SUPPORTING ALTRUISTIC PROTOCOL IN MULTIDATABASE SYSTEM

BY

MAHESH M. J. RAM

Bachelor of Science

University of Madras

Madras, India.

1990

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

SUPPORTING ALTRUISTIC PROTOCOL IN MULTIDATABASE SYSTEM

Thesis Approved:

Huizhu Lu

Thesis Advisor

Jacques LaFrance

Thomas C. Collins

Dean of the Graduate college

PREFACE

The Heterogeneous Distributed DataBase Systems (HDDBS) is an interconnection of Local Database systems which differ in data model, concurrency control mechanisms, etc. As in any other database systems the transaction processing system is at the heart of HDDBS. The global distributed transaction management of Heterogeneous transactions is much complicated than its counterpart in Distributed Database systems due to the property of local autonomy of the component databases.

Indirect conflicts may arise between global sub transactions as there is no way the global transaction manager knows the local conflicts in the local databases. Researchers have come with variety of model to address the issues of the transaction processing of HDDBS. Some of the models violate local autonomy of the component databases or have low degree of concurrency.

The model proposed by [JUHA91] has many desirable properties. It uses the same protocol for distribution and serializability and hence will reduce the cost of distributed global control. [JUHA91] improvises the 2 PC protocol used in DDBS by adding two stricter states called source state and the serializable state. However, this model demands that all the component databases need to follow this strict TM protocol.

Advanced databases like CASE tools etc. do not support serializable state due to the demand for higher degree of concurrency. Hence these databases cannot join the federation. The global user is deprived of accessing these sophisticated databases for their use due to this. This inspired the idea of externally supporting the serializable state for an advanced database so that they can participate in the federation. External support of serializable state for altruistic protocol is done in this thesis. Likewise it can be done for other protocols too and be benefited.

Analytical proof is done to show that this external support does not violate the data consistency. Simulation of the project is done using C on Sequent S81 to prove the feasibility of the idea and performance evaluation is done in aspect of time taken for a given set of operations.

I wish to express my genuine thanks to my advisor, Dr. Huizhu LU, for her invaluable guidance and strong motivation. I greatly appreciate her personal attention and care towards this thesis. Also, my sincere appreciation to the other committee members, Dr. K. M. George and Dr. Jacques LaFrance for their advise and inspiration for this thesis. Also, special mention must be made about Dr. G. E. Hedrick for his advise.

Sincere thanks to the computer department for excellent lab facilities. I appreciate the library staff for their dedicated service.

Special thanks are due to my wife who was very understanding and helpful throughout this work. Special thanks to my four month old daughter, Kirthiga, who kept me awake most of the time and helped me finish my work. My sincere gratitude to my Mother and Father for their great support to pursue my degree.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table | Page

LIST OF FIGURES

Figure | Page

Figure Page

CHAPTER 1

INTRODUCTION

Information is the ultimate power, the commodity of kings, in this information age. The concept of database systems has evolved to address the issues of ever-growing information industry. The necessity to access information across several databases, geographically separated but homogeneous, led to the concept of Distributed DataBase Systems (DDBS). Now a stage is reached where the demand is to access information or data across the boundaries of heterogeneous databases.

Heterogeneous Distributed DataBase Systems (HDDBS) is one of the research area subjected to intense interest of the industry in the last decade. What is a HDDBS? A HDDBS is an interconnection of several Local DataBase Systems (LDBS) which may differ in their data model, data manipulation languages, concurrency control mechanisms, etc. HDDBS is variously known as MultiDataBase Systems (MDBS) or Heterogeneous and Autonomous Transaction Processing System (HATP) or Federated Heterogeneous DataBase System (FDDBS) in the literature.

The property of a HDDBS is to make the heterogenieties of the system transparent to its user. This allows the user to access data contained in various local databases in a uniform way. A HDDBS may be classified into two categories. One that supports only distributed queries and the other that supports both distributed queries and distributed updates. The former type, known as retrieval-only, is quite easy to handle. The later type of more complicated to handle due to its support to distributed queries and distributed updates. To ensure the correctness in the execution of these distributed queries and distributed updates, a Global Concurrency Control (GCC) or Transaction processing (TP) mechanism is needed in a HDDBS.

GCC is at the heart of a HDDBS. GCC preserves the consistency of data despite the concurrent execution of global sub transactions, presence of local transactions and system crashes. The Local DBS that joins the MDBS wants to retain its authority to make its own decisions. This property is referred to as autonomy of the LDBS. The autonomy of a LDBS is further classified into local autonomy, control autonomy and execution autonomy (APPENDIX B).

The autonomy of the component database system of a MDBS makes the task of GCC more complicated than its counterpart in Distributed Database Systems. The reason is that the GCC is not aware of the presence of the local transactions. These local transactions may cause indirect conflicts among global sub transactions.

Let us see an example of indirect conflicts among sub transactions due to the presence of local transactions (Figure 1). Let G1 and G2 be two distributed transactions. Let G1 read a data item i at site 1, performs certain operations and then read data j at site 2. Let G2 performs certain operations and write the result to the data item k. Now there is no direct conflict between G1 and G2. However, the local transaction T local at site 2 performs operations and write to j and k at site 2. This may cause indirect conflict between the global transactions G1 and G2.

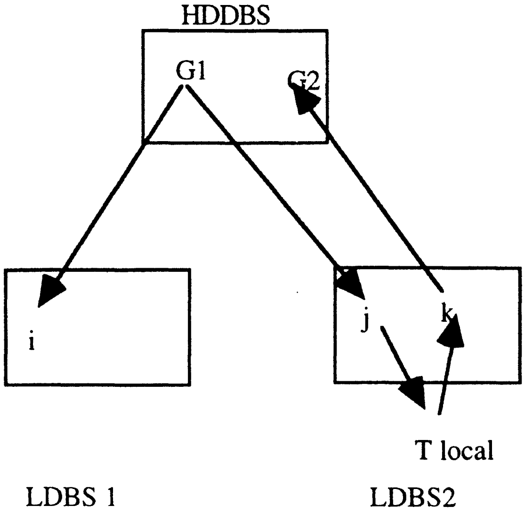


Figure 1. Example of possible indirect conflict among global sub transactions (G1, G2) due to a local transaction (T local) at site 2.

Several attempts have been made to handle the problem of indirect conflicts. Let us briefly see the basic principle of the various models found in the literature and their short falls.

The model Superdatabases [PU87] uses the principle of "order vector" which is based on a global data structure. Thus it is unsuitable for distribution. The model proposed by [ALON87] uses site locking technique. Du and Elmagarmid [BRIE91] have shown that local conflicts may cause this scheme to allow globally non-serializable schedules. The concept of assuming the existence of conflicts between sub transactions (even if there is no actual conflict) is used in the Implicit Ticket Method (ITM) proposed by Geogak Opulus and Rusinkiewiez [BRIE91]. Amoco's Distributed DataBase System (ADDS) [THOM90] uses site graph method. This results in low degree of concurrency due to the reason that only one global transaction can be active at a particular LDBS at a given time. The strong-TM protocol proposed by [JUHA91] improvises the standard 2 PC protocol for DDBMS to support two more states called the "source state" and the "serializable state". This method ensures global serialization and supports higher degree of concurrency. This method requires that all the LDBS to support the above mentioned states. The LDBMS following non-2 PC protocols do not support these states and hence they cannot participate in the MDBS.

The existing solutions in the literature for GCC of HDDBS either compromises the property of local autonomy or has very low degree of concurrency. Effort need to be made to come up with a MDBS model that will make sure of correct global serialization, provide higher degree of concurrency and guarantees local autonomy.

The strong TM protocol proposed by [JUHA91] satisfies the first two requirements. The local autonomy of the LDBS is violated in that they have to give up their locking methods and change to that of strict 2 PC protocol. This is not only agreeable for the LDBS but also affects the performance in advanced database applications. If the strong TM protocol can be supported externally, i.e. by supporting the source and serializable state externally, the local autonomy of the system can be preserved and at the same time the global serialization order can be maintained. The need to support a non 2 PC protocol in the federation has arisen because of the increasing demand for use of non traditional protocols to achieve higher performance in advanced database applications.

There are four objectives for this thesis. They are

1. To externally support the "source state" and "serializable state" [JUHA91] to enable a LDBS following Altruistic protocol, a non 2 PC protocol, compatible with the HDDBS mode of [JUHA91].

The fundamental difference between these two protocols is that in an Altruistic protocol a transaction is required to pre declare all the data items it needs to access in its life-time. Only if all these data items are available the transaction is allowed to proceed. In a 2 PC protocol, no such pre declaration is needed.

The goal is to not disturb the site following Altruistic protocol but to support externally the source state and serializable state for that site.

2. To analytically prove that the proposed external support does not affect the global data consistency.

This analytical proof is based on the assumption that the LDBMS is responsible for the correct execution of the transactions under its control. The proof will be based on the line of proof for data consistency done for databases by Papadimoutri.

3. To prove the feasibility of the external support of the source state and the serilzable state for Altruistic protocol and its participation in the original federation by simulation.

The simulation project is proposed to be done on Sequent using C language. A federation of HDDBS with two sites, one following strict 2 PC protocol and other following Altruistic protocol is to be simulated. The Altruistic protocol will be externally support the source state and the serializable state.

4. To do performance comparison between the original model and the one with the proposed external support in aspect of time taken for a given set of operations.

The main advantage of the proposed external support is that the participating component LDBS with Altruistic protocol need not give up its local autonomy and change its protocol to a strict 2 PC protocol. The effort needed to make this conversion and the subsequent effect of this change on the local user can be avoided.

In chapter 2, the various existing popular models and their implementation techniques that are found in the literature for addressing the issues of GCC for MDBS are discussed.

In chapter 3, analytical proof of correctness is given to show that the external support does not violate global data consistency. In chapter 4, the simulation of the federation with a component following Altruistic protocol with the proposed external support is done to show the feasibility of the thesis is given. The performance comparison between this model and the original model [JUHA91] is also done. In chapter 5, the summary and conclusion of this thesis effort is listed.

CHAPTER 2

LITERATURE REVIEW

TERM DEFINITIONS

Let us see the relevant term definitions which we will be using throughout this report. A transaction is a unit of system's work. In a HDDBS, a transaction might be initiated due to either the request of the local user or global user. The former is called the Local transactions and the latter is termed as Global transactions. The global transactions are decomposed into units that operate on one site. These units are called sub transactions or distributed transactions of the given global transaction.

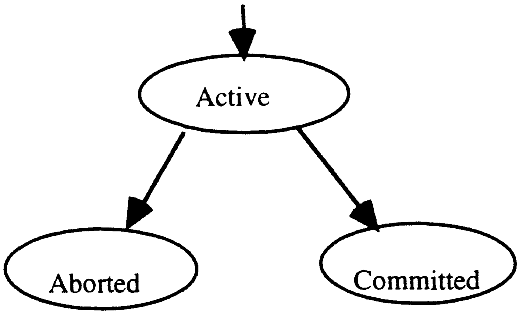


fig 1: The states of transactions from the user's point of view.

