end
    end(do forever)
```

CHAPTER IV

PROOF OF CORRECTNESS

Under the assumption that the LCCs ensure local serializability at the site, it can be shown that the proposed algorithm for integrated scheduler ensures serializability, local autonomy, and provide improved degree of concurrency. Correctness of the proposed algorithm is discussed in the following section.

Definition 1 A history H is serializable if and only if the serialization graph SG(H) of the committed projection of any prefix of H is acyclic.

Theorem 1 The algorithm enforces serializability.

Proof Let g_i and g_j be the subtransactions of global transactions G_i and G_j respectively and L be the local transaction at site D_k. Suppose that all three transactions conflict. Since every transaction is assigned a timestamp at the integrated level, the following timestamp orders are possible:

$ts(g_i) < ts(g_j) < ts(L)$

$ts(g_j) < ts(g_i) < ts(L)$

$ts(g_i) < ts(L) < ts(g_j)$

$ts(L) < ts(g_i) < ts(g_j)$

$ts(L) < ts(g_j) < ts(g_i)$

$ts(g_j) < ts(L) < ts(g_i)$

However, all the above timestamp orderings might produce serializable schedules, provided none of the operations belonging to a transaction arrives later than its assigned timestamp, in which case it is rejected. That is, the transaction will not be in the committed projection of history H.

To show now that gi is serialized before g_j and so on depending on the timestamp ordering, it is sufficient to point out that the time stamp assigned to a transaction gi first and then to gj create a direct conflict g_i -> g_j between g_i and g_j. This direct conflict forces g_i and g_j to be serialized according to the order in which timestamps are assigned to them. Since the operations belonging to local and global transactions are treated in the same way in scheduling, i.e., scheduled according to their assigned timestamps, no indirect orders introduced between global transactions due to local transactions. Since timestamps assigned to transactions are unique, there can not be conflicts of the type g_i -> g_j and g_j -> g_i which might create a cycle in the committed projection of the history produced by the scheduler. Since the serialization graph does not contain a cycle, the history representing the schedule is serializable. Hence the possibility of the execution becoming non-serializable is eliminated.

To maintain global consistency, the algorithm ensures that each global transaction will have some relative serialization order in their corresponding LDBSs. Since the relative serialization order of the subtransactions at each LDBS is reflected in the value of the time stamps, the algorithm allows the subtransactions of each global transaction to proceed but commit only if their relative serialization orders are compatible in all participating sites.

Definition 2 Autonomy of LDBSs is maintained if the implementation of an MDBS does not require any changes to existing database schema, database applications, and the database itself.

Theorem 2 The algorithm preserves autonomy of component LDBss.

Proof The LDBSs are not required to inform the GCC about the local transactions executed at the local sites. MDBSs transactions are scheduled by getting information about which sites contain the data items to be accessed by the global transactions, and unaware of the local transactions.

. No modification in the existing LDBMS is demanded by MDBS transactions.

. The MDBS does not require any specific commit protocol to be supported LDBMSs and assumes that any local DBMS is capable of properly committing the results of local

transactions. If a global transaction is to be aborted, GTM instructs the LTM to rollback the updates to the local database using global 2PC protocol.

Definition 3 An algorithm is said to be deadlock free if there is no cycle contained in the serialization graph of the committed projection of any history produced by the algorithm.

Theorem 3 The algorithm is deadlock free.
Proof From theorem(1), there can not be conflicts of the type g_i -> g_j and g_j -> g_i because of timestamp ordering. Therefore, there can not be a situation in which one transaction is blocking other transactions from accessing a data item while requesting to access some other data item which might lead to deadlock. Hence the proof.

Degree of concurrency is improved by the proposed method due to the following reasons:
. Transactions submitted to the integrated scheduler are either immediately scheduled or rejected.
. More than one MDBS transaction is allowed to execute at a site concurrently.
. Number of aborting transactions is reduced by scheduling local and global transactions at the integrated level by using pure integrated scheduler, which avoids the possibility of indirect orders between global transactions.

In order to ensure the atomicity and durability properties of transactions, recovery mechanisms are used in the MDBS environment. 2PC protocol is used for global commitment of global transactions. Before entering the prepared to commit state, global subtransactions are made to store the effects of execution in a non-volatile storage called write-ahead log [Kim93]. The log is used to make the updates permanent in LDBSs, after the local site is recovered from failure.

