Multidatabase Performance Evaluation

Y. Breitbart†
Computer Science Dept.
University of Kentucky
Lexington, KY 40506

H. Morales
Amoco Production Co.
Research Center
P. O. Box 3385
Tulsa, OK 74102

A. Silberschatz‡†
Computer Science Dept.
University of Texas
Austin, Texas 78712

G. Thompson
Amoco Production Co.
Research Center
P. O. Box 3385
Tulsa, OK 74102

Abstract
We define a model of a multidatabase transaction management system, and analyze the performance of two concurrency control algorithms operating in such an environment. The first algorithm does not impose any restrictions on either the structure of the concurrency control mechanisms used by local DBMSs, or the type of multidatabase transactions. The second algorithm assumes that each local DBMS uses the two phase locking protocol. The performance results presented here demonstrate that in both cases the concurrent processing of global transactions provides a better throughput than their serial execution. On the other hand, the first algorithm may cause a significant number of global transaction rollbacks for some combinations of local and global transactions. In the case of the second algorithm, we show that the number of global transaction rollbacks is quite small for reasonable multiprogramming levels, and that the algorithm performs almost as well as a distributed homogeneous database system.

1. Introduction
In the last few years, multidatabase (MDBS) concurrency control problems have received significant attention. However, only very few specific concurrency control algorithms were proposed ([ALON87], [ELMA87], [PU87], [BREI88], [BREI90], [LITW89]). Multidatabase concurrency control algorithms proposed to date fall into two basic categories: algorithms that require the multidatabase system to obtain some local DBMS control information (such as a wait-for-graph, a local schedule, a DBMS log, etc.) ([PU87], [ELMA87]), and algorithms that are not using any such information ([ALON87], [BREI88], [BREI90], [LITW89]). Algorithms of the second type make the strongest assumptions about local DBMS autonomy. Namely, that each DBMS integrated by the MDBS operates without the knowledge of either other DBMSs or the MDBS and a global transaction cannot access local DBMS control information. In this paper we study the performance of algorithms of the second type.

The performance characteristics of centralized DBMS concurrency control mechanisms have been extensively studied ([AGRA83], [CARE83], [GOOD83], [TAY84]). Most of these studies concentrated on a performance of either a specific concurrency control algorithm, such as locking, for example, ([RIES77], [LIN82]) or a comparison of different concurrency control algorithms in a centralized DBMS ([AGRA83], [AGRA85]). Performance characteristics of distributed concurrency control algorithms also have been studied ([GARC79], [KOL85], [L87], [CARE86]). However, the performance studies in a multidatabase environment have not been sufficiently addressed. In [BREI84] we reported performance evaluation results for a retrieval-only multidatabase system. We are not aware of any performance studies conducted in a multidatabase environment which permit updates also. The major difficulty in conducting such studies is in trying to simulate an interaction between the MDBS system and local DBMSs that, in general, may use any type of concurrency control mechanism.

In this paper we define a model of a multidatabase transaction management system and analyze performance of two concurrency control algorithms. The first algorithm [BREI88] does not impose any restrictions on concurrency control mechanisms of local DBMSs or the types of multidatabase transactions that can be used. The second algorithm [BREI90] does assume that each local DBMS uses the two-phase locking protocol [ESWA76].

The simulation model used in this paper is influenced by the model originally described by Ries [RIES77] and later modified by Agrawal [AGRA83].
the performance study presented here we address the following question:

How does a multibase database multiprogramming level (a number of concurrently executed global transactions), and a number of database local sites affect a global transaction throughput, a number of global transactions rollbacks that can occur, resources utilization, and an average response time.

The performance results presented here clearly demonstrate that the transaction throughput is larger for a concurrent multidatabase transaction processing than their serial execution for both algorithms in all cases, except the fully replicated multidatabase in the case of the first algorithm. On the other hand, the first algorithm may cause a significant number of global transaction rollbacks for some configurations of local and global transactions. In the case of the second algorithm, we show that a number of global transaction rollbacks is almost negligible for medium sized multiprogramming levels and the algorithm performs almost as well as a distributed homogeneous database system.

The remainder of the paper is organized as follows. Section 2 describes the multidatabase model used in this study. Section 3 outlines the two concurrency control algorithms whose performance is studied here. In Section 4 we describe our simulation model and simulation parameters. The simulation results are presented in Section 5. Section 6 concludes the paper.