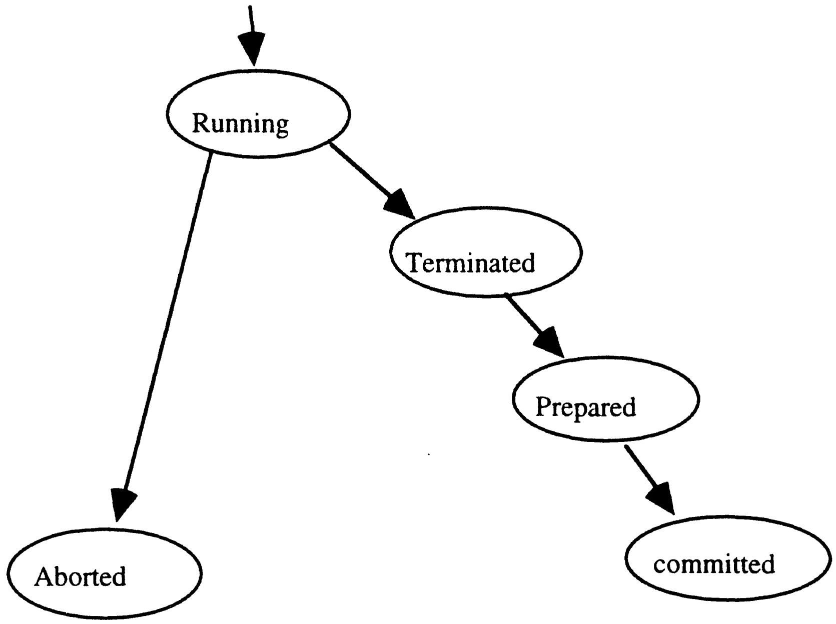


Figure 3. States of transaction from the system point of view.

The different state of transactions from the user's point of view and from the system's point of view is given in Figure 2 and Figure 3, respectively. A transaction is said to be aborted if the system terminates a transaction. The effect of the aborted transaction, if any, is undone to ensure the consistency of the database. A transaction is said to be recoverable if it has not read a data written by an uncommitted transaction. A transaction is terminated if its run to completion successfully. A transaction is said to be in the prepared state if it has terminated successfully and can be committed from the atomicity point of view, recoverability point of view and the serializability point of view. The effects of the terminated transaction or prepared transaction is not reflected by the permanent memory. The effect of a committed transaction is shown by the database even in the event of system failure.

The atomicity property of a transaction means that the effect of the transaction is either reflected wholly or not at all by the permanent memory. The concept of serializability means that the result of the concurrent execution of transactions on the database is equivalent to some serial execution of the transactions.

The transactions that are initiated by the local user are called the local transactions. The transactions that are initiated by the global user are called the global transactions. The global transactions are decomposed by the transaction processing system into parts that act on one LDBS. Each of these parts is called the sub transaction or distributed transaction. After submission of the sub transactions, the local transaction processing system has absolute control over the execution of them.

Transactions are said to be in direct conflict if they access common data objects. Direct conflicts occur in traditional databases. The problems due to direct conflicts are well studied and various effective solutions are available in the literature. Conflicts between transactions are represented using conflict graph. The conflict may be read-write, write-read or write-write. For a pair of conflicting operations, relative order of execution is important. If the order is the same for each pair of conflicting operations from the two transactions, the transactions can be regarded as having executed in the serial order.

The notion of conflict serializability is captured in the serializability graph (SG) that consists of a node for each committed transaction and an edge from a node T to a node T' if and only if an operation of T conflicts with and is executed before, an operation in T'. It can be shown that the acyclicity of the SG is a necessary and sufficient condition to guarantee conflict serializability since a topological sort of the graph provides an ordering that corresponds to an equivalent serial execution.

To ensure that unexpected executions do not occur when a transaction aborts, a transaction must always maintain its isolation properties. To do this serializable executions are often further restricted to prevent uncommitted data from becoming visible to other transactions. In the event of system failure the result of all committed transactions should be reflected by the database and also any partial results of all uncommitted transactions must be undone. This action ensures the properties of atomicity and durability.

The unique problem faced by HDDBS is indirect conflicts. Due to the property of local autonomy the GCC is not aware of local transactions. The presence of local transactions might result in an indirect conflict among global sub transactions. Indirect conflicts arises due to the property of autonomy of the component databases. The three types of autonomy are execution, control and local autonomy.

Execution Autonomy. The right of the local DBMS to delay or reject any operations of the local transactions or the global sub transaction is called the execution autonomy. Similarly the LDBMS has the right to abort the execution prior to a successful committment. Infringement of the execution autonomy is not advisable because it needs major changes to the underlying connote mechanisms.

Control Autonomy. The right of the local user to submit local transactions to the LDBMS directly without any knowledge of the presence of the MDBS. The control autonomy of a LDBMS refers to the degree that the MDBS does not control the local transactions executing at the site. So control autonomy is preserved for a LDBMS if the MDBS can neither abort the execution of a local transaction nor delay its operations.

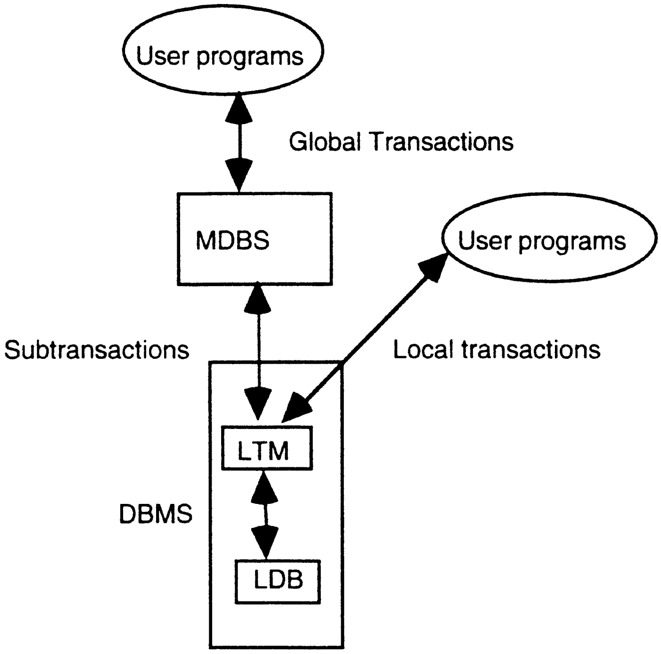


Figure 4. A MDBS system preserving control autonomy [NAND91].

The local autonomy requirement implies that the DBMSs being integrated into an MDBS environment cannot participate in the execution of the Global Atomic Committment (GAC) protocol. The reason is LDBMS has to relinquish its right to either abort or commit a sub transaction until it receives a final decision from the GAC protocol.

In the following section, we are going to discuss the various solutions that are available in the literature to address the problem posed by these indirect conflicts and their drawbacks.

Superdatabases [PU87]

The basic concept of Superdatabases is "order vector" [PU87] which is based on a global data structure. The idea here is to rely on local concurrency control mechanisms to guarantee local serialization order and then add a global check to provide global serialization. This is done using a global data structure called order-elements which forms the order vector.

An order-element (o-element) is defined to be a representation of the serialization order of sub transactions in a component database: If $T^1{}_A$ -> $T^2{}_A$, then o-element($T^1{}_A$) < = o-element($T^2{}_A$). An example of an o-element is an integer representing the serial order of the corresponding transaction in the local TP system.

An order-vector(o-vector) is the concatenation of o-elements from the component databases. For example, o-vector(T_1) is (o-element($T^1{}_A$),o-element($T^1{}_B$)).The order induced on o-vectors by the o-elements is defined strictly: o-vector(T_1) <= o-vector(T_2) if and only if for all component databases X, o-element($T^1{}_X$) <= o-element($T^2{}_X$). According to this definition, if o-vector(T_1) <= o-vector(T_2), then all sub transactions are serialized in the same order. The global serializability is ensured by checking its o-vector against the o-vectors from previously committed supertransactions.

The global concurrency control mechanism can be either centralized or distributed. In a centralized scheme all the global control resides on one site. If that system crashes, the entire system is inaccessible. On the other hand, in a distributed GCC mechanism, each site plays an identical role in performing the decisions related to global concurrency correctness. Hence, even if one site crashes the remaining system is still accessible.

The drawback of the superdatabase model is that it is primarily based on a global data structure order-vector. This fact makes it suitable only for centralized GCC and not for a distributed GCC of a HDDBS.

Site Locking [ALON81].

This model is based on the concept of controlling the submission and the execution order of global sub transaction. The method it uses to employ this technique is site locking. When a global sub transaction needs to access a data item belonging to a LDBS, that particular LDBS is locked by this global sub transaction and is not available to any other global sub transaction.

In Altruistic protocol, all the data items needed by a transaction are requested before the transaction is initiated. If all the resources are available it locks them all and then only proceeds. At the end, either successful termination or aborted by the concurrency mechanism, it releases all its locks. [ALON87] has extended this altruistic locking protocol principle for HDDBS as follows. He proposed to use site locking in the altruistic protocol to avoid undesirable conflicts between global transactions.

A global transaction pre-declares the different component databases it needs. These sites, if all of them are available, are locked and the global transaction runs to completion. Only then it releases all its locks. Any other global transaction can proceed only when there is no other active predecessor to it. So at any given instance only one global transaction is allowed to access a LDBS. In other words, given two global transaction G1 and G2, this protocol allows their concurrent execution only if they access different LDBS. If there is a LDBS that both G1 and G2 need to access, G2 cannot access it before G1 has finished its execution there. Du and Elmagarmid have shown that local conflicts, which are neglected by this model, may cause this scheme to allow globally non-serializable schedules.

Implicit Ticket Method (ITM)

The basic assumption of the ITM, proposed by Georgak Opulus and Rusinkiewiez, is that there exists conflicts among sub transactions, even if there is no actual conflict. Conflicts are forced between global transactions by the system to ensure global serialization.

The conflicts between global transactions are forced as follows. With each LDBMS a unique ticket is associated. For example the $LDBMS_i$ as a unique ticket T_i associated with it. Any sub transaction which acts on this LDBMS, should first access this ticket. Only if this ticket is free, the sub transaction can lock it and proceeds its execution in that LDBMS. At the end of successful termination, the ticket is unlocked so that the next global sub transaction can access the LDBMS. So, at any instance, only one global sub transaction can be active in a given LDBMS.

This scheme is basically another way of implementation of the method proposed by [ALON87] and so it has all the drawbacks of that method. This method totally disregards the conflicts among the global sub transactions due to the presence of local transactions. Hence the correctness of the transactions is not ensured. The resulting degree of concurrency among global transactions is also very low.

Amocco Distributed DataBase Systems (ADDS) [LITW90].

ADDS uses site graph method for its GCC mechanism. The method does not rule out the conflicts between global transaction whenever they access the same local database.

The implementation of the site graph is done as follows. The nodes of a site graph correspond to the LDBS which stores the referred data objects. The edges of the global site graph are the global sub transactions. When a global sub transaction request access to a data at a LDBS, the LDBS is included as a node in the global site graph. An appropriate edge is added to the global site graph to represent the global sub transaction accessing that data object. If this addition does not create a cycle in the global site graph, the multidatabase consistency is preserved and hence that sub transaction is allowed to proceed. If the addition of this edge creates a cycle in the global site graph, this operation may result in the data inconsistency and so this operation is denied permission. The appropriate node and edges are then removed from the global site graph to reflect the correct picture of the situation.

The advantage of this method is it detects conflicts among the global sub transactions and ensures local autonomy. The drawback of this scheme is low degree of concurrency among the global sub transactions. Also no discussion was provided as to when the edges can be safely removed from the global site graph [BRIE91].

Failure resilient transaction mechanism in MDBS [NAND91]

[NAND91] combines various existing techniques available in the literature. It infringes on the control autonomy to provide reliability for distributed MDBS.