CHAPTER V

SUMMARY AND FUTURE WORK

A List of major approaches to CC in MDBS

Algorithm	Global Execution Correctness	Local Autonomy	Degree Of Concurrency
1. Altruistic locking alg.	Not guaranteed	Preserved	Low
2. GCC protocol based on site graphs	Guaranteed	Preserved	Low
3. optimistic alg.	Not guaranteed	Preserved	Low
4. Hierarchical alg.	Guaranteed	Preserved	Low
5. GCC alg. used in Super databases	Guaranteed	Not preserved	High
6. Quasi-Serializability	Guaranteed	Preserved	Low
7. Distributed cycle detection alg.	Guaranteed	Not preserved	High
8. Ticket based approach	Guaranteed	Preserved	Not Analyzed
9. Proposed method	Guaranteed	Preserved	High

Summary

Multidatabases are one of the very active database research areas. The 1990 National Science Foundation(USA) workshop on future directions in DBMS research named the area of MDBS as one of the two most important research areas for the 90's. 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 MDBSs have been developed that attempt to provide a tool for managing heterogeneous distributed data sources.

Maintaining database consistency is of critical importance for user acceptance of a MDBS. Therefore it is imperative to develop methods that do not require major modifications to existing DBMS software, but are able to support users data in a consistent and reliable manner.

A multidatabase concurrency control algorithm, based on "Integrated Scheduler" concurrency control mechanism is proposed as a solution for the problem of indirect orders between global transactions due to local transaction, still preserving local autonomy and ensuring global serializability.

The proposed method maintains global database consistency in presence of global and local transactions. The degree of concurrency is improved since more than one multidatabase transaction is allowed to execute concurrently

in an LDBS. It ensures serializability by maintaining analogous serialization and execution orders of transactions and by maintaining relative serialization orders of global transactions at all sites they execute. Also, local autonomy is retained since MDBS transactions do not demand for the modification in LDBMS structure or software.

Future Work

Global transaction recovery is one of the requirements that has to be satisfied by any CC protocol in MDBS environment. This problem has to be given vital importance since the recovery actions of the local DBMSs are outside the control of multidatabase system. Further investigation is needed in this area.

The suitability of concurrent B-tree algorithms for concurrency control and recovery in MDBS environment should be discussed to increase the degree of concurrency further.

Most research to date has focused on how to run transactions in a heterogeneous environment, but we also need to evaluate the cost of transaction processing. For instance, how much more expensive will it be to run transactions when each site runs a different concurrency control protocol.

BIBLIOGRAPHY

[AGMS87] R. Alonso, H. Garcia-Molina, and K. Salem: "Concurrency control and recovery for global procedures in federated database systems," *A quarterly bulletin of the computer society of IEEE technical committee on data engineering, September 1987, PP.5-11.*

[ARNS91] M. Ansari, M. Rusinkiewiez, L. Ness, A. Sheth: "Executing multidatabase transactions," *Proceedings of the twenty-fifth Hawaii International conference on system Sciences, IEEE Computer Society Press 1991, Vol 2., PP.335-46.*

[Ba90] K. Barker: "Transaction management in Multidatabase systems," Ph.D. dissertation technical report, TR 90-23, Department of computing science, University of Alberta, Alberta, CA 1990.

[BKa91] N. S. Barghouti and G.E. Kaiser: "Concurrency control in advanced database applications," *ACM Computing Surveys, Vol. 23, No. 3, September 1991, PP.269-317.*

[BKh87] M.A. Bassiouni, U. Khamare: "Algorithms for reducing rollbacks in concurrency control by certification," BIT(Denmark), Vol. 27, No. 4, 1987, PP.442-57.

[BRG92] R. Batra, M. Rusinkiewicz, and D. Georgakopolous: "A decentralized deadlock-free concurrency control method for

MDBS transactions," *Proceedings of the 12th International Conference On Distributed Computing Systems, June 1992.*

[Ber87] Bernstein, Philip A.: "Concurrency Control and recovery in database systems," *Addison-Wesley Pub, 1987.*

[BGRS91] Y. Brietbart, D. Georgakopoulos, M. Rusinkiewicz, A. Siberschatz: "On Rigorous Transaction Scheduling," *IEEE transactions on software engineering, Vol. 17, No. 9, September 1991, PP.954-60.*

[BS88a] Y. Brietbart, and A. Silberschatz: "Multidatabase update issues", *Proceedings of ACM SIGMOD International Conference on management of data, Vol, 17, No. 3, September 1988, PP.135-42.*