2. The MDBS Model

A global database is a collection of local databases distributed among different local sites interconnected by a communication network. A global transaction is an execution of user’s program on a global database. Transactions considered in our model consist of the operations read, write, commit and abort.

To install permanently in the local databases the results of a global transaction, the operation commit is used. If a user decides to abort a transaction and cancel all its changes in local databases, the operation abort is used. The other two operations are read (denoted by r) and write (denoted by w). A read copies a global data item into the user address space and a write causes a new value of the data item to be written onto a local database. We assume that each data item can be read only once by the transaction and if a data item is read and written by the transaction, then a read occurs before a write.

Formally, a transaction is a sequence of operations \( \sigma_1, \sigma_2, \ldots, \sigma_k \), where \( \sigma_k \) is either commit or abort, and each \( \sigma_j \) (0 < j < k), is either read or write. We define the notion of serializable global (local) schedules in the usual manner [BERN87] and use serializability as a correctness criterion for the MDBS and local DBMS concurrency control mechanisms.

We assume that the MDBS software is centrally located. It provides access to different DBMSs that are distributed among various local sites interconnected by a network. The model discussed in this paper is based on the following assumptions:

(1) No changes can be made to the local DBMS software. This means that local DBMSs cannot be modified in a manner that will provide the MDBS with local control information. This assumption is adopted for purely practical reasons. Any attempt on the part of the user to modify the DBMS software results in the vendor’s dropping support of the product. As a result the maintenance costs skyrocket and obliterate all advantages that the MDBS can bring to the user’s organization. Consequently, while the MDBS is aware of the fact that local transactions may run at local sites, it is not aware of any specifics of the transactions and what data items they may access.

(2) A local DBMS is not able to distinguish between local and global transactions which are active at the local site. This assumption ensures local user autonomy. Local and global transactions are getting the same treatment at the local sites. Therefore, global users cannot claim any advantage over local ones. It ensures the greater acceptance of the MDBS system at local sites.

(3) A local DBMS at one site is not able to communicate directly with local DBMSs at other sites to synchronize the execution of a global transaction active at several sites.

(4) Each local DBMS ensures a local serializability and a freedom from local deadlocks.

Thus, the MDBS is the only mechanism that is capable of coordinating global transactions execution at different local sites. However, any such coordination must be conducted in the absence of any local DBMS control information. Hence, the global transaction manager must make the most pessimistic assumptions about the behavior of the local DBMSs in order to ensure global database consistency and freedom from global deadlocks.

The MDBS system consists of the following three major components:

- **Transaction Manager.** The transaction manager (TM) is responsible for the users’ interactions with the MDBS system. For each operation of the user’s transaction, the TM using the MDBS directory prepares all information required to access the data item that the operation refers to. The TM controls
the execution of global transactions. For each global operation to be executed, the TM selects a local site (or a set of sites) where the operation should be executed. In each such site, the TM allocates a server, (one per transaction per site) and the operation is sent to the scheduler for scheduling and further execution at the selected site. Once the TM allocates a server to the transaction, it is not released until the transaction either aborts or commits. A server allocated to a transaction at a local site acts as a global transaction agent at that site. All transaction operations that are to be executed at the site are eventually sent to the server. In submitting transaction operations for execution, the TM uses the following restriction:

*No operation of the transaction (except the very first one) is submitted for scheduling and execution until the TM receives a response that the previous operation of the same transaction has completed.*

- **Scheduler.** The scheduler manages the order of execution of the various read, write, commit, and abort operations of different global transactions. The scheduler receives the next entry from the TM and makes the determination whether the operation should be executed, whether the transaction issuing the operation should be aborted, or whether the transaction issuing the operation should wait until it can be executed.

- **Set of servers.** A server is a process generated by the transaction manager to act as an agent for the global transaction at the local site. Each server is responsible for translating global operations into the appropriate query language operations of the local DBMS, and submitting these operations for execution to the local DBMS. Each time a global transaction operation is scheduled and is submitted for execution, it is eventually received by the server. Results of the operation execution by a local DBMS are reported to the TM.

The general structure of the system is depicted in Figure 1.

Each global transaction is submitted from a central site, where the MDBS system software is located. For each local site at which a global transaction manipulates the local data, the TM generates a subtransaction that is run by the server allocated to the transaction at the local site. Similar processing of distributed transactions in a homogenous distributed environment is considered in System R [LIND84] and Distributed Ingres [STON79].