The system structure is given in figure 5. Each LDBMS is assumed to provide local recovery from failure and deadlocks and to provide Atomicity, Committment, Isolation and Durability (ACID) properties.

The MDBS$_i$ engages in the GAC protocol when a sub transaction T have been successfully executed (except for the final commit). MDBS$_i$ saves the set of changes ,say ch(T), made by T in stable storage before it declares that the T is in the prepared state to the coordinator. No further operations from any other transactions is submitted at site DBMS$_i$ till a decision is reached. When a decision is reached, either commit or abort, MDBS$_i$ passes it to the LDBS$_i$ which performs the actual operation. If the decision is to commit and if it is successfully executed then ch(T) can be discarded and the site process the next transaction.

However, if the commit request for T is not carried by the DBMS$_i$, the MDBS$_i$ performs the following actions. All active transactions directly following T are aborted forcibly to isolate T. The set ch(T) is converted to a write-only transaction which is repeatedly resubmitted to DBMS$_i$ until successfully committed.

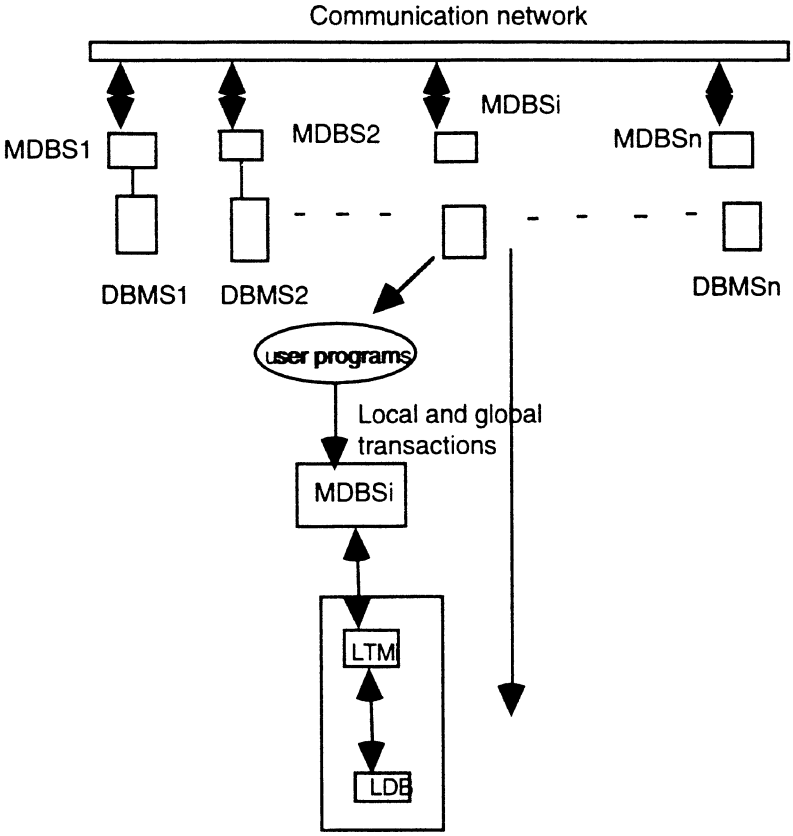


Figure 5. A distributed MDBS with the control autonomy traded for reliability [NAND91].

The method violates control autonomy and local autonomy of the LDBS. MDBSi has the right to abort all the local transactions which violates the execution autonomy too. Further this action may lead to poor performance of the system.

Rigorous Transaction Scheduling [BRIE91].

[BRIE91] defines Strictness, rigorousness and commit-differed properties for the transaction mechanisms. [BRIE91] shows that if the MDBS is commit-deferred and all the LDBMS are rigorous than global serialization is ensured.

A schedule produced by a transaction mechanism is said to be strict if no data item may be read or written until the transaction that previously wrote it either commits or aborts. A transaction mechanism guarantees rigorousness if it guarantees strictness and no data item may be written until the transaction which previously read it either commits or aborts. A global transaction is said to be commit-deferred if its commit operation is submitted by the MDBS to the various local DBMSs at which it was executed, only after all the read/write operations of the transaction have been executed by the appropriate LDBMS.

[BRIE91] shows that Altruistic locking [ALOn90], the site-graph method, ITM and the 2 PC agent scheme guarantees global serializability if each global transaction has at most one sub transaction at each LDBS and all LDBMS are rigorous.

This method requires all LDBMS to be rigorous which violates local autonomy and has low degree of concurrency.

Principle of Committment Ordering [YOAV91].

In a 2 PC protocol if the write locks issued on behalf of a transaction are not released until its end it is said to be a Strict-2 PC (S-2 PC) [BRIE91]. If all the locks are not released before the transaction ends (either commit or aborted) it is called Strong-S 2 PC (S-S 2 PC) [BRIE91].

[YOAV91] generalizes S-S 2 PC using Committment Ordering (CO) which is a property of the histories that guarantees serializability. In a history if the order of any 2 conflicting operations in any two committed transactions matches the order of the respective commit event then it is said to be CO. CO can be implemented in a non blocking manner, which is deadlock free. However, there is the possibility of cascading aborts when recoverability is applied. This method is useful for LDBS which follows S-S 2 PC.

Strong-TM protocol [JUHA91]

[JUHA91] proposed the strong-TM protocol for global distributed control of HDDBS. In this method, [JUHA91] has increased the semantics of the 2 phase commit protocol used in Distributed DataBase Systems (DDBS) by replacing the prepared state by more restrictive states called the source state and the serializable state.

The primary assumption of this model is that the atomicity of the distributed transaction is ensured by the 2 PC protocol. The global coordinator of a 2 PC protocol of a DDBS decides to commit the distributed transaction if all its sub transactions are in a state called the "prepared state". However, for the GCC of a HDDBS, this prepared state is not good enough to ensure the global serializability. Let us see an example to explain this point. Consider the situation where all the sub transactions of a global transaction in a HDDBS are in the prepared state. At this point the GCC coordinator can commit the distributed transaction from both the atomicity point of view and the recovery point of view. However, from the serializability point of view the coordinator is not in a position to commit the distributed transaction.

Consider the execution of the distributed transactions T_1 and T_2 with history h_A at site A and h_B at site B.

h_A = r1[a]w1[b]w2[a]

h_B = r2[c]w2[d]w1[c]

Since neither T_1 has read a value written be T_2 nor T_2 has read a value written by T_1, the sub transactions T_1 and T_2 can be in the prepared state at the same time. However the execution is not serializable because T_1 and T_2 have accessed conflicting data item in a different order. The GCC is not aware of this and this may lead to globally non-serailizable schedules.

Source State and Serializable State. [JUHA91] improvises the 2 PC protocol be defining two more restrictive states to handle this problem. If a sub transaction is in the prepared state and no other active transaction has accessed a data item before T wrote it, then it is said to be in the source state. The source state ensures that the execution of the committed distributed transactions is not only atomic but also serializable. It also ensures that the distributed transactions cannot interfere with other distributed transactions even though the local transactions are taken into account.

[JUHA91] proves that this source state is not enough to produce globally serializable schedule. [JUHA91] defines another state called the serializable schedule to take care of this problem. A sub transaction T is in the serializable state if it is in the prepared state and none of its predecessor is an active transaction. The intuition behind the serializable state is that the sub transaction in each site is in the serializable state according to its local serialization order.

The desirable properties of this model are it is distributed and it follows the same protocol for atomic committment and global concurrency control which decreases communication cost.

The drawback of the model proposed by [JUHA91] is that it requires each of the participating LDBS to support the source state and the serializable state. The motivation of this thesis is to support externally these states for a non 2 PC protocol. The same principle can be applied to any other non 2 PC protocol to achieve global serializability and local autonomy in HDDBS.

CHAPTER 3

ANALYTICAL PROOF OF CORRECTNESS

In this chapter, we are going to theoretically prove that the proposed external support of the serializable state for a local database not supporting serializable state will not lead to inconsistent database.

A database is said to be consistent if it reflects the effect of a committed transaction wholly or not at all. The transaction manager should ensure that the concurrent execution of several transactions is equivalent to some serial execution to maintain the data consistency. In our proposed external support, the global transaction manager gets a different picture about the states of transactions of its component database from what actually happens there.

In the original model, all the component databases follow the strict TM protocol. In this case, the global transaction management is sure of committing a global transaction from the serializable point of view due to the fact that the local database will commit a transaction only if it is in the serializable state. [JUHA91] shows that if all the local databases support this serializable state and the global transaction mechanism also support it, then the correctness of global transactions is ensured. In our case, a component database is not supporting the serializable state. Instead an external layer presents as though the LDBS supports serializable state. The global transaction management makes it decisions based on the picture presented by this external layer. Hence the need to prove that this external support does not violate data consistency.

[PAPA86] states an elegant and simple way of showing data consistency of a database. The approach is adopted here for our analytical proof. [PAPA86] states that a database is consistent if each transaction that operates on its data sees a consistent state and if the resultant data after the operations of set of transactions is consistent. In our external support model, the global transaction mechanism submits the distributed transactions to the component databases. The local transaction mechanism has absolute control over the transaction that are submitted to it and the global transaction manager does not have any say in it. Hence clearly the local transaction manager is solely responsible for the correct execution of all local transactions and the global transaction manager should take care of the distributed global transactions.

The basic assumption of the model is that each LDBMS ensures the atomicity and the serializability of transactions under its territory. It is upto the global transaction manager to take care of the atomicity and the serializability of its distributed transactions. Since the global transaction manager commits its transactions only when they are in the serializable state, according to [JUHA91] it automatically ensures correct execution. When a component database does not support serializable state, we have to make sure that the local transaction manager does not violate its data consistency inspite of the presence of the global sub transactions. So it will be enough to show that the LDBS maintains its data consistency while participating in the HDDBS to prove that the data consistency of the overall system is not violated.

The external support also does not lead to globally non serializable schedule. The global transaction manager follows strict TM protocol [JUHA91] and [JUHA91] have shown that this assures correct execution of global transactions. The external support layer shows that a transaction is in serializable state only when it actually is in serializable state. So the global transaction manager's decision is in no way affected by the external support.

The analytical proof of correctness to show that external support does not violate data consistency is to be done by contradiction. The formal proof with the basis and assumption made is given below.

Analytical proof of correctness

To prove that the proposed external support does not violate data consistency.

[PAPA86] states that to show that data consistency of a system is not violated it is sufficient to show that

(I) each transaction sees the consistent state of the database and

(ii) the database is consistent after the execution of the transactions.

Assumption:

The LDBMS is responsible for the correct execution of transactions under its control.

Theorem:

To prove external support of serializability does not violate the data consistency of the HDDBS.

Proof:

(I) By contradiction assume that a sub transaction sees an inconsistent state of the database. It might be because of either a local transaction (say T_{local}) or another sub transaction (say T) left the database in an inconsistent state.

The LDBMS of the site ensures that the execution of T_{local} or T is equivalent to the serial execution. So T_{local} or T is executed correctly and the execution of it left the database in an inconsistent state, the transaction must be incorrect, a contradiction.

(ii) Assume that the state of the database is inconsistent after the execution of a set of transactions. So, there must be a sub transaction or local transaction that left the database in an inconsistent state. Since the LDBMS of the site ensures that the transaction is executed correctly, the transaction must be incorrectly executed, a contradiction.

CHAPTER 4

IMPLEMENTATION

OBJECTIVES OF THE SIMULATION

The primary objective of this simulation project is to show that the idea of externally supporting the source and serializable state for a local database system not supporting those states is feasible.

