Simulating MDBS Transaction Management Protocols

Ramon Lawrence, Ken Barker, Aruna Adil
Department of Computer Science
University of Manitoba
umlawren,barker,aruna@cs.umanitoba.ca
Technical Report: TR-98-05

August 20, 1998

Abstract

This paper investigates the performance of two MDBS transaction management protocols with different transaction submission characteristics. One algorithm, the Ticket Method algorithm, has an aggressive approach to global subtransaction submission to allow greater concurrency between subtransactions. The other algorithm studied, the Global Serial Scheduler (GSS), is a pessimistic algorithm in that it will not submit subtransactions if there is a possibility they may conflict. We have used a detailed simulation study implementing both algorithms in a realistic MDBS environment to study their performance. The Ticket Method algorithm displayed very poor performance in the studies and did not scale well when presented with higher database loads. The GSS, although it submitted transactions more serially, had much better performance and scalability under all database loads. Thus, this work demonstrates that optimistic protocols suffer surprisingly worse performance than more pessimistic, serial-like protocols, due to the high abort rates and conflicts among global transactions present in optimistic protocols.

1 Introduction

A multidatabase system (MDBS) is a collection of autonomous database participating in a global federation to exchange data. Ensuring consistency of the data in each autonomous database is a difficult task because the global level transaction manager cannot directly access local database objects. Several transaction management protocols have been proposed to allow global transactions access to data across numerous autonomous local databases. This work examines and simulates two such protocols: the Ticket Method algorithm, and the Global Serial Scheduler.

This work contributes to existing work by summarizing and analyzing these two proposed algorithms. It examines their strengths and weaknesses and uses simulation to prove the arguments. A general, MDBS simulator capable of simulating any global transaction management protocol and MDBS configuration has been implemented. This simulation system can be used to investigate new protocols under varying MDBS loads and organizations. Finally, the simulation studies on the two algorithms yield many interesting results including that aggressive subtransaction submission results in worse performance than more serial submission approaches.

2 Multidatabase Architecture

A **multidatabase** is a collection of autonomous databases participating in a global federation that are capable of exchanging data. The basic multidatabase system (MDBS) architecture (see Figure 1) consists of a global transaction manager (GTM) which handles the execution of global transactions (GTs) and is responsible for dividing them into subtransactions (STs) for submission to the local database systems (LDBSs) participating in the MDBS. Each LDBS in the system has an associated global transaction server (GTS) which converts the subtransactions issued by the GTM into a form usable by the LDBS. Certain algorithms also rely on the GTS for concurrency control and simulating features, such as visible 2PC, that the LDBS does not provide. Each LDBS is autonomous so modifications are not allowed. Many proposed algorithms assume the LDBS exhibits certain properties which may violate the principle of full autonomy. Finally, local transactions not under the control of the GTM are allowed at each LDBS as if the LDBS was not participating in the MDBS.

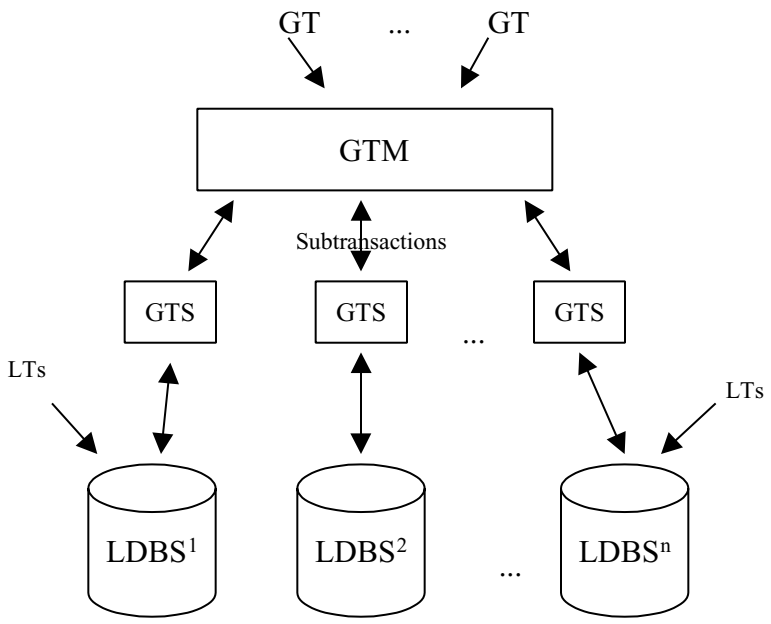

Figure 1: MDBS Architecture

3 Previous Work

There have been several algorithms proposed to handle the MDBS concurrency control problem. The two algorithms examined in this paper are the Ticket Method proposed in [4], and the Global Serial Scheduler proposed in [1].