3. Scheduler Algorithms

In this section we present a brief description of the two concurrency control algorithms whose performance is the subject of study in this paper.

3.1. General Algorithm

The algorithm is described in [BREI88]. It does not make any assumptions about nature of local DBMSs' concurrency control algorithms and uses a transaction graph data structure that defined as follows:

A transaction graph $TG = (V, E)$ is an undirected bipartite graph whose set of vertices $V$ consists of a set of global transactions that are being processed by the MDBS, and a set of local sites. Edges in $E$ may connect only transaction vertices with site vertices. An edge $<T_i, S_j>$ is in $E$ if and only if the transaction $T_i$ has a server at site $S_j$.

The algorithm employs the transaction graph to ensure global database consistency and freedom from global deadlocks in the presence of local transactions. The algorithm works as follows. For each read/write operation submitted to the scheduler, the scheduler attempts to find local site(s) to execute the operation such that an addition of new edges to the transaction graph (maintained by the scheduler) does not create a loop in the graph. If such site selection is not possible, then the transaction is aborted. The aborted transaction is restarted at some later time. Otherwise, the transaction operation is scheduled for an execution at the site whose addition to the transaction graph does not generate a cycle.

The algorithm makes the most pessimistic assumptions about the possibility of a global deadlock. Such assumptions lead to situations in which the algorithm claims a global deadlock, when, in fact, no global...
deadlock has occurred. In a multibase environment with no assumptions about local DBMSs and no knowledge on the part of the MDBS system of a configuration of local transactions at each site, such situations are unavoidable. Thus, in a multibase environment there is a classical trade-off situation: the more autonomy is granted to local DBMSs, the less likely the MDBS deadlock detection algorithm will avoid false deadlocks.

3.2. Two-Phase Locking Algorithm

The algorithm is described in [BREIB90]. The algorithm assumes that each local DBMS uses the two-phase locking protocol, and ensures freedom from local deadlocks. The MDBS takes advantage of this knowledge and submits the global operation of the various transactions in a restricted manner that ensures global database consistency. Unfortunately, global deadlocks still can occur. The algorithm assures freedom from global deadlocks by using a timeout mechanism to detect global deadlocks. In order to describe the algorithm, let us first introduce a notion of active and waiting transaction and a potential conflict graph.

A transaction $T_i$ is active at site $S_j$ if it has a server at $S_j$ and the server is either performing the operation of $T_i$ at the site, or has completed the current operation of $T_i$ and is ready to receive the next operation of transaction $T_i$. A transaction that is not active at site $S_j$ is said to be waiting at site $S_j$, provided that it has a server at the site, and at least one operation of the transaction was submitted to the site. A transaction that is either active or waiting at a local site is called executing at the site.

We assume that each global transaction can be in the waiting status at most at one site. This restriction obviously holds for nonreplicated global databases. It can also hold for a replicated global database as well. In the latter case, if the transaction should execute a write operation on a replicated data item, then local write lock requests should be submitted in sequence. The next site's write lock request is not sent until the local write lock request from the previous site is satisfied.

A Potential Conflict Graph (PCG) is a directed graph with a set of vertices $V$ consisting of all global transactions executing in the system, and a set of edges $E$ such that edge $T_i \rightarrow T_j$ is in $E$ if and only if there is a site at which $T_i$ is waiting and $T_j$ is active.

The algorithm works as follows. For each read/write operation submitted to the scheduler, the scheduler requests local locks to execute the operation. If a lock is granted, the operation is submitted to a local site for the execution and the transaction status at the local site is active. Otherwise, the transaction status at the site is waiting. The scheduler allows the transaction to wait for a local lock until timeout occurs. If during that time the lock is still not obtained, the scheduler checks for a cycle in the potential conflict graph. If the graph contains a cycle, the transaction is aborted and restarted at some later time. Otherwise, the transaction continues to wait until the next timeout occurs.

4. Global Simulation Model

The simulation model for studying the performance of multibase concurrency control algorithms consists of a single global component and a set of local components. A global component consists of a model of a global database, a set of randomly generated global transactions that are executed under the control of the MDBS system, and a model of the scheduler. Each local component consists of a model of a local database, a local DBMS, and a set of randomly generated local transactions that are submitted to the local DBMS outside of the MDBS control.

Local components communicate with the global component via messages. Local DBMS places information about a transaction operation execution on a response queue that is available to the MDBS system. The MDBS submits an operation for execution by placing one or more messages on a message queue that is available to any local DBMS.