The external support is to be implemented for the underlying site following Altruistic locking. Altruistic locking has been selected for the simulation because it is a non 2 PC protocol and it is widely used in modern advanced applications. The first step will be to study the altruistic locking in depth and find out its similarities and differences with that of 2 PC protocol. This information can be used to support externally the required states for the altruistic protocol. This will enable it to participate in the federation of MDBS [JUHA91].

The other objective of the simulation is to study the effect of the external support on the performance of the system. For the performance comparison, two models of HDDBS is to be implemented. The first model of MDBS will follow the requirements of the strong TM protocol. The second model of MDBS will consists of a distributed global transaction manager and with all but one component databases following strong TM protocol with the LDBS (say the X^{th}

component) following altruistic locking. The performance comparison between the models is to be done with throughput as the criterion.

Consider the original locking protocol of the X^{th} component to be Altruistic locking. This locking has higher degree of concurrency in supporting long transactions. If this LDBS as such wants to join in the model 1 [JUHA91], it has to modify its protocol to that of strong TM protocol. So the comparisons of performance of this LDBS under similar conditions need to be done for model 1 Vs model 2 to find the effect of the external support on the LDBS.

Another factor is the effect of the external layer on the performance of the federation. So the performance comparison of the HDDBS of model 1 Vs model2 need to be done under similar conditions. The external support preserves local autonomy and if it does not affect the overall performance of the system then it will play a major role.

The performance comparison is done in aspect of time taken for a given set of operations. Histogram is to be used to represent the result graphically.

Problem Statements for the simulation project

1. To simulate a Heterogeneous Distributed Database [HDDBS] environment which follows the model proposed by [JUHA91].

The environment will consists of two separate Local database systems each having its own transaction processing systems and a global transaction processing system for the HDDBS. Both the local transaction processing systems and the global transaction processing systems follows the strong-TM protocol [JUHA91].

2. To simulate a HDDBS environment with the proposed external support for the heterogeneous local database following Altruistic protocol.

The environment will consists of two separate local databases, one following the strict 2 PC protocol and the other following the Altruistic protocol. The global HDDBS will be the same as that of [JUHA91]. In order to make the altruistic protocol compatible with the global transaction mechanism, the proposed external support of the serializable state will be implemented.

3. To compare the performance of the two models simulated in aspect of time taken for a given set of operations..

In order to make the performance comparison, throughputs of both the model 1 and the model 2 are to be calculated under exactly similar conditions. This will be repeated by varying the length of the transactions and also the number of the transactions to do the comparison study under different situations. These results will be used to plot the histogram for easy comparison of performance between the two models.

Environment of the simulation.

Platform	:	Sequent.
Language	:	C.
Special command used	:	Fork

Implementation details.

The simulated environment will consists of two separate local database systems each having its own transaction processing system and database and a global transaction processing system for the distributed global control of MDBS.

We need to create a practical environment where several transactions, both user initiated and system initiated, are executed parallely by the transaction manager. Each transaction may differ in duration, operations, etc. Some transactions run to completion successfully and are committed by the system and some transactions are aborted before completion and have to be started all over again. Some transactions may be blocked by the system from proceeding in order to maintain the data consistency inspite of the concurrent execution of transactions. The various modules implemented, their function, operations and all the other details of their implementation are presented below.

Transaction Simulator. A transaction simulator is made to simulate the real life simultaneous generation of transactions. These transactions will represent the situation where the local users of both the LDBS and the distributed global user make transaction request. The transactions generated by the appropriate transaction simulator are then submitted to the respective transaction processing system. The transaction simulator uses a random number generator to fix the various characteristics of a transaction namely duration and operations. Let us first see about the random generator implemented.

Random number generator.[PARK89] It uses a variable seed to calculate the random number. The value of the seed is initialized to one. Each time the routine is invoked it calculates the new value of seed using a formula and returns a floating point value for seed.

The following values are used every time in the calculation.

a = 16807.0

m = 2147483647.0

q = 127773.0

r = 2836.0

The formula used for the random number generator is

```
hi = seed/q
lo = seed - q * hi
test = a * lo - r * hi
if  ( test > 0.0 ) seed = test
else
seed = test + m
Return ( seed/m )
```

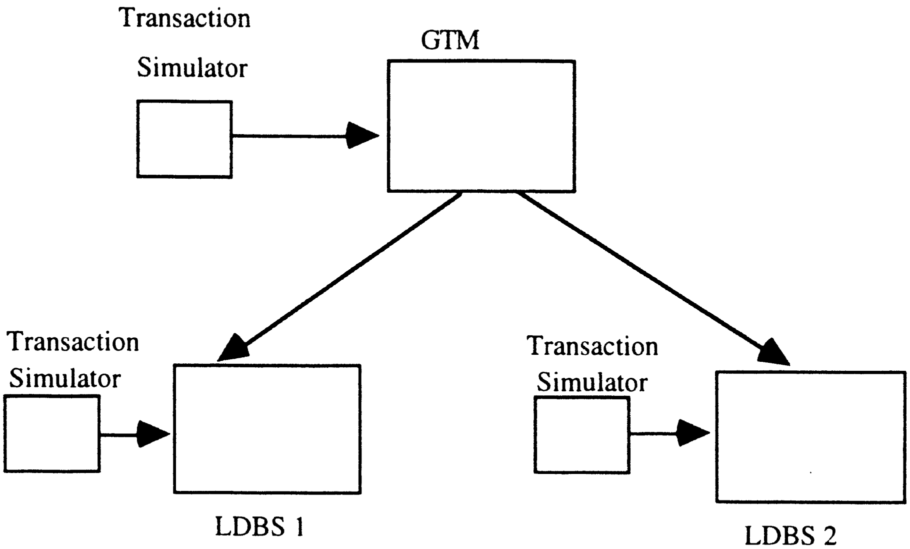


Figure 6. Implementation of MDBS following strict TM protocol [Model 1]

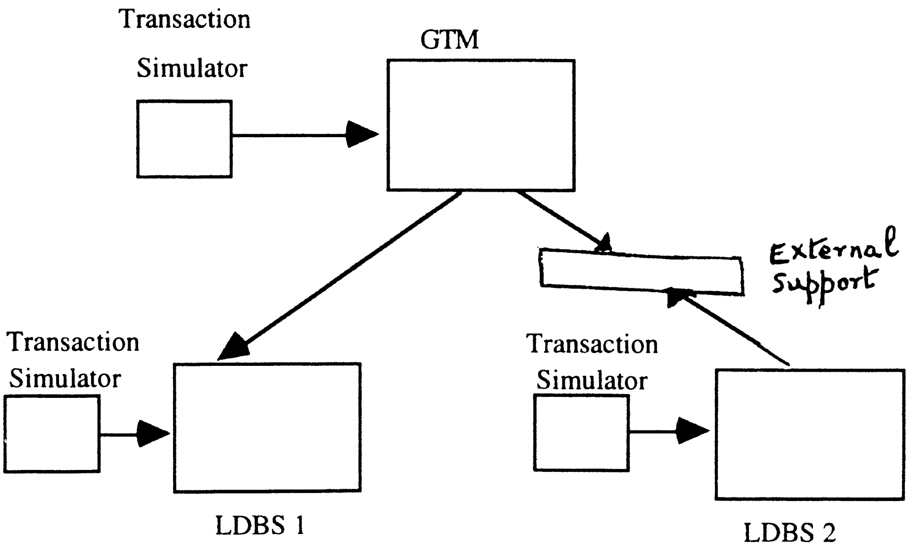


Figure 7. Implementation of MDBS following strong TM protocol with LDBS 2 following Altruistic protocol supporting strong TM externally [Model 2].

The transaction simulator module consists of four modules. They are the transaction initiator (tran_init), local transaction simulator (loc1_tra_sim and loc2_tra_sim) and global transaction simulator (Glo_tra_sim).

Transaction initiator. The transaction initiator initiates the concurrent transactions (32 in total) using fork command. For every fork command 2 child transactions are generated. Using a loop construct the required numbers of transactions are generated. The parent transaction is used to initialize all the shared memory structures that will be used.

First a process id array is build by using a shared variable as index to a shared memory array. The index to the array is set to zero. Each transaction's process identification number (process id #) can be obtained using the system function called getpid(). One by one, each transaction id # is obtained and stored in the process id array incrementing the array pointer. S_lock() and s_unlock() functions are used to make sure that only one transaction can access a shared memory variable at a given time. This array is build to facilitate easy identification of all the transactions. In effect, we are building a tree of transactions.

To identify a transaction, we need to get its process id # and make a match against the process id array. The corresponding index number of the array is our transaction number. This enables us to identify any transaction of interest easily.

The transaction simulator after initiating the transactions classifies them into three sets: Local 1 transactions, Local 2 transactions and Global transactions. The transaction sets that form the array positions zero to 15 are assigned to be the distributed global transactions. These transactions are classified to form 8 groups of two transactions each. Each group forms the two components of a single distributed transaction. The idea is to simulate the presence of eight global transactions which are processed and decomposed into 16 distributed sub transactions. So that each simulated global transaction accesses both the local databases and each of them is decomposed into two sub transactions which accesses only one of the LDBS.

Next step is to fix the details of the transactions. A similar shared memory array is formed to be used as the transactions private memory. By this, the transaction details associated with its number and hence can be easily obtained. The information stored are status of transaction (Local 1, Local 2 or Global), Modes of transaction (Active, Inactive, terminated, committed, aborted, source and serializable), # of operations to be performed and the kind of operation (Add, Subtract, Multiplication or addition) and operands for each operation. The data structure used to implement is the array of structure of structures.

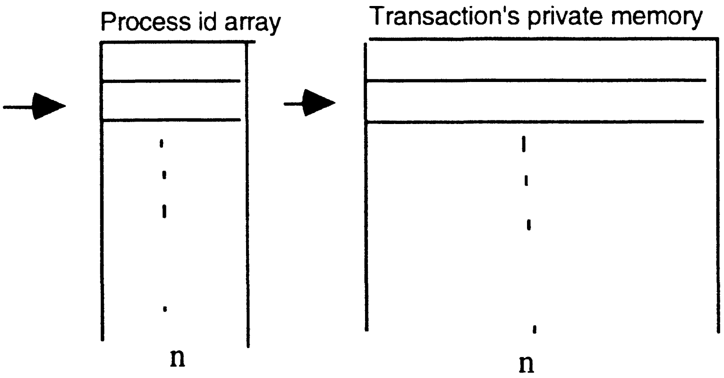


Figure 8. Data structure used for process array and it's private memory and their relation.

The initial modes of all transactions are set to be inactive. All the other details like the duration of transaction, operation and operands are decided using the random number generator. For each of these data an integer is multiplied to the random number to fix the range for that data. The initial duration of the transaction was fixed to be 10 * the random number. Then it is varied to 20, 30, 40, 50 and 60. The transaction with the lowest number is the shortest transaction and the one with the highest number is the longest transaction. The four operations implemented are addition, subtraction, multiplication and division. Each of these operations is assumed to require the same amount of time. This is calculated by multiplying the random number with 4. The number zero represents addition, one represent subtraction, two represent multiplication and three represent division.

For the local transactions, the operands for these operations are selected randomly out of the 2000 data of the database. For the global transaction, for each operation the database from which the data need to be taken is also randomly selected. To simplify the implementation for the global transaction its operands are made to be from the same database instead of across the two local databases. This transaction simulator is common for both the models.