[BS88b] Y. Brietbart and A. Silberschatz: "Multidatabase systems with a decentralized concurrency control scheme," *IEEE Distributed Proc. Technical Committee Newsletter, 1988, Vol. 10, No. 2, PP.35-41.*

[BST90] Y. Brietbart, A. Silberschatz, G.R. Thompson: "Reliable transaction management in a multidatabase system," *Proceedings of ACM SIGMOD International conference on management of data, 1990, PP.215-224.*

[Card87] A.F. Cardenas: "Heterogeneous distributed database management: The HD-DBMS," *Proceedings of the IEEE, Vol. 75, No. 5, May 1987, PP.588-99.*

[Chung90] C.W. Chung: "DATAPLEX, an access to HDDBS," *Communications of the ACM, Jan 1990, Vol. 33, No. 1, PP.70-80.*

[CM87] A. Croker, J. Manage: "Improvements in database concurrency control in locking," *Information Systems, Vol. 4, No. 2, Fall 1987, PP.74-92.*

[Deac91] A. Deacon: "Concurrency control mechanisms for multidatabase systems," *Proceedings of the sixth southern African Computer symposium, July 1991, PP. 118-34.*

[DP92] B.C Desai, R. Pollock: "On schema integration in a HDDBMS," *Information and Software Technology, Vol. 34, Jan 1992, PP.28-42.*

[DEK91] W. Du, A.K. Elmagarmid, W. Kim: "Maintaining transaction consistency in mutidatabases using quasi-serializable execution," *Proceedings of the seventh international conference on data engineering, 1991, PP.360-7.*

[DELO89] W. Du, A.K. Elmagarmid, Y. Leu, S.D. Ostermann: "Effects of local autonomy on global concurrency control in HDDBSs," *Proceedings of the Second International Conference on data and knowledge systems for manufacturing and engineering(IEEE), October 1989, PP.113-20.*

[EH88] A. Elmagarmid and A. Helal: "Supporting updates in heterogeneous database systems," *IEEE Proceedings of the fourth international Conference on data engineering., February 1988, PP.564-569.*

[ELHMRS87] A. Elmagarmid, W. Litwin, S. Heiler, R. McCord, M. Rusinkiewicz, A.P. Sheth: "When will we have true heterogeneous databases," *Proceedings of 1987 Fall Joint Conference PP.746-53.*

[GM91] H. Garcia-Molina: "Global consistency constraints considered harmful for heterogeneous database systems," *First International workshop on interoperability of multidatabases, IEEE Comput. Soc. Press 1991, PP. 248-50.*

[Geo90] D. Georgakopoulos: "Transaction management in multidatabase systems," *Ph.D Thesis, University of Houston, Department of computer science, 1990.*

[Geo91] D. Georgakopoulos: "Multidatabase recoverability and recovery," *Proceedings of the first international workshop on interoperability in multidatabase systems, 1991.*

[GR89] D. Georgakopoulos, M. Rusinkiewicz: "Transaction management in multidatabase systems," *Technical report UH-CS-89-20, Department of computer science, University of Houston, September 1989.*

[GRS92] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth: "Using ticket-based methods to enforce the serializability of MDBS transactions," *IEEE Transactions On Data And Knowledge Engineering, February 1992.*

[HK89] D.K. Hsiau, M.N. Kamel: "Heterogeneous databases," *IEEE transactions on knowledge data engineering, Vol. 1, No. 1, March 1989, PP.45-62.*

[IKK88] T. Ibaraki, T. Kameda, N. Katoh: "Cautious transaction schedulers for database concurrency control," *IEEE transactions on software engineering, Vol. 14, No. 7, July 1988, PP.997-1009.*

[JS93] T. Johnson, D. Shasha: "The performance of current B-Tree algorithms," *ACM Transactions On Database Systems, Vol. 18, No. 1, March 1993.*

[KM92] S. Kang, S.C. Moon: "An integrated access control in HDDBSs," *Eighteenth EUROMICRO Symposium on microprocessing and microprogramming, September 1992, Vol. 35, No. 1-5, PP. 429-36.*

[Ket87] U. Ketter: "The complexity of strict serializability revisited," *Information Process Letter(Netherlands), Vol. 25, No. 6, July 1987, PP.407-11.*

[Kim93] P.C.Kim: "Concurrency control and recovery in multidatabase systems," *Technical report, Korea Advanced Institute Of Science and Technology, Department of computer science, S.Korea, April 1993.*

[KM91] Y. S. Kim, S.C. Moon: "Update synchronization pursuing site autonomy in HDDBS," *Seventeenth EUROMICRO Symposium on microprocessing and microprogramming, September 1991, Vol. 34, No. 1-5, PP.41-44.*

[Kum87] V. Kumar: "An analysis of the roll-back and blocking operations of three concurrency control mechanisms," *AFIPS Confernece proceedings, Vol 56, June 1987, PP.485-97.*

[LE90] Y. Leu and A.K. Elmagarmid: "A hierarchical approach to concurrency control in Multidatabases," *IEEE transactions on database systems, 1990, PP.202-10.*

[LMR90] W. Litwin, L. Mark, N. Roussopoulos: "Interoperability of multiple autonomous databases," *ACM*

transactions on computing surveys, Vol. 22, No. 3, September 1990, PP.267-293.