4.1. Global Database

A global database is modeled by a collection of global data items uniformly distributed among local sites. Each global data item could be thought of as a relation, possibly replicated among different local sites, managed by different DBMSs. Figure 2 summarizes global database input simulation parameters that include a number of global data items, a number of different local sites, a number of replicated data items, and a number of data items in each local database. We define the replication level of a global database as a percentage of replicated data items. To simulate different replication levels we used two parameters: upper and lower bounds on the number of local sites at which a data item can be located. Replicated data items in the model were uniformly distributed among local sites.

4.2. Global Component Workload Parameters

The performance of any multibase concurrency control system depends essentially on both the workload of a global component of the model, and the workload of each local component. In this subsection we describe the global component workload parameters that significantly affect the MDBS performance.

A fixed set of global transactions generated at the start of the simulation process were circulating continu-
| ITEM_CNT | number of global data items. We used ITEM_CNT = 1000 to simulate a medium size global database |
| SITE_CNT | number of local sites. We conducted experiments for 10, 20, and 30 sites multidatabase |
| ITEMREP | number of replicated data items |
| MAXSITE | maximum number of local sites at which each data item can be replicated MAXSITE = 1, for non-replicated global database; MAXSITE = SITE_CNT, for fully replicated global database |
| MINSITE | minimum number of local sites at which each data item can be replicated MINSITE = 1, for non-replicated global database; MINSITE = SITE_CNT, for fully replicated global database |
| LDBSIZE | a number of data items at the site $S_i$ |

Figure 2: Global Database Input Simulation Parameters

ously through the simulation model as shown in Figure 3. The set consists of 400 different transactions. The size of the set ensures that during the simulation process each transaction circulated through the simulation process approximately 8 - 9 times. One of the model assumptions is that the system is never idle and as soon as a global transaction is completed, there is always another transaction available and waiting to start processing.

We assume that any global transaction in the model cannot write a data item unless it reads it first. We also assume that each global transaction accesses no more than 5% of the global database, and on the average each global transaction accesses 3% of the global database. Each transaction in the system writes no more than half of the read items. Thus, each global transaction in the system does not contain more than 7 operations with no more than two writes among them.

Figure 4 summarizes basic global transaction parameters which include a maximum number of global transactions that can be concurrently processed by the system (multiprogramming level of the system), CPU and I/O times spent by the concurrency control algorithm of the MDBS system for each transaction, a restart delay - a minimal time that a global transaction should wait after the rollback before it can be restarted by the system, upper and lower bounds on a number of global read and write operations for each global transaction in the system.

Initial values for these parameters were selected to approximate a realistic computing environment. The restart delay value was chosen in such a way that ensures a completion of at least one of the executing transactions in the model. In such case we can reduce a number of transactions rollback, since a completion of a transaction in the system, creates new conditions for the restarted transaction that hopefully would eliminate conditions that caused it to abort.

![Global Transaction Simulation Diagram](image)

Figure 3: Global Transaction Simulation Diagram

Each read/write and abort/commit request that is issued by a global transaction is translated into one or more messages that are to be sent to local sites as determined by the MDBS. Thus, a global transaction parameters also include required I/O, CPU, communication times to send a data message from MDBS to a local site, and a number of messages required per one data item in case of read/write operations. Since abort/commit messages are relatively short, we assumed that only one message per site for abort operation and one message per site for each stage of the commit operation is required.

4.3. Local Component Workload Parameters

In a multidatabase environment a global transaction response time significantly depends on the amount of local processing. Local DBMSs process transactions of two types: local transactions generated by local users outside of the MDBS system control, and global transactions whose operations are sent by the MDBS for execution to the local site. Each local simulation model may d'priori have their own local simulation parameters. To simplify the multidatabase simulation model, we assume that input simulation parameters at each site are independent of a local site.