The Original Model. The simulated original model consists of two local sites following the strict TM protocol proposed by [JUHA91] and a Global Transaction Manager following the strict TM protocol.

First two local database sites are implemented. Each local site has a local database consisting of 2000 data items and a local transaction scheduler. Each local database is an array of 2000 integer items and these data are integer values created using the same random number generator with 5000 as the limiting factor.

Local scheduler. The local scheduler acts as a controller for the set of transactions submitted to it. It ensures the atomic and serializable execution of all the transactions according to the model of strict TM protocol by [JUHA91].

The transaction simulator is made to generate the desired set of transactions. These transactions are then submitted to the appropriate scheduler. This can be easily done since we have the information about which transaction belongs to which database.

The local scheduler concurrently executes all the transactions submitted to it. All the data items needed for operation of a transaction is checked for availability. If available, the transaction puts appropriate lock on the data and accesses it. These locks are released only after the completion of the transaction. If the data is not available, the scheduler blocks the transaction and puts the appropriate edges in the conflict site graph.

For example if transaction T1 is using a data while T2 request the same data, then the request of T2 is denied. A conflict edge T1-->T2 is put in the conflict graph. The conflict graph was implemented using the adjacency matrix.

In the implementation, the transaction's number is obtained using getpid() function. If the transaction belongs to the LDBS then only it is allowed to proceed. If not it is routed out of the scheduler. The transactions which belong to the scheduler 1 of site 1 are the local transaction set from 16 to 23 and sub transactions one, three, five, seven, nine, eleven, thirteen and fifteen. The transactions that belong to the local scheduler 2 of site 2 are the local transaction set 24 to 31 and the global sub transactions zero, two, four, six, eight, ten, twelve and fourteen.

The scheduler is implemented to make sure that a transaction does not acquire any partial data. Check is made to make sure that partially acquired data are released promptly. A shared memory variable status is used to check whether all the data items requested were allotted to the requesting transaction are not. A transaction that gets all its data is made to be an active transaction by the scheduler.

The blocked transaction becomes active when the data on which it was blocked becomes available. An active transaction invokes the routine active_tra(). It executes all the operations of the transaction. To simulate concurrent environment, time slice technique is used. Each active transaction is allotted a certain equal amount of time period by the scheduler. A transaction can execute its operations only within this time period. If the transaction is completed executing within its time slot, it proceeds to check whether it is in the prepared state. The long transactions will require more number of time slots. No priority was implemented. The transactions were served on first come first serve basis.

When a transaction completes all its operation, the scheduler checks whether it can be committed from serializability point of view (supports serilaizable state). If so, it commits the transaction. A terminated transaction having a predecessor in the conflict graph cannot be committed.

To simulate a real environment abort of transactions by scheduler is done using the random number. A transaction can be aborted at any stage of its lifetime. In real life it may be user initiated or system initiated abort. To simulate this, transaction is made to perform an abort check routine before performing each operation. These abort check is done using the random number generator. When number zero is generated, an abort routine is invoked which aborts the transaction.

When a transaction is aborted, the recoverability routine is invoked to make the appropriate modifications in the database so as to erase all the effects of the aborted transaction on the permanent memory. This is done using the history file which registers each and every action of the operations of a transaction. For each write operation, the previous data values before the write is written in the history file along with the transaction that wrote it. History file was implemented as a shared structured array. It has the same relation with the process id array as that of the transactions private memory array.

Global scheduler. The global scheduler process the global transactions and decomposes them into sub transactions that acts on different databases. After the successful completion of all its sub transaction a global transaction can commit successfully. In the simulation, all odd sub transactions are submitted to the site 1 and all the even sub transactions are submitted to the site 2.

Using the conflict graph supplied by the sites the global controller forms the union of conflict graph called the global site graph. The global scheduler checks if all the component sub transactions of a global transaction have reached the serializable state. If that transaction does not have an active transaction as its predecessor in the global site graph then it is in the serializable state. In that case the global scheduler decides to commit the transaction. It passes this information to the appropriate local scheduler that does the actual work. When the sub transaction in its control is successfully committed, it passes that information to the global scheduler. After receiving this message the global scheduler commits the global transaction. In case the sub transaction that was in the serializable state is aborted this message is conveyed to the global scheduler. Then the global scheduler has to resubmit the sub transaction to that local scheduler.

A global universal graph is maintained which is the union of all the local conflict graph to check for any indirect conflicts among the sub transactions and wrong order among sub transaction. This is done by maintaining the union of all the site adjacency matrix as the global adjacency matrix. This matrix represents the global site graph. In case of a direct conflict between the sub transactions, the global scheduler submits them one by one to the sites.

For this model, the throughput of the system and that of the site 2 was calculated in milliseconds. Several readings were done by varying the duration of the transactions (from 20 to 50), number of transactions (from 8 to 32). These values were used for the performance comparison study.

Model 2. The model 2 consists of a LDBMS supporting serializable state, a LDBMS supporting altruistic locking, the software external support layer and Global transaction manager. The scheduler for the site supporting serializable state is the same as that of model 1.

Local Scheduler 2. In an altruistic locking [APPENDIX A] negative access information and early releasing of locks are done to improve the performance of the system. To simulate this situation we need to maintain information about whether a transaction is done with a data item or not. This was implemented using a count down counter. The scheduler process the transaction and finds out the information how many times that transaction will access each data in the database. Each time the transaction accesses the same data item the counter is incremented. The final value of the counter will give the number of times that data is needed by the transaction. This value is stored in the private memory of the scheduler.

The first time a data item is used by the data item the counter is decremented by one. A data item is used by a transaction whenever that data item is an operand for the operation performed by the transaction. Next time when the same transaction access the data, the associated counter is decremented. A counter value becomes zero means that the transaction in progress is done with the data. As soon as a transaction is done with a particular data item all locks on the data item are released and it is made available for the next transaction.

The local scheduler maintains the conflict graph for the active transactions. It passes this information to the external software support layer. Since in the altruistic protocol a transaction can commit even if it not in the serializable state the global serialization may be incorrect. To make this protocol compatible with the HDDBS this heterogeneity must be avoided.

External Support of serializable state. The external software layer takes as input the conflict graph of the site and use it to maintain its own conflict graph which will present a serializable view to the global transaction manager.

To understand how the external support of the required states is to be achieved, we should firsts understand the principles of Altruistic protocol clearly. Then we can identify the basic similarities and differences this protocol has with that of 2 PC protocols. Using this information and by getting the information about the states of the transactions in the underlying site, we can support the desired external states for it.

Let us see an example to understand the principle of Altruistic protocol used for advanced applications. More details about the protocol is given in APPENDIX A. Consider a software development environment in which a programming team consists of X and Y. Let Z be a new member who wants to join the programming team. To start with Z wants to familiarize himself with the code of all the procedures of the project. To do this, Z initiates a long transaction (LT), T_Z, that accesses all of the procedures, one procedure at a time. He needs to access each procedure only once to read it and add Some comments about the code; as he finishes accessing each procedure he releases it.

In the meantime, let X start a short transaction (ST), T_X, that accesses only two procedures p1 and p2, in that order, from module A. Assume T_Z has already accessed p2 and released it and is currently reading p1. T_X has to wait until T_Z is finished with p1 and releases it. At this point T_X can start accessing p1 by entering the wake of T_Z. After finishing with p1, T_X can start accessing p2 without delay since it has already been released by T_Z. Now let Y starts another ST T_Y, that needs to access both p2 and a third procedure p3. T_Y can access p2 after T_X terminates but then it must wait until either p3 has been accessed by T_Z or until T_Z terminates. If T_Z never accesses p3 T_Y is forced to wait until T_Z terminates.

Under 2 PC protocol, the transactions T_X and T_Y cannot proceed until T_Z runs to completion. Since T_Z is a LT both these transactions have to wait for a long time. This is not acceptable in advanced applications.

Under altruistic protocol for advanced applications, using the negative access information and early release of read locks, more concurrency is attained. Continuing the example, the mechanism would add p3 to the wake of T_Z by issuing a relief on p3 even if it had not locked it. This allows T_Y to access p3 and thus continue executing without delay.

Assume that an advanced application supporting an altruistic protocol as the one given above wants to participate in the federation of HDDBS. Since this concurrency control does not follow the strict 2 PC protocol which is shown to be the necessary one in order to achieve the global serialization of the

HDDBS [JUHA91], the protocol must be modified to that of a strict 2 PC protocol to support the "source state" and "serializable state".

On the surface these two protocols look lot different. Let us study the similarities and differences between these two protocols. In the above example, if T_Z does not run to completion successfully and is aborted by the system, externally by the user or by the system due to system crashes, in order to ensure the database consistency the system has to undo all the operations of T_X, T_Y and T_Z. This is because T_X and T_Y have used data written by an uncommitted transaction. In a strict 2 PC protocol this situation is avoided by making sure that a transaction read only data read or written by committed transactions. In the Altruistic protocol, in order to achieve higher concurrency this is allowed. The price paid in this protocol is relatively high in the case of system crashes or user initiated abort of the long transactions. However the demand for the higher efficiency outweighs this disadvantage. So the final result of a given set of transactions on a database for both the protocol its the same. In other words, both the protocols preserve data consistency.

In the given example T_X and T_Y enter the wake of the LT T_Z. As soon as they are done with their work they commit without waiting for T_Z to commit. But under [JUHA91] scheme, the requirement of the strong-TM protocol demands that a transaction can commit only if it is in the serializable state (no transaction can commit unless it does not have any active transaction as its predecessor). But in the example for Altruistic protocol, T_X and T_Y are allowed to commit even though they are not in the serializable state.

This heterogeneity in its protocol mechanism prevents it from its participation in the federation [JUHA91]. Our goal is to make it appear to the GCC of the federation as though the altruistic protocol site follows the restrictions of the federation.

The external support layer present to the global transaction manager that the underlying site to be supporting the serializable state. In order to perform this function, the software layer need the conflict graph of the underlying site. In the strong TM protocol [JUHA91] each site needs to maintain a conflict graph and pass this on to the GCC. The GCC maintains a global conflict graph which is the union of all the site graphs of the federation. So we do not need any more information than required by the original model.

The software layer uses the information in the conflict graph and present it to the GCC. Clearly for the external support for Altruistic protocol we have only two cases to consider. First a transaction with no predecessors commits. Second a transaction with an active predecessor commits. There is no problem with the first case. The second case need to be handled differently. We will see now how the external layer can handle both the cases.

When the first active transaction (which has conflicting transactions as successors) or transactions without any predecessors commits, the software layer passes this information immediately to GCC. This is because the transaction is in the serializable state (has no active predecessor and is in prepared state). And appropriate changes are made in the conflict graph of the software layer. If a transaction which has other active transactions as predecessor commits, the software layer stores this information (the implementation details are given in the simulation chapter) and does not pass on this to the GCC. This is so because this violates the serializability state. Whenever an active transaction with no predecessors commits, the software layer check its private memory to see whether its immediate successor have already committed in the underlying site. In that case, appropriate edges are modified in the conflict graph of the software layer to reflect this situation. This will make the commitment of the transactions appear to the GCC as that of following the serializable state. At any stage, the GCC sees only the conflict graph presented by the software layer. The software layer uses the conflict graph of the underlying site, updates the graph when transactions in serializable state commits and stores the transactions which commit when not in serializable state and makes it appear that it commits only when it is in the serializable state. The information about the non conflicting transactions are presented as usual.