The amount of published database simulation work is limited. Shasha [6] developed a simulation model for centralized database systems to evaluate the performance when chopping random transactions. A MDBS simulation model based on a closed queuing model was designed in [7] to evaluate the performance of their proposed deadlock detection algorithms. They conclude that their method of deadlock detection is superior than timeout-based methods.

Determining the performance characteristics of different global transaction managers for a MDBS is an important issue because of the growing need for MDBSs in industry. With the large variety of database systems in existence and the current trends of consolidation and intercommunication, many organizations are tackling the problem of database system interoperatability. The interoperability problem can be solved by defining a suitable transaction management protocol and a mechanism for integrating the diverse database schemas. No practical and general solutions have been found to solve this problem. This research aims to tackle the transaction management portion of the interoperability problem. Simulating current transaction protocols will allow greater insight

into their benefits and shortcomings which will lead to better protocols or pragmatic proofs that the consistency problem is too restrictive as defined.

4 Ticket Method Algorithm

The ticket GTM is a method for global concurrency control proposed by Georgakopolous *et al.*[3, 4], and henceforth referred to as the Ticket Method (TM) algorithm. The basic idea behind the algorithm is that normally undetectable local conflicts can be detected by forcing every subtransaction of a global transaction to "take-a-ticket" at the LDBS. Global transactions can then be serialized based on this ticket information.

The ticket GTM algorithm processes a GT (G) by:

- Setting a timeout for G and submitting all its subtransactions
- Waiting for all STs to enter the prepare-to-commit state
- Performing a global serializability graph (GSG) check
- If G passes the GSG check, commit G otherwise abort and restart G
- G is also aborted if any subtransactions abort

The global serializability graph (GSG) check involves constructing a digraph with nodes being "recently" committed GTs. There is an edge from $G_i \rightarrow G_j$ if at least one ST of G_i was serialized before a ST of G_j. To check G, add node G and any edges. If the graph has a cycle, G fails the GSG check and is aborted, otherwise it is committed.

This method has the advantages of preserving LDBS autonomy, simplicity in implementation, and the potential for high concurrency. Unfortunately, the algorithm has numerous disadvantages. The algorithm may be susceptible to GT conflicts and restarts, has no method of guaranteeing transaction execution order, and requires a prepare-to-commit state from all participating LDBS. Also, the ticket in each LDBS may be a bottleneck. Finally, since all STs are submitted with no flow control, the algorithm is very susceptible to a high abort rate, overloading the LDBSs, and a high residence time for GTs. Solving this problem requires tuning the timeout and resubmissions times which is difficult for specific cases and virtually impossible in a large, dynamic MDBS.

5 Global Serial Scheduler

The Global Serial Scheduler (GSS) ensures MDB-serializable schedules by submitting global transactions serially[1]. It determines if all active global transactions access no more than one of the databases accessed by the global transaction (GT) being scheduled. The GT with only one subtransaction will always meet this condition and thus can always be submitted.

The following information is stored and updated. DBMS_set contains databases which are required by a GT but have not been accessed so far. The transactions currently scheduled in a database form an Active_set. The Conflict_set of a global transaction GT_i contains databases in DBMS_set of all the active transactions present from the time transaction GT_i was scheduled until GT_i is complete.

When a global transaction GT_i is initiated, scheduling decisions are made. Since a GT_i with only one subtransaction can always be submitted, it will be submitted immediately. If the DBMS_set of GT_i does not conflict with the Conflict_set of all active GTs, i.e. there is no more than one database in common, then GT_i is submitted, otherwise GT_i goes into a wait queue.

A GT reaches the commit phase after completion of all its subtransactions. DBMS_set and Active_set are updated at the completion of the GT. Then the wait queue is checked to schedule all eligible transactions. To commit the transaction, a cycle check is not required as GTs are submitted with interleaving allowed at only one DB, thereby avoiding the possibility of a cycle.