We assume that at each local site there is a constant ratio of local and global transactions. We simulate local transactions at local sites by generating a system of local transactions and circulating them through the local
4.4. Global Transaction Processing Model

The performance of the multidatabase schedulers was studied using a global transaction simulation process shown in Figure 3. A global transaction can be submitted only from the central site where the MDBS system is located. Initially, a fixed set of global transactions is generated, and a subset of $GMAXACT$ transactions is placed on the $ACTIVE$ queue. There is a limit as to the number of transactions concurrently executed in the system. A transaction considered to be executing in the system if it is either receiving or waits for service at either local site or inside of the MDBS model. At any time during the simulation the system maintains $GMAXACT$ active transactions in the system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GMAXACT$</td>
<td>Maximum number of reads in each global transaction</td>
</tr>
<tr>
<td>$MAXGR$</td>
<td>Minimum number of reads in each global transaction</td>
</tr>
<tr>
<td>$MAXGW$</td>
<td>Minimum number of writes in each global transaction</td>
</tr>
<tr>
<td>$MINGR$</td>
<td>Maximum number of reads in each global transaction</td>
</tr>
<tr>
<td>$MINGW$</td>
<td>Maximum number of writes in each global transaction</td>
</tr>
<tr>
<td>$GIOM$</td>
<td>I/O time to prepare a message from MDBS to a local site</td>
</tr>
<tr>
<td>$GPUM$</td>
<td>CPU time to prepare a message from MDBS to a local site</td>
</tr>
<tr>
<td>$GCOMTM$</td>
<td>Time to send a message from MDBS to a local site</td>
</tr>
<tr>
<td>$GMDM$</td>
<td>Number of messages per data item to be sent from MDBS to a local site</td>
</tr>
</tbody>
</table>

Figure 4: Global Component Workload Input Simulation Parameters

The number of generated local transactions can be changed dynamically during the simulation to maintain a constant ratio of local and global transactions. It is reasonable to assume that in the multidatabase environment most of transactions at a local site are local and only very few are submitted by the MDBS system.

A local simulation model is closely related to the model designed by [AGRA85]. Figure 5 contains local workload input simulation parameters that include a local multiprogramming level, upper and lower bounds for a number of local read/write operations, time delay before a local transaction gets restarted after it has been rolled back, and local CPU and I/O times to perform one transaction operation.

Generally, a local DBMS does not know whether a transaction is local, or has been submitted by the MDBS. Results of the global transaction are communicated by the server to the MDBS system. Thus, local component workload parameters should include I/O, CPU and Communication times to send data messages from a local system to the MDBS, and the number of required messages per one data item in case of read/write operations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MAXACT$</td>
<td>Local multiprogramming level, $MAXACT = 15$</td>
</tr>
<tr>
<td>$MAXLR$</td>
<td>Maximum number of reads in a local transaction, $MAXLR = 5$</td>
</tr>
<tr>
<td>$MAXLW$</td>
<td>Maximum number of writes in a local transaction, $MAXLW = 2$</td>
</tr>
<tr>
<td>$MINLR$</td>
<td>Minimum number of reads in a local transaction, $MINLR = 2$</td>
</tr>
<tr>
<td>$MINLW$</td>
<td>Minimum number of writes in a local transaction, $MINLW = 1$</td>
</tr>
<tr>
<td>$LIO$</td>
<td>Local I/O time to perform one transaction operation at a local site, $LIO = 0.15$</td>
</tr>
<tr>
<td>$LCPU$</td>
<td>Local CPU time to perform one transaction operation, $LCPU = 0.07$</td>
</tr>
<tr>
<td>$LRESTRT$</td>
<td>Restart delay at the site, $LRESTRT = 30$</td>
</tr>
<tr>
<td>$LIOM$</td>
<td>Local system I/O time to prepare a message from a local site to the MDBS, $LIOM = 0.035$</td>
</tr>
<tr>
<td>$LCPUM$</td>
<td>Local system CPU time to prepare a message from a local site to the MDBS, $LCPUM = 0.015$</td>
</tr>
<tr>
<td>$LCOMTM$</td>
<td>Time to send a message from a local site to the MDBS, $LCOMTM = 1$</td>
</tr>
<tr>
<td>$LMDM$</td>
<td>Number of messages per data item to be sent from a local system to the MDBS, $LMDM = 2$</td>
</tr>
</tbody>
</table>

Figure 5: Local Workload Input Simulation Parameters

The first transaction on the $ACTIVE$ queue makes its request to the scheduler. If the transaction operation is scheduled, the transaction proceeds consecutively to the I/O, CPU, and $COMMUNICATION$ queues to perform I/O, CPU, and Communication operations, respectively to prepare and send messages to local sites where the transaction operation should be executed. Sending messages to a local site is simulated either by updating the global subtransaction that is already on the $ACTIVE$
4.5. Local Transaction Processing Model

The local transaction processing model is a slightly extended version of the model proposed by Ries and Stonebraker [RIEST77], and further extended by Agrawal [AGRA85]. For the purposes of this simulation, we assumed that each local site uses the two-phase locking (2PL) protocol, specifically, the blocking algorithm described in [GRAY79] to simulate a local concurrency control in a multidatabase environment. The general algorithm, however, does not take this fact into consideration, while the 2PL algorithm does. The global transactions that are to be executed at a local site along with generated local transactions are placed on the local READY queue at the start of the simulation. During the simulation, there is a limit MAXACT on the number of local and global transactions that can be active at the local site. A transaction at the local site is active if it either on the local ACTIVE or I/O, or COMMUNICATION, or BLOCKED queues.