Table 1 gives the sequence of action the external software support will follow to present a serializable picture of our example transaction set T_X, T_Y and T_Z.

This intermediate layer is simple and helps to preserve the local autonomy of the site following altruistic protocol. The GTM does not have to care about the heterogeneity of the local concurrency control by this scheme.

The function of the global scheduler is the same as that of model 1. For this model also, the throughput of the system and that of the site 2 was calculated in milliseconds. Several values were calculated by varying the duration of the transactions (from 20 to 50), number of transactions (from 8 to 32). These values were used for the performance comparison study.

Result. The two set of values obtained for the amonut of time taken for completion of the set of operations in both the models by varying the duration of the transaction and the numbers of transactions present in the system were used to plot the histograms.

Performance comparison for site 2. From the Histograms (pages 47, 48 & 49), it is clear that for more longer transactions in the site the model in which the site following Altruistic protocol is better than when the same site followed strong TM protocol. To do the comparison the data were taken at similar conditions. This is because of the advantage of using the altruistic locking scheme for long transactions.

In the original model, this site has to change its protocol to strict TM to participate in the federation. In that case its performance for the local user is affected. We should also consider the number of man-hours needed to change this protocol. The external support eliminates this unnecessary conversion.

From the histogram for local site 2, it is also clear that more the number of longer transactions better the performance of the altruistic protocol. This is the reason for its popularity in the advanced databases.

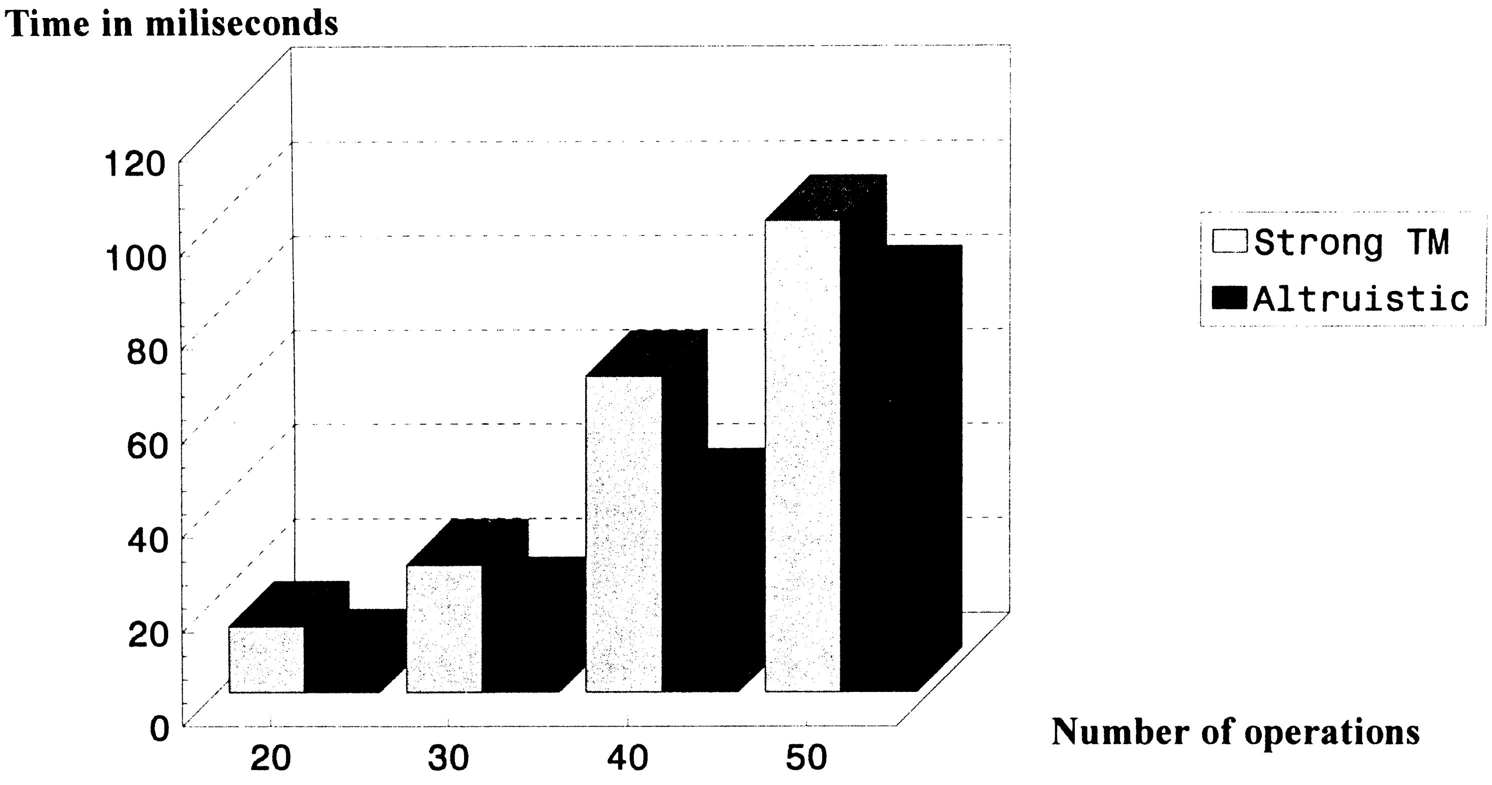
Performance comparison of the MDBS. The histograms (pages 50, 51 & 52) for the overall MDBS shows that the external support for Altruistic protocol resulted in a slight improvement of performance for the overall system. Higher the number of transactions and longer their duration, better is the performance of model 2.

So the introduction of the external support for the altruistic protocol to make it participate in the MDBS did not have any negative effect. However, further reasearch needs to be done before coming to a conclusion.

Feasibility of the project. The feasibility of the simulation project was proved by the successful and correct implementation of the MDBS with a non 2 PC protocol supported externally. The global transactions were executed correctly. The serializability state was correctly supported externally.

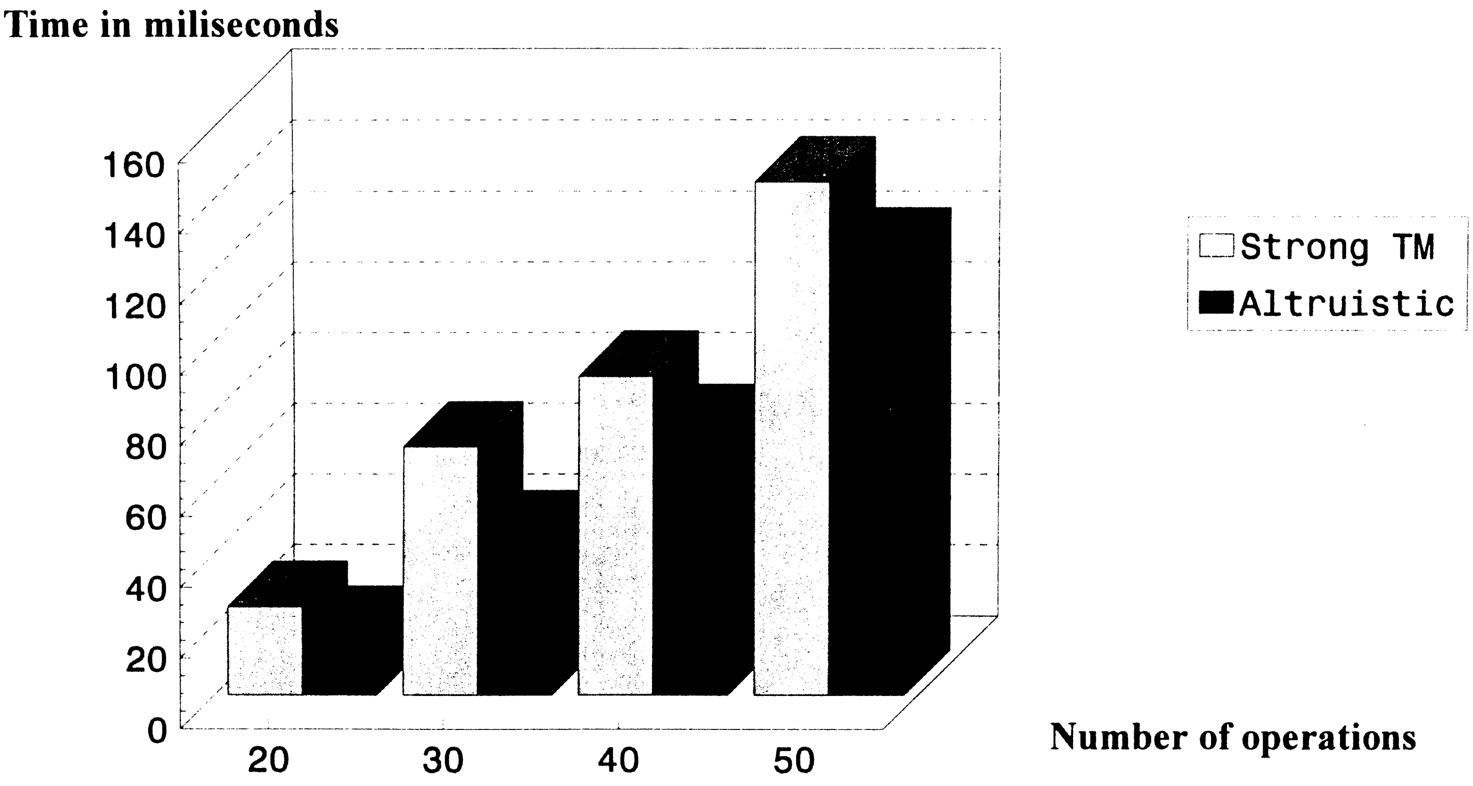
Local autonomy of site 2. Furthermore, the simulation result shows that the modified model maintains local autonomy of Altruistic protocol site.