The most important feature of the GSS is that it indirectly restricts the number of global subtransactions executing at each LDBS. This is because it is highly likely that as the number of global subtransactions currently submitted to all LDBSs increases, the potential for conflicts between the subtransactions increase. As soon as a conflict may be possible between a subtransaction which may be submitted and ones that are already submitted, the subtransaction is not submitted. This provides a form of high-level flow control and insures that the GTM does not overload LDBSs with global subtransactions.

Since conflicts are prevented at submission time, subtransactions at local databases can be committed without a pre-commit stage because their is no need for global-level serialization. Any subtransaction that commits at a LDBS will never need to wait on other subtransactions to commit at other LDBSs. This reduces the time that subtransactions hold locks on LDBS objects and enhances concurrency. However, submission at the global level may be nearly serial as the number of submitted global transactions increases as there are many potential conflicts, even though not all

conflicts may actually occur. The GSS algorithm considers a "conflict" as two global transactions accessing more than one database in common. Note, that a conflict may occur even if both global transactions are read-only. Thus, the GSS could be improved by examining the types of operations (read,write) performed at each LDBS, though this is beyond the scope of this paper.

The major advantages of the GSS is that by controlling global subtransaction submission it insures that global subtransactions do not overload individual LDBSs and guarantees that global transactions are committed in the order that they arrive. Also, since a global subtransaction can commit in an LDBS without a pre-commit state or waiting for other global subtransactions at other LDBSs to complete, the autonomy of each LDBS is more strictly enforced as the execution time of a transaction is only related to the object access in each LDBS. This eliminates the problems associated with a prepare-to-commit state and the potential for global transaction deadlocks and aborts due to global level conflicts beyond the control of the individual LDBSs.

6 Simulation Architecture

The simulation system used to simulate the Ticket Method and the GSS protocols was developed using a C++ library designed at Vrije Universiteit [2] which provides a C++ interface for common simulation requirements including event scheduling, statistics gathering, and random number generation. The simulation system consists of approximately 15,000-20,000 lines of C++ code and is structured using object-oriented techniques to limit code duplication.

Simulating a MDBS requires simulation of both local databases (and their local transactions) participating in the MDBS, and the global-level transaction management protocols handling the submission and execution of global transactions. The following sections briefly describe the simulation system with a more detailed description available in [5]. A short overview of the implementation of the two global transaction management protocols is also discussed.

6.1 LDBS Simulation

The first step in constructing a MDBS simulator involves constructing a simulator capable of modeling a local database system (LDBS). We have developed a generic database simulator designed to handle different transaction managers with varying database configurations. The database simulation is capable of modeling one database and its transactions. Collections of these (local) databases can then be combined into a MDBS simulation. Although it is typical to use a LDBS simulator as

part of a larger MDBS simulation, it is possible to simulate an individual LDBS without considering global transactions.

A local database system (LDBS) consists of objects and their interrelationships and a mechanism for accessing the objects. The transaction management (TM) protocol is the mechanism that the local database uses to mediate access to the data objects. Current protocols include two-phase locking (2PL), timestamping, and optimistic methods. Database users enter queries to the database which vary in size, complexity, frequency, and data usage and are managed by the TM protocol.

A local database is simulated by using two types of simulation processes: transaction processes and the transaction generator process. The transaction generator process runs for the duration of the simulation and generates new transactions at a given arrival rate. Each transaction entering the system is its own process which competes for database objects under the control of the transaction manager. A transaction executes a specific query from a list of possible queries read as input by the system. A query is selected by its probability of occurrence.

Database objects are modeled using a db_object class which is uniquely identified by name and stores the size of the object being accessed and references with other database objects. A db_object provides a locking facility that may be used by transaction management protocols, and gathers usage statistics including percentage of time that it is locked and average queue size for the object. The time to access an object (read or write) is a linear function based on the size of the object and the speed (measured in bytes/sec.) of the LDBS. Lock activities are assumed to take zero time because the time to access a lock in a real system is negligible compared to the time to read/write the relation. By combining database objects in a suitable structure, a LDBS can be represented. For this study, relational databases were simulated by associating each relation with a db_object and storing the set of all relations in a B+Tree.

The transaction management protocol is implemented as a separate C++ class. The transaction management protocol examined in this work is relational strict-2PL. All LDBSs in the MDBS simulations use the relational strict-2PL protocol and provide a visible-2PC for use by GTM algorithms. The TM gathers statistics on the number of transactions (committed and aborted) and throughputs in both transactions/sec and bytes/sec. An important statistic is average transaction execution time (avg_exec) which is a good comparison between transaction managers.