Each read/write operation of a transaction requires a lock on a local data item. If the concurrency control module can grant the lock, then the transaction operation is executed. If, however, the lock cannot be granted, and the transaction waiting for the lock does not cause a local deadlock, then the transaction is placed on the BLOCKED queue. If the transaction's wait for the lock causes a deadlock, then the transaction is aborted at the local site and is placed on the local RESTART queue. Transactions from the local RESTART queue can be restarted only after LRESTR delay. If the transaction has received the lock to execute the operation, it proceeds consecutively through local I/O and CPU queues to perform I/O and CPU operations required to access the local data item. The results of the operation in form of messages are placed on a local COMMUNICATION to be send to a global component site, if required.

The global and local components exchange information on a global transaction execution through a common area known as the RESPONSE queue. Messages prepared by the local concurrency control mechanism are placed on the RESPONSE queue to simulate the message sending from the local site to the MDBS site.

If a transaction request at the local site was either the commit or the abort operation, and it was successfully executed, the transaction is purged from the system and a new transaction is generated and placed at the back of the local READY queue. For the purposes of this study, we simplified this process by distinguishing between local and global transactions. Global transactions were purged from the local system, while local transactions were placed at the back of the local READY queue. In either case, after a transaction has completed, the local system checks first the local RESTART queue...
and then the READY queue to determine whether a new transaction can become active. In addition, the BLOCKED queue is checked to determine whether any transaction from the queue can be unblocked by granting locks to a transaction that were released by the committed or aborted transaction.

![Local Transaction Simulation Diagram](image)

**Figure 6: Local Transaction Simulation Diagram**

### 4.6. Physical Queuing Model

Associated with each transaction operation is a set of messages to be prepared and sent from/to the MDBS. A preparation of the messages requires some physical resources at both the MDBS site and each local site. Both logical models are characterized by three physical resources: CPU, I/O, and Communication.

Whenever a global or a local transaction requests some services described by their logical models, it will use one of these resources. The amounts of I/O, CPU, and Communication used by each transaction operation are specified as input simulation parameters (global or local). The physical queuing model is depicted in Figure 7. The Physical model is a collection of global and local I/O devices, CPU devices and Communication devices.

In this study we assumed that a global and each local component contains a single CPU, a single I/O disk and a single Communication port. The first two resources are used to prepare messages that are used in the exchange messages between global and local sites, while the last resource is used to simulate message sending through a network of global and local sites.

![Physical Queuing Model](image)

**Figure 7: Physical Queuing Model**

Each request submitted to the system is entered first on a I/O queue of the global model, and after being served, is entered at the end of the CPU queue. If any messages have been created, they were placed on the bottom of the Communication queue. For **read**/**commit**/**abort** operations the MDBS creates a single message to be sent to a local site. For a **write** operation the MDBS creates **GMDM** messages to send to a local site new values of a data item that is to be updated. For **write**/**commit**/**abort** operations a local site creates a single message to be passed to the MDBS. For a **read** operation, a local site creates **LMDO** messages containing the data requested by the MDBS.

Physical resource requests queues generally will be served on a first-come-first-serve basis. We do not exclude, however, that local resource queue will assign higher or lower priority to global subtransactions in experiments to clarify the impact of global subtransactions on local transaction processing.

### 5. Performance Results

In this section we describe the results of our performance simulation experiments. The results presented here assume a non-replicated multibase, although our complete results include data pertaining to multibases with different replication level.

In our experiments we measured global transaction throughput, CPU, I/O, and communication times, average global transaction response time, restart ratio, and average number of global restarts per a restarted transaction as a function of a global multiprogramming level, and a number of local sites. A margin of error in our experiments is within 20%.
To measure global transaction throughput, the parameter $TOTSUB$, the total number of submitted transactions, was kept by the system. Also, the parameter $TOTCOMP$, the total number of completed transactions, was kept in the system. Each time a transaction moved from the READY to the ACTIVE queue, $TOTSUB$ was increased. Each time a transaction commits and is placed at the back of the READY queue as a completely new transaction, $TOTCOMP$ was increased. We also measured a percentage of completed transactions out of transactions that were submitted for execution during the simulation process.