Performance Comparison Strong TM Vs Altruistic Protocol



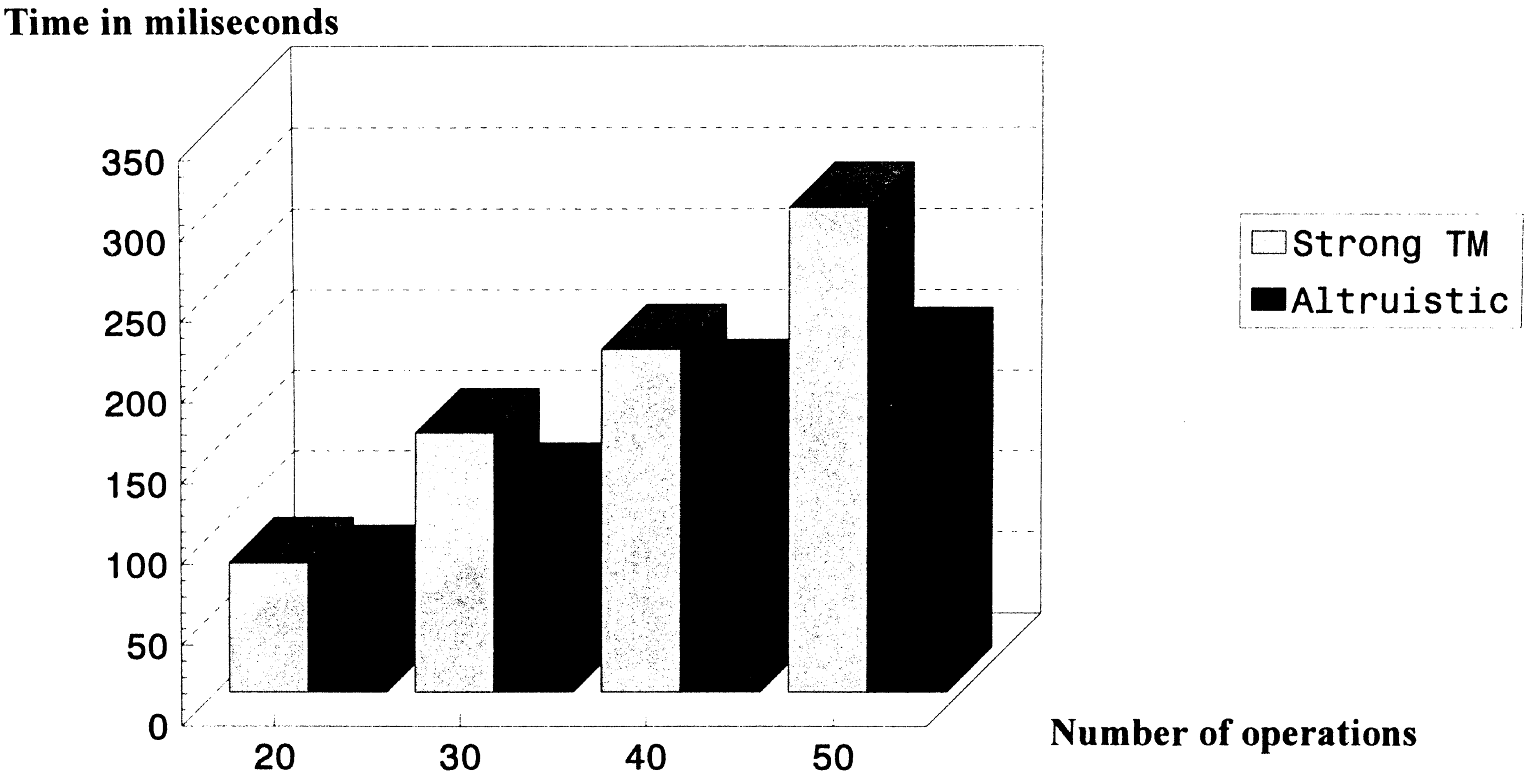
# of transactions = 2

Performance Comparison Strong TM Vs Altruistic Protocol



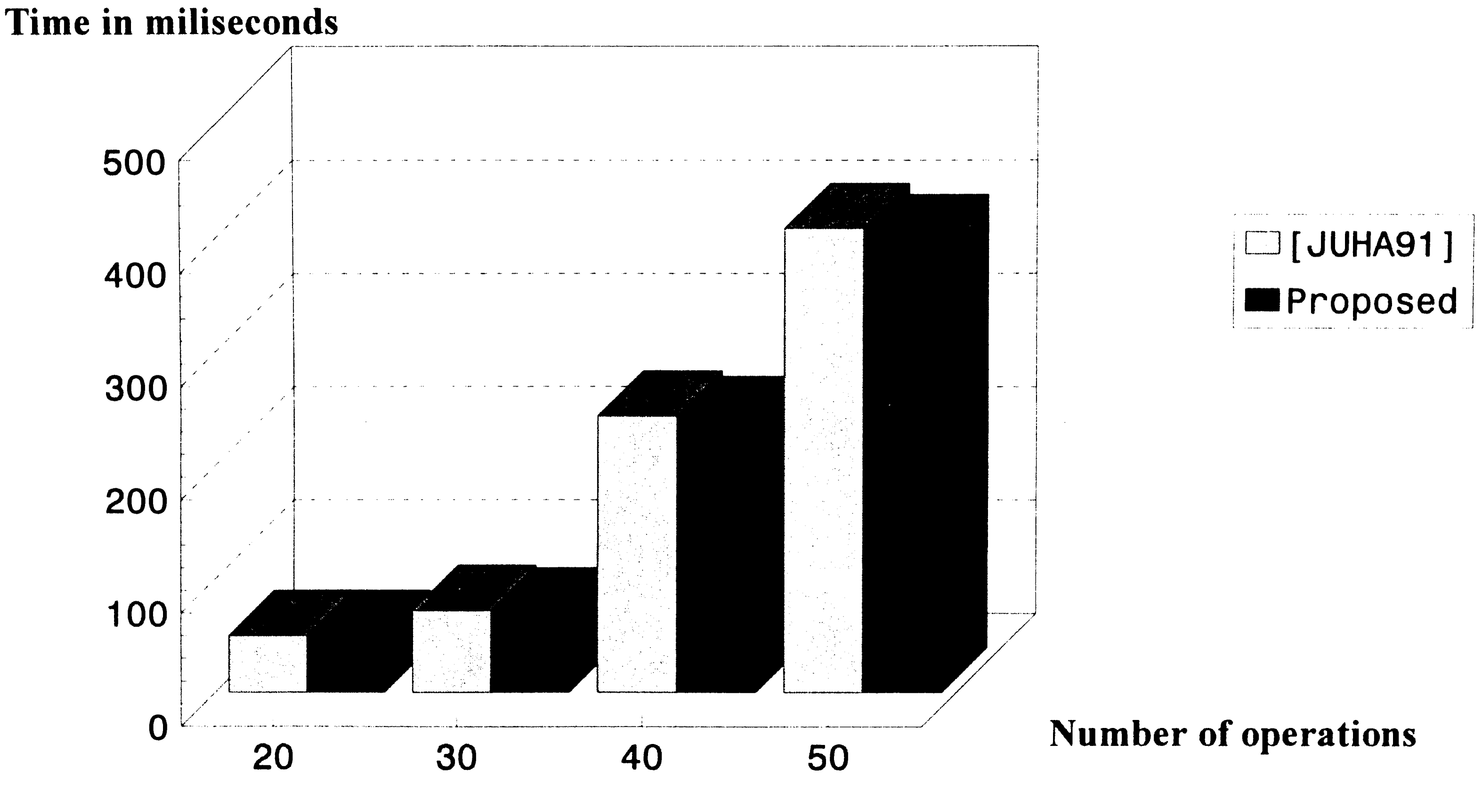
# of transactions = 4

Performance Comparison Strong TM Vs Altruistic Protocol



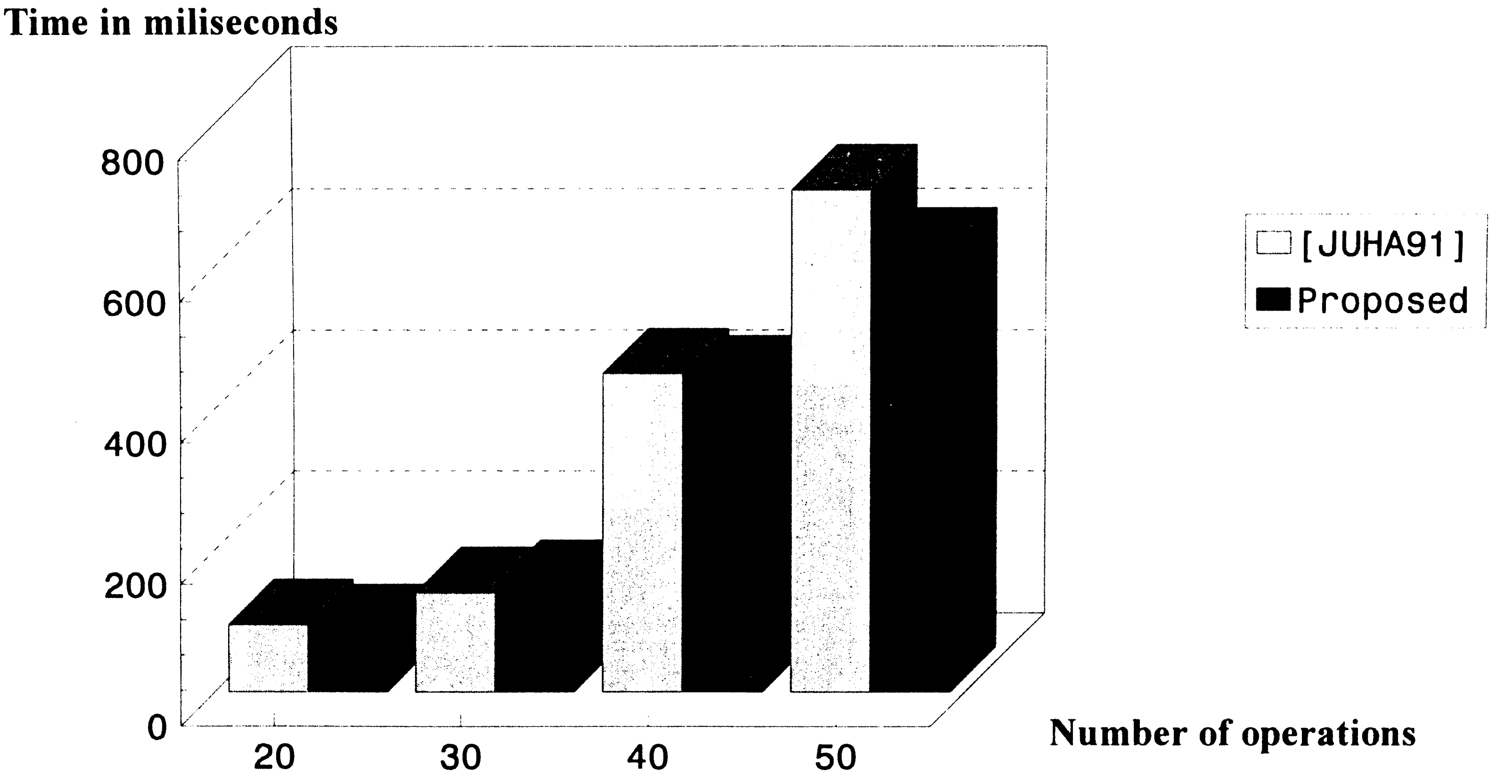
# of transactions = 8

Performance Comparison MDBS [JUHA91] Vs MDBS with Proposed external software support