6.2 MDBS Simulation

The multidatabase simulator allows multiple MDBS configurations and different global transaction managers (GTMs). The MDBS simulator is divided into a set of classes which provide the MDBS functionality. Testing different MDBS transaction managers only requires redefining the one class associated with transaction manager specific functions. This allows for greater code reuse and continuity across simulations of different transaction managers.

The basic simulation functionality is provided by the same C++ library as used for the LDBS simulations. It is important to note that the MDBS is simulated using one simulator (one scheduler for the whole simulation) and not by using a number of intercommunicating local database simulators. Intercommunicating local database simulators (schedulers) would allow for increased parallelism and performance as each simulator could be run on a different machine and communicate using IPC. However, the complexities involved in parallelizing the simulation did not warrant the benefits, as the simulation had adequate performance with a single scheduler running on a SPARC machine.

At the global level, there are two simulation processes: the global transaction processes and the global transaction generator process. Both of these types of processes behave similarly to their local database counterparts. The global transaction generation process runs for the duration of the simulation generating new global transaction processes at a set interval. Each global transaction process represents a single GT which will be either executing or waiting for local transaction completion. A process terminates after completion of the global transaction. Figure 2 shows how processes are scheduled in the simulation. In the figure, a dashed line represents initiation of a process at startup time. A solid line represents process scheduling that is always performed, and a dotted line represents other scheduling that may occur.

Simulation initialization involves setting up the MDBS configuration and initializing the simulation processes. A global transaction server (GTS) is created for each LDBS to communicate with the MDBS. The GTS is responsible for managing the global view provided by the LDBS and initializing structures to maintain a communication channel between the two entities. It is also responsible for creating a new local transaction from a global subtransaction and returning the result after its execution.

We assumed that the time to communicate between the global level (MDBS) and the LDBSs through the GTS is zero. Obviously in a real system, this would be unrealistic, but the value is the same across all simulation experiments so it would have a uniform impact. Recall out goal is to

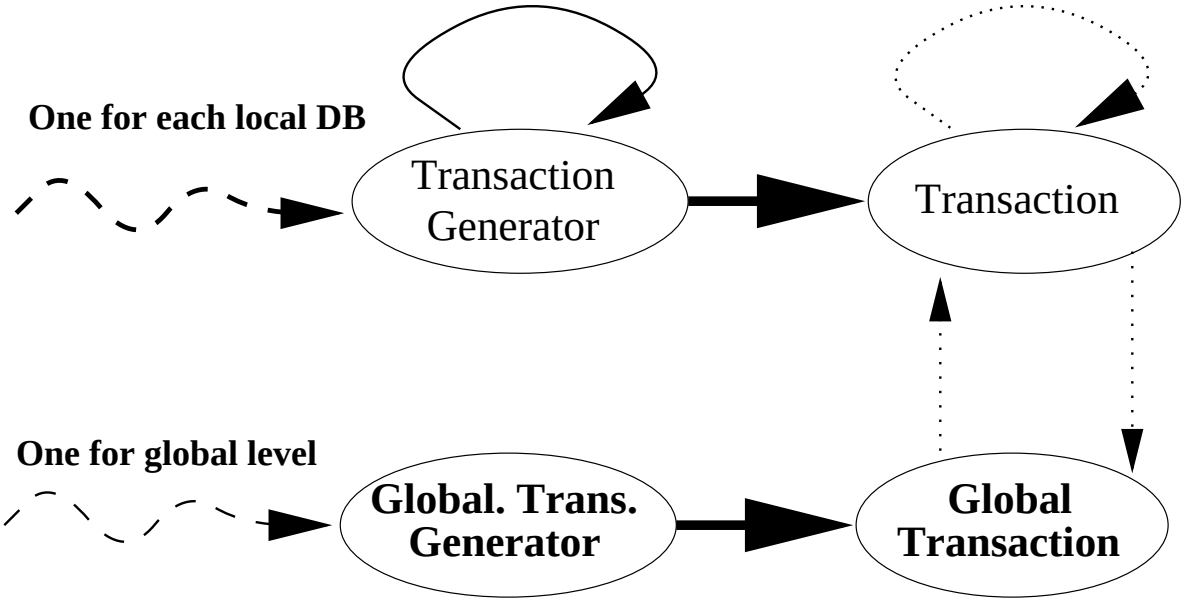

Figure 2: Global Process Scheduling

determine which GTM algorithm performs best under identical communication conditions based solely on the restrictions on performance/parallelism inherent in the algorithm and its management procedures independent of the environment or MDBS configuration.