For each completed transaction we measured the transaction response time. This time consists of a global CPU, I/O and Communication times spent for all global data items and a maximum of a local response times for each local site where the transaction was active, and, finally, an overhead caused by all transactions restarts due to a global deadlock. During the simulation each global transaction accumulates times spent at local sites and also times that a transactions waits on any of the global model queues. If a transaction is aborted and restarted cumulative CPU, I/O and Communication times that the transaction has spent before the abort are retained and further updated after the transaction is restarted.

A total global transaction response time for a serial transaction execution does not include the time a transaction spends on the RESTART queue, since the transaction cannot enter the global deadlock situation in the absence of concurrently executed global transactions. Therefore, a transaction response time is measured in the absence of concurrent execution of global transactions. However, results of our tests indicate that the number of completed transactions during the simulation period is much larger for a concurrent transaction execution than in case of their serial execution for both algorithms that were simulated.

For each completed transaction we also measured a percentage of transactions that were completed and restarted at least once along with an average number restarts per completed transaction that was restarted at least once. A global transaction is restarted for one and only one reason: a scheduler aborts the transaction due to the scheduler algorithm requirement. If a global transaction is aborted at a local site (by a local concurrency control mechanism) then it is placed on a local restart queue. After local restart delay, the transaction is restarted invisible to the global scheduler. Such an assumption is valid in our model, since we did not consider the case of failure during transaction processing.

Finally, we measured I/O, CPU and Network communication utilizations. These values were computed as a ratio of I/O, CPU, and Network communication times used by completed transactions to total available I/O, CPU, and Network communication time, respectively.

Experiments were conducted for multiprogramming levels with 10, 20, and 30 sites multidatabases for 3000 simulation units. The total number of transactions that have completed during the simulation process as well as a percentage of completed transactions for the general and 2PL algorithms are shown in Figures 8 and 9, and 8A and 9A, respectively. In both cases, the number of completed transaction increases with the increase of the multiprogramming level. It reaches the maximum (in the case of the general algorithm, the maximum is reached for $GMAXACT = 25$ or $GMAXACT = 50$) in the case of the general algorithm a total number of completed transactions oscillates around its maximum value for larger multiprogramming levels. In the case of the 2PL algorithm, however, the number of completed transactions after reaching its maximum value starts to decrease. In both cases, a number of local sites makes a little impact on a number of completed transactions.

There is a simple explanation for these facts. In the case of general algorithm, a significant number of global transactions aborts caused by cycles in the transaction graph overshadows the effects of local deadlocks that may cause the global transaction abort as well. On the other hand, in the case of the 2PL algorithm, there is no possibilities of global transactions aborts, except for the reasons of deadlock (global or local). The large multiprogramming levels of global transactions increase significantly a possibility of such deadlocks. It is interesting to observe that with the increase of a multiprogramming levels, the majority of global deadlocks are false deadlocks and caused by increased response time from local sites that exceeds the predetermined value of the timeout. These results indicate that in the multidatabase environment with each local DBMS using the 2PL protocol, the number of concurrently executed global transactions should be limited in order to achieve larger throughput.

Figures 8 and 9 also indicate that a concurrent processing of global transactions provides better throughput than their serial processing. In case of fully replicated multidatabase, however, the serial execution of global transactions provides better throughput than their concurrent execution in the case of the general algorithm. However, the case of fully replicated databases in a multidatabase environment is highly unlikely. If data would be fully replicated then there is no need to integrate the data under the MDBS, since addition of different data sources does not supply users with any additional information.
Figures 8A and 9A illustrate the percentage of completed transactions for the general and the 2PL algorithms, respectively. In the case of the general algorithm, a percentage of completed transactions rapidly decreases with larger multiprogramming levels. For multiprogramming level equal 25, only about 20% of transactions are completed. It remains around this number regardless of the increase in the multiprogramming level. Our explanation of this fact is the following: With larger multiprogramming levels, and under the conditions of our model, the transaction graph generates a significant number of loops that causes a large number of transactions aborts. Therefore, the number of completed transactions cannot be as much affected by local database aborts as it is affected by the loops in the transaction graph. Apparently the higher multiprogramming levels are an insignificant factor in total number of aborted transactions.

In the case of the 2PL algorithm, the percentage of completed transactions monotonically decreases with an increase of a multiprogramming level. It reflects the fact that with large multiprogramming levels, the 2PL algorithm generates a significant number of false global deadlocks that causes an increase in global transactions aborts.

Figures 10 and 11 illustrate an average response time for the general and the 2PL algorithms, respectively. The response time figures are not surprising. In both cases, the average response time increases with an increase in multiprogramming levels. On the other hand, in the case of the 2PL algorithm, the average response time is approximately a quarter of the average response time for the general algorithm.

In order to measure an impact of a multiprogramming level and a number of local sites at which a multibase is distributed on a total number of global transactions rollbacks, the parameter $TOTRESTRT$ was increased each time a global transaction (that eventually was completed) was aborted due to the global deadlock problem. A ratio of $TOTRESTRT$ to a total number of global transactions completed during the simulation period - $PERRESSTARTSG$ - indicates a relative frequency of transaction aborts caused by the algorithm. Figures 13 and 13 illustrate a behavior of $PERRESSTARTSG$ relatively to a multiprogramming level. In the case of the general algorithm the per cent of restarts increases very fast with the increase of a multiprogramming level. Figure 13 illustrates that approximately 80% of completed transactions are aborted at least once. The significant number of global transactions aborts does not seem to depend significantly on a number of local sites in a multibase. The 2PL algorithm, however, provides much more optimistic picture (see Figure 13). In the situations where the general algorithm causes 80% of global transactions to abort, the 2PL algorithm causes no more than 4% of transactions to abort. It is quite acceptable number of rollbacks for any practical multidatabase system.

Our next results show an average number of global transactions restarts computed as a total number restarts divided by the number of restarted transactions. In the case of the general algorithm (see Figure 14) each transaction can be restarted up to 5 - 6 times depending on the multiprogramming level and the number of local sites. This is a very high number of aborts and in all likelihood can not be tolerated in a practical multidatabase system. In the case of the 2PL algorithm (see Figure 15), very few transactions are restarted more than once.

Our last results pertain to resource utilization of the computer system model used in the simulation model. These results are shown in Figures 16, 18, and 20 for the general algorithm and in Figures 17, 19, and 21 for the 2PL algorithms. Interestingly enough that for both cases the resource utilization is very similar: about 40 - 55% I/O utilization, about 20 - 25% CPU utilization, and about 90 - 97% network communication utilization. The latter result indicates that in the multibase environment is spent on exchange messages between local sites and the MDBS system site for reasonable large multiprogramming levels. It appears that this fact is independent of the type of algorithm being used for the multidatabase concurrency control. These results also very much in accord with [BRE184] where similar results were obtained for the retrieval only multidatabase systems.

The presented results clearly demonstrate an advantage and a practicality of the 2PL algorithm versus the general algorithm. On the other hand, it is hard to expect that in general case where no information available about local concurrency control algorithms any algorithm will perform much better than our general algorithm. This lead us to make two conclusions: one optimistic and one pessimistic. The good news is that in practically important situations multidatabase concurrency control that ensures a global database consistency and freedom from global deadlocks is a viable option. In fact, the 2PL algorithm is implemented in a pilot version of the ADDS system [BRE186]. The bad news is that there is a little hope that without any information about local concurrency control mechanisms any practical multidatabase concurrency control mechanism could exists.

6. Conclusions

In this paper we have attempted to understand the performance characteristics of multidatabase concurrency control mechanisms using a general performance evaluation simulation model to compare the performance of two concurrency control algorithms. We stu
died the impact of global multiprogramming level, and the number of local sites on global transaction throughput, the number of global transactions rollbacks, and resource utilizations for both algorithms. In terms of performance, we found that the 2PL algorithm performs much better than the general algorithm in terms of both global transaction throughput, and the number of global transactions restarts. We also found that the number of local sites has little impact on global performance. Both algorithms exhibit a very similar resource utilization. Our results clearly indicate a general trend: the more information is available to the multidatabase concurrency control the better the performance of the algorithm will be.

References


[PU 87] Pu, C. 'Superdatabases: Transactions across Database Boundaries' IEEE Data Engineering, 1987


[TAY 84] Tay, Y. Suri, R. 'Choice and Performance and Locking for Databases', Proceedings of the Tenth Int'l. Conf. on Very Large Databases, Singapore, 1984