[MRBKS92a] S. Mehrotra, R. Rastogi, Y. Breitbart, H.F. Korth, A. Silberschatz: "The concurrency control problem in multidatabases, characteristics and solutions," *Proceedings of ACM SIGMOD international conference on management of data, 1992, PP.288-97.*

[MRBKS92b] S. Mehrotra, R. Rastogi, Y. Breitbart, H.F. Korth, A. Silberschatz: "Ensuring transaction atomicity in multidatabase systems," *Technical report, Department of computer science, University of Texas at Austin, 1992.*

[MRKS91] S. Mehrotra, R. Rastogi, H.F. Korth, A. Silberschatz: "Non- Serializable executions in heterogeneous distributed database systems," *Proceedings of the first International Conference on parallel and distributed information systems, IEEE Computer Society Press 1991, PP.245-52.*

[Moon87] S.C. Moon: "Performance of 2PL schemes in DDBSs," *Software and Hardware applications of microcomputers, Proceedings of the ISMM International Symposium, Fort Collins, CO, USA, February 1987, pp.84-7.*

[NHE86] M.H. Nagi, A.A. Helal, A.K. Elmagarmid: "Optimistic Vs Pessimistic Concurrency Control Algorithm," *Proceedings of international conference on parallel processing, St. Charles, IL, USA, August 1986, PP.131-8.*

[Oxb87] E.A. Oxborrow: "Distributing a database across a network of different database systems," *IEE colloquium on distributed database system, Apr 1987, PP.5/2-5/7.*

[PPR91] W. Perrizo, J. Rajkumar, P. Ram: "HYDRO, Heterogeneous Distributed Database Systems," *Proceedings of ACM SIGMOD International Conference on management of data, 1991, PP.32-39.*

[Pu88] C. Pu: "Superdatabases for composition of heterogeneous databases," *IEEE Proceedings of the fourth international Conference on data engineering, 1988, PP.548-555.*

[SKS91] N. Soparkar, H.F. Korth, A. Silberschatz: "Failure resilient transaction management in multidatabases," IEEE Computer, December 1991, PP.28-36.

[SL90] A. Sheth and J. Larsen: "*Federated database systems for managing heterogeneous, and autonomous databases," ACM Computing Surveys, Vol. 22, No. 3, September 1990, PP.183-230.*

[Stan87] W. Staniszkiz: "Integrating Heterogeneous Database," *State of the art report, 1987, pp.229-47.*

[Su87] K. Sugihara: "Concurrency control on distributed cycle detection," *Proceedings of the IEEE international Conference on data engineering, 1987, PP.267-274.*

[Temp87] M. Templeton: "MERMAID - A front end to distributed heterogeneous databases," *Proceedings of the IEEE, Vol. 75, No. 5, May 1987, PP.695-707.*

[Thom87] G.R. Thompson: "Concurrency Control in Multibases," *Thesis, OSU, Department of computer science, 1987.*

[VW92] J. Veijalainen and A. Wolski: "Prepare and Commit certification for decentralized transaction management in rigorous heterogeneous multidatabases," *Proceedings Of the International Conference on Data Engineering, February 1992, PP.470-479.*

[Wolf87] O. Wolfson: "Concurrent Execution Of Transaction," *Information Process Letter(Netherlands), Vol. 24, No. 2, January 1987, PP.87-93.*

[WV90] A. Wolski and J. Veijalainen: "2PC agent method," *Proceedings of PARBASE-90 conference, February 1990, PP.321-30.*

VITA 2

KALPANA HALLEGERE CHIKKANNA

Candidate for the Degree of

Master Of Science

Thesis: CONCURRENCY CONTROL IN MULTIDATABASES

Major Field: Computer Science

Biographical:

Personal Data: Born in Karnataka State, India, On May 20, 1967, the daughter of Chikkanna Hallegere and Lalitha.

Education: Graduated from Vijaya High School, Karnataka State, India in May 1982 and received a Pre-University degree in Science and Mathematics from Government College, Karnataka State, India in June 1984; received a Bachelor of Engineering degree in Computer Science and Engineering from Mysore University in December 1988. Completed the requirement for the Master of Science degree with a major in Computer Science at Oklahoma State University in May 1994.

Experience: Teaching assistant, Department of Computer Science, A.I.T., Mysore University, India, December 1988 to January 1990; Teaching assistant, Department of Computer Science, J.C.I.T, Bangalore University, India, February 1990 to July 1991; Teaching assistant, Department of Computer Science, P.E.S.C.E, Mysore University, India, August 1991 to December 1991; Programmer, Oklahoma Department of Environmental Quality, Summer 1993; Technician, University Computer Center, Oklahoma State University, March 1992 to Present.

Professional Memberships: Member of Indian Society of Technical Education.