Obviously, a zero communication time benefits algorithms which do a lot of communication more than algorithms that perform little. Every GTM algorithm must submit and receive results for all STs at least once, so the communication costs would be identical if not for transaction resubmission or prepare-to-commit validation (using prepare-to-commit requires a round-trip message).

Each LDBS provides a list of objects accessible by the global view which may be a subset of the objects in its local view. The global transaction manager provides access to these global objects and insures that the submission of global transactions and their subtransactions are serializable. A GTM has virtual functions for initializing, running, and committing transactions. It also maintains statistics on the number of committed and aborted transactions and the residence time of global transactions in the system.

The local flow of control at each LDBS is uneffected by global transactions and vice versa, but it is important to remember that although each LDBS and the global level are sharing the same scheduler, logically there are separate run-time environments for each LDBS and the global level. Each LDBS has its own local transactions, and global transactions can only work in a LDBS by issuing local transactions. Thus, each LDBS is a separate logical entity with its own scheduling

mechanism. Similarly, the global level scheduling can be considered a separate entity with global transaction generation and execution separate from any local scheduling. Note however that global transaction execution depends on local scheduling decisions as global transactions must submit transactions and retrieve results from local databases.

6.2.1 The Ticket GTM Implementation

Implementing the Ticket Method algorithm based on the MDBS simulation framework is relatively straightforward. The ticket GTM implementation starts by adding a ticket object to each LDBS in the MDBS. A ticket consists of a 1-tuple relation of size 10 bytes. The ticket for each LDBS is also added to the global view.

When initializing global transactions, the GT is registered (assigned a unique id) and its initialization time is recorded. Since the ticket method imposes no restrictions on the order of subtransaction submission, all subtransactions of the GT are submitted immediately. The GT process is then passivated waiting for results from subtransaction completion. However, in addition to the normal operations of the subtransaction, the ticket method adds an operation to increment the ticket counter. This operation was added to the start of each subtransaction of the GT.

When subtransactions complete, the GTS reactivates the GT. If there are still outstanding STs, the GT is passivated again. Otherwise, it attempts to commit. In the commit phase, the GTM determines if there were any global conflicts using the global serializability graph (GSG) test. If there were conflicts, all STs are aborted, and the GT is restarted. Otherwise, all STs are committed.

6.2.2 The Global Serial Scheduler Implementation

The GSS simulation implementation is more complex than the Ticket Method. This is because the GTM must manage the different conflict sets to insure their are no global conflicts instead of relying on the conflicts generated at the LDBS-level by ticket accesses.

Recall that the DBMS_set contains databases which are required by a GT but have not been accessed so far, and the transactions currently scheduled in a database form an Active_set. Also, the Conflict_set of a global transaction GT_i contains databases in DBMS_set of all the active transactions present from the time transaction GT_i was scheduled until GT_i is complete.

When a global transaction GT_i is initiated, scheduling decisions are made. Since a GT_i with only one subtransaction can always be submitted, it will be submitted immediately. If the DBMS_set

of GT_i does not conflict with the Conflict_set of all active GTs, i.e. there is no more than one database in common, then GT_i is submitted, otherwise GT_i goes into a wait queue.

A GT reaches the commit phase after completion of all its subtransactions. DBMS_set and Active_set are updated at the completion of the GT. Then the wait queue is checked to schedule all eligible transactions. To commit the transaction, a cycle check is not required as GTs are submitted with interleaving allowed at only one DB, thereby avoiding the possibility of a cycle.

6.3 Simulation Performance

The overview presented of the simulation system hides some of its complexity and flexibility. At the LDBS-level, the simulation can be designed to handle any database model, including object-oriented models, with appropriate definitions of db_objects and their interrelationships. Also, it is relatively simple to implement different transaction management protocols by defining different instances of the transaction manager class.

The LDBS simulations in this work use the relational model and strict-2PL. The number of relations was kept small(6), so that the different interactions could be studied in-depth. However, there is no limitation on the number of relations that could be in the system, although it does represent considerable effort to define a realistic set of queries which access a large number of relations. The relations sizes (75-11,000 bytes) and the database processing speeds (10,000 bytes/sec.) in the simulation are very small compared to today's standards. However, they can be scaled appropriately as the time to access an object is defined by a linear function based on object size and database speed. Clearly, a relation of size 1,000,000 bytes and a processing speed of 1,000,000 bytes/sec. may be more reasonable values in today's environment, but the effect on the simulation would be the same: the time to access the particular object would be still about 1 second. Thus, object sizes and database processing capabilities can be scaled appropriately to match real-world situations.

The MDBS simulation system is very general. It allows any number of local database which may have different objects, stored under different data models, and accessed using varying transaction management protocols. By appropriately defining the transaction management classes and the GTS class which maps from a global-level view into a local view, any conceivable MDBS configuration could be simulated using this architecture.

However, there are practical limitations on the number of LDBSs that can be used in the simulation. For more than 10 LDBSs, the time to simulate the local transactions is high and

results in long simulation times. It is highly unlikely that a global transaction would access more than 10 LDBSs, so this limitation is not overly restrictive.

7 Determining MDBS Simulation Parameters

This section explains how the simulation parameters were derived. Since there is limited data on the performance and use of MDBS systems, many parameters were chosen as "good estimates" that did not favor either of the two simulated transaction management protocols.

7.1 Simulating a LDBS

A simulation of a local database was performed to validate the system and determine how increasing the transaction generation frequency (database load) affects database performance. These performance statistics were then used to set the database loads of the local databases in the MDBS simulations. The MDBS simulations will use the same local database configurations, local transactions, and database performance measures as this local database. Thus, a local transaction arrival rate (local database load) can be determined from this simulation for each LDBS used in the MDBS simulations.

The local database simulator contains 6 relations varying in size from 75 to 11000 bytes with average relation size of 3400 bytes. The database processing speed is set at 10000 bytes/sec. Obviously, these numbers are too small to be realistic in today's environment but can be scaled appropriately.

There are ten different query types presented to the database. They are divided into five read and five write query types and are a representative set for the possible queries that can be posed to the database. The probability of a read-only query type is 0.8 with the remainder being write query types. Each query type has an associated probability. The probabilities of all read query types sum to 1, as do the probabilities of all write query types. The average number of bytes accessed over all query types is approximately 10500 bytes. Thus, the average time to execute a query should be 1.05 sec.

The local database simulator was run 10 times for 1000 seconds. Each run has no run-up period and is terminated with a hard-close after 1000 seconds. Statistics were gathered on transaction residence time and object lock times, wait times, and queue sizes. As Figure 3 shows, the average transaction residence time increases rapidly as the interarrival time increases beyond 0.3 transac-

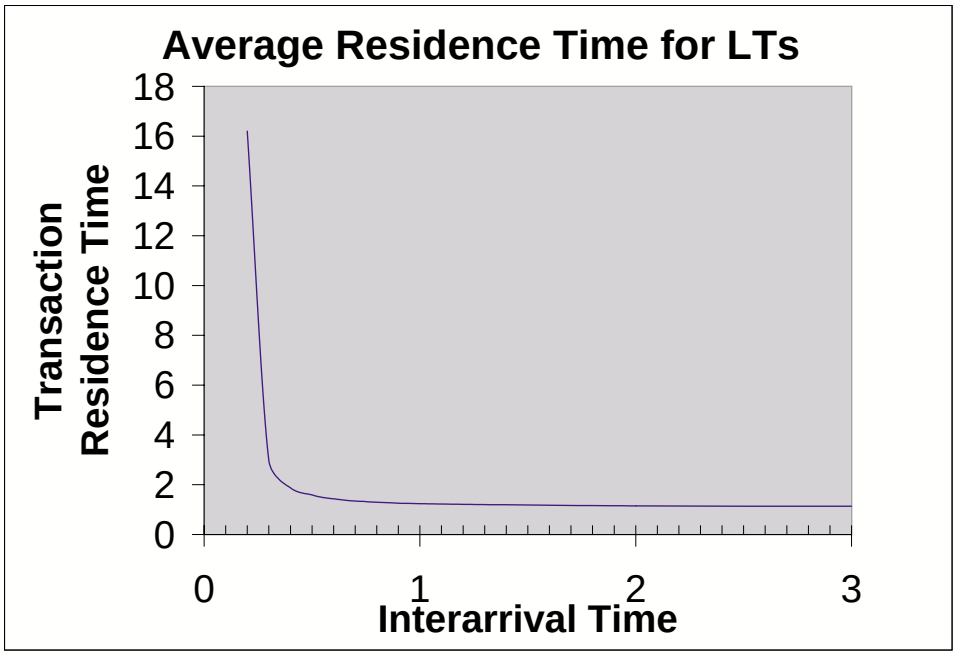

Figure 3: Transaction residence time vs. Interarrival Time

tions/sec. Note that the system does not get overloaded at an arrival rate of 1 transaction/sec as queuing theory would suggest. This is because even though the arrival rate is greater than the service rate, the increased parallelism allowed by executing multiple read transactions simultaneously allows for a higher arrival rate than could normally be achieved.

Based on these results, an interarrival time of 1 second represents a moderate database load. This arrival rate will be used for the local databases when simulating the MDBS.

7.2 MDBS Parameters

The MDBS configuration simulated consisted of 3 nearly identical LDBSs. Local databases 1 and 3 are identical. Local database 2 has the same object sizes as the other local databases, but some objects have different names. Each LDBS has a local transaction arrival rate of 1 transaction/sec. and a processing speed of 10,000 bytes/sec. The set of local query types are the same for each. Thus, the average local query execution time is 1.05 seconds for each local database. Finally, without loss of generality, all local database objects are available in the global view.

The set of global queries to the MDBS consists of 10 query types of varying probabilities. A global query can access from 1 to 3 LDBSs. If all global subtransactions are executed serially, the average global transaction execution time is 3.1 sec and the maximum time is 5.3 sec If all GSTs

are executed in parallel, the average global transaction execution time is 1.5 sec with the maximum being 2 sec.

The MDBS simulator is run 10 times for 100 global transactions. A global transaction must commit before it is allowed to leave the system, so it may be restarted many times until it commits. After 100 global transactions have been generated, the global transaction generator no longer submits global transactions although the local database transaction generators continually submit local transactions. This demonstrates how different GTMs handle the same load. The set of 100 GTs generated will be exactly the same for both GTMs.

Although we would have like to run the simulation for more than 100 transactions, the Ticket Method algorithm suffered poor performance and often would fail when presented with high global transaction arrival rates or long simulation times. Thus, a value of 100 transactions was chosen in order to have a meaningful comparison. The GSS algorithm did not suffer these failings and could be run for 10,000 transactions or more with very high global transaction arrival rates. We must stress that the simulation failed because of the characteristics of the Ticket Method algorithm and not due to software or hardware problems in the simulation implementation.

Statistics are gathered on the number of global transaction aborts, and the mean and maximum global transaction residence times. Also, the number of aborts and average transaction execution time are recorded at each LDBS.

7.3 Scalability and Realism Arguments

The parameters chosen for the MDBS simulations were not intended to directly model real-world systems, however they can be scaled appropriately. Larger relations and faster databases can easily be accounted for in the simulation by increasing the relation sizes and database speed parameters.

The type and frequency of both local and global queries is completely defined in input files which can be varied as needed. Currently, both local and global level queries are read-only with about an 80% probability which tracks real DBMS query mixes. Although the number of relations and databases simulated is small, these values can always be increased. However, adding more relations to each LDBS does not change the validity of the simulation. Rather, adding more relations will tend to expose features of global schedulers related to relation access. Although it is beyond the scope of this paper, it would be possible to model GTM performance based on the number of relations in the system. It is important to note that both algorithms are mostly uneffected by the number of relations in the component LDBSs. In the case of the GSS algorithm, conflicts are at

the LDBS-level so the number of relations in each LDBS is unimportant. In the Ticket Method algorithm, each subtransaction must take a ticket at each LDBS. Thus, regardless on how many other relations a set of global subtransactions contend for, they will always contend for the ticket resource which will be the major bottleneck.

8 Simulation Results

The simulations yielded many interesting results that were unexpected. The most important result is that additional concurrency offered by the Ticket Method algorithm is more of a drawback than an asset. The major results are summarized below.

8.1 Tuning the Ticket Method

The Ticket Method GTM has two parameters critical to its performance. They are the timeout value assigned to a global transaction (g_timeout), and the time for global transaction resubmission after abort (g_resubmit). The algorithm is very sensitive to these parameters. If the g_timeout is set too low, global transactions may abort when they are not in global deadlock. If g_timeout is too high, the system suffers from lower concurrency as the time to recognize global deadlock is high. Also, when g_timeout is too high, local database overloading is possible as global transactions hold local resources from local transactions which queue up waiting for the resources. Since the global transactions tend to access about the same number of bytes in each database, g_timeout is made constant over all transactions.

Defining the global transaction resubmit time is even more complex. If g_resubmit is zero, the GTM may overload local databases by continually resubmitting global transactions which cannot complete. Furthermore, besides taking resources and impeding local transactions, these resubmitted global transactions are more likely to continually abort as the delay times at the local databases increase due to overloading. G_resubmit has been defined to be the square of the number of times the transaction has aborted times a constant factor 10. Even the choice of this constant factor is very sensitive. In testing for a constant value of 10, the average GT residence time was 4 seconds. However, with constant values of 5 and 20, the average GT residence times were 131 seconds and 32 seconds, respectively.

The tuning performed resulted in a g_timeout value of 5 sec, and a g_resubmit = 10*num_aborts^2. Also, it turns out to be better not to limit the growth of g_resubmit (say at some constant 100).

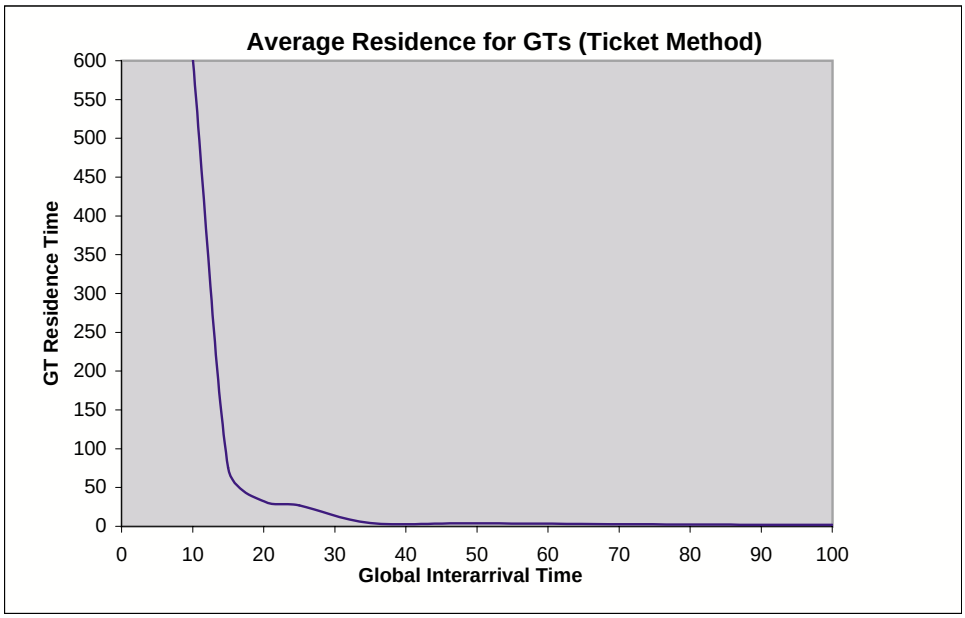

Figure 4: GT residence time vs. Global Interarrival Time for Ticket Method

This tuning is specific to the MDBS configuration. It is highly unlikely that this tuning can be performed in a general and dynamic MDBS. Thus, the Ticket Method GTM suffers from a lack of robustness.

8.2 Ticket Method Results

The Ticket Method algorithm was run on the MDBS configuration outlined 10 times for 100 global transactions. The results gathered indicate the fundamental flaws with the Ticket Method algorithm. Figure 4 shows the residence time of global transactions, and Figure 5 shows the residence time of all transactions at local database 1. Also, Figure 6 shows the number of global transaction aborts that occurred.

The Ticket Method allows all global subtransactions to be submitted as soon as a global transaction is submitted. The ticket in each LDBS, and the GSG check insures global serializability, but there is no global flow control. That is, all global subtransactions are submitted regardless of the load on the LDBSs. This often results in lower concurrency as the LDBSs become overloaded. The overloading causes more global transactions to timeout and abort. However, restarted aborted transactions reenter the system competing for resources again. Without this global-level flow control, the Ticket Method cannot guarantee that transactions are committed in the order they are received which often results in highly variable global residence times and abort rates. This