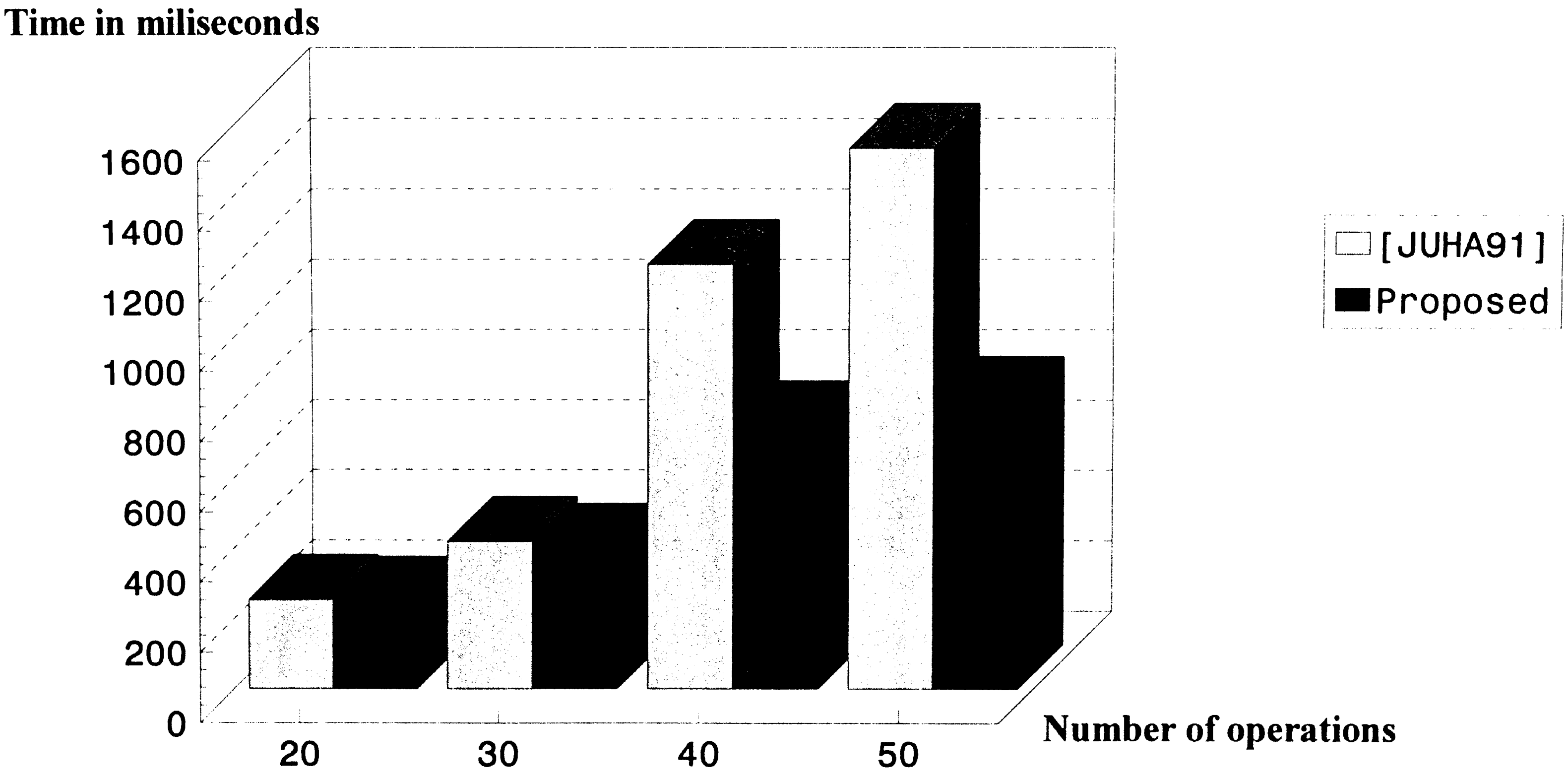
# of transactions = 8

Performance Comparison MDBS [JUHA91] Vs MDBS with Proposed external software support



# of transactions = 16

Performance Comparison MDBS [JUHA91] Vs MDBS with Proposed external software support



# of transactions = 32

CHAPTER 5

SUMMARY AND CONCLUSION

The advanced databases are widely used nowadays. The participation of an advanced database, without any compromise to its local autonomy, will greatly increase the usability of the HDDBS. In the literature review chapter we saw several models for distributed control of HDDBS. Their implementation techniques and drawbacks were discussed.

The desirable properties of the strong TM protocol [JUHA91] makes it one of the best choice for distributed global management of HDDBS. However its restriction on the protocol of the component database compromises local autonomy. The proposed external support of the serializable state for an advanced application overcomes this restriction.

The analytical proof clearly shows that the proposed external support does not violate data consistency of the system. A software external layer to make a site following Altruistic protocol compatible with the federation of HDDBS was successfully implemented. Likewise we believe that the intermediate layer for any other non 2 PC protocol can be implemented externally. This will help to have the advantages of the model proposed by [JUHA91] and the same time for any non 2 PC protocols serializable state can be supported externally.

The external support will play a vital role in an advanced environment where non 2 PC protocols are commonly used. The feasibility of external support for the altruistic protocol was proved. The performance comparison done clearly shows the performance gain of both the local site and that of the HDDBS for higher number of long transactions.

If effort is made to externally support other non 2 PC protocols they too can join the federation of the HDDBS. The external support makes any new addition to the federation can be easily implemented. If the new member supports serializable state it poses no problem and if it does not support it also, it can be done externally. There is no need to make any changes in the global distributed control mechanism.

REFERENCES

1. [ALON87] Alonso, R., Gracia-Molina, H., Salem. K., "Concurrency control and recovery for global procedures in Federated DataBase system", Data Engineering Bulletin, 10:3 (1987), 5-11.

2. [ASER91] Aser S. Barghouti & Gail E. Kaiser., "Concurrency control in Advanced data base applications", ACM Computing surveys, vol 2, 3, (Sept, 1991), pp. 269-317.

3. [BERN81] Bernstein, P. A. Goodman, N., "concurrency control in distributed database systems", ACM Computing surveys, vol 13, 2, (1981), pp.185-221.

4. [BERN87] P. Bernstein, V. Hadzilacos, N. Goodman, Concurrency control and Recoverability in Database Systems, Addision-Wesley, 1987.

5. [BHAR87] Bharat K. Bhargava, Concurrency control & Reliability in Distributed systems, (1987), Van Nostrad Reinhold company Inc.

6. [BRIE87] Brietbart, Y., Silberschatz, A., Thompson, G., "An update mechanism for Multidatabase systems", Data Engineering Bulletin, 10:3 (1987), pp. 12-18.

7. [BRIE88] Breitbart, Y., Silberschatz, A, "Multidatabase update issues", Proceedings of ACM-SIGMOD International conference on management of Data, (1988), pp. 135-142.

8. [BRIE91] Brietbart, Y., Georgakapoulas. D, Ruiskieuricz. M, Silberschatz. A., "On Rigorous Transaction Scheduling", IEEE Transactions on Software Engineering, (Sept, 1991), pp. 954-960.

9. [CALT91] Calton Pu, Avaaraham Leff, and Shu-Wie F. Chen, "Heterogeneous and Autonomous Transaction Processing", IEEE computer, (1991), pp. 64-72.

10. [CATR88] Catriel Beeri, Philip A. Bernstein, & Nathan Goodman, "A model for concurrency in nested transaction systems", Journal of ACM, vol 30, #2, (April 1989), pp. 230-269.

11. [ESWA76] Eswaran, K., Gray, J., Lorie, R., and Traiger, I., "The notions of consistency and predicate locks in a database systems", COMM. Of ACM, vol 19, 11 (Nov, 1976), pp. 624-632.

12. [ELIO85] J. Eliot B. Moss., Nested Transactions: An application to reliable Distributed computing, Research reports & notes on Information systems", (1985), MIT press.

13. [ELNA90] A. Elmagarmid, W. Du, "A paradifm for Concurrency control in HDDBS", Procc. of the Sixth Int. Conference on Data Engineering, Los Angeles, California, (Feb, 1990).

14. [DOND92] Don D. Fisher, Data Structures II: Course notes, Oklahoma State University, (Fall, 1992).

15. [JUHA91] Juha Puustjarvi, "Distributed Management of Transactions in Heterogeneous Distributed Database systems", BIT 31 (1991), pp. 406-420.

16. [KUNG81] Kung H. T., Robinson J. T., "On optimistic methods for concurrency control", ACM Transactions on Database Systems, vol 6, 2 (June, 1981), pp. 213-226.

17. [NAND91] Nandit Soparkar, Henry F.Korth, & Abraham Silberschatz., "Failure-Resilient Transaction Management in Multidatabases", IEEE computer, (1991), pp. 28-35.

18. [LEFF90] Leff A. and Pu. C., "A classification of transaction Processing systems", IEEE computer, vol 24, 6 (June, 1992),. pp. 63-76.

19. [LITW90] Litwin. W., Mark. L., and Roussopoulas. N., "Interoperability of Multiple Autonomous Databases", ACM computing Surveys, vol 22, 3 (Sept, 1990), pp. 267-293.

20. [PAPA86] Papadimoutri C. H., The theory of Concurrency control, Computer science press, 1986.

21. [PRAD86] Pradel, U., Sclageter, G., & Unland. R., "Redesign of optimistic methods:Improving performance and availability", In the proceedings of the second International conference on Data Engineering, (Feb, 1986), pp. 466-473.

22. Pu. C., "Superdatabases for composition of local databases", Proceedings of the fourth International conference on Data Engineering, (1987), pp. 267-274.

23. [SHET90] Sheth A. P., and Larson J. A., "Federated database System for managing distributed, Heterogeneous and Autonomous database", ACM Computing Surveys, vol. 22, 3(Sept, 1990), 183-236.

24. [SILB91] Avi Silberschatz, Michael Stonebraker, Jeff Ullman, "Database Systems: Achievements and oppurtunities", COMM. of ACM, vol 34, 10 (Oct, 1991).

25. [THOM79] Thomas R. H., "A majority consensus approach to concurrency control for multiple copy databases", ACM Transactions on database systems, 4:2 (1979), pp. 180-209.

26. [THOM90] Thomas et. al., "Heterogeneous Distributed Database System for production use", ACM Computing surveys, vol. 22, 3(1990), pp. 237-266.

27. [THOM87] Thompson, Glen Ray., Multidatabase concurrency control, Thesis1987D T471m, Oklahoma State University.

28. [WEIH89] William E. Weihl, "Local atomicity properties: Modular concurrency control for Abstract data Type", ACM Transactions on Programming Languages and Systems, vol 11, 2 (Apr, 1989), pp. 249-282.

29. [YEH91] Yeh, Song-Shen., A concurrency control with the BANG file for distributed database systems, M.S Thesis 1991, Oklahoma State University.

30. [YOAV92] Yoav Raz., "The principle of Committment Ordering", Proceedings of the 18th VLDB Conf., Vancouver, British Columbia, Canada 1992.

APPENDIX

APPENDIX A

ALTRUISTIC LOCKING

Altruistic locking is one of the popular concurrency control mechanism for advanced applications. It is an extension of the basic 2 phase locking algorithm [SALE87]. It makes use of information about access patterns of a transaction to decide which resources it can release. The technique, in particular, makes use of two types of information to meet the demands of advanced applications:

1. Negative access pattern information, which describes objects that will not be accessed by the transaction and

2. Positive access pattern information, which describes which and in what order objects will be accessed by the transaction.

The combination of these 2 informations allow long transactions to release their resources as soon as they are done with them. The wake of the transaction is used to describe the set of all data items that have been locked and then released by a Long Transaction (LT). Releasing of the resource is an conditional unlock operation as it allows other transactions to access the released resource as long as they abide by the following restrictions stated in the protocol below to ensure serializability.

Restriction 1. No two transactions can hold locks on the same data item at the same time unless one of them has locked and released the object before the other locks it. The latter lock holder is said to be in the wake of the releasing transaction.

Restriction 2. If a transaction is in the wake of another transaction, it must be completely in the wake of that transaction.

The basic advantage of altruistic locking is its ability to use the knowledge that a transaction no longer needs access to a data object it has locked and it maintains serializability. Furthermore, if access information is not available, any transaction at any time can run under conventional 2PL protocol.

VITA 2

Mahesh Ram

Candidate for the Degree of

Master of Science

Thesis: SUPPORTING ALTRUISTIC PROTOCOL IN MULTIDATABASE SYSTEM

Major Field: Computer Science

Biographical:

Personal Data: Born in Madras, India, August 8, 1968.

Education: Graduated from D. R. Higher Secondary School, Madras, India, in May 1986; received Bachelor of Science in Computer Science and Engineering from University of Madras, India, in May 1990; completed requirements for the Master of Science degree at Oklahoma State University in May, 1994.

Professional Experience: Programmer/Analyst, APEX Computers, India, May, 1989 to August, 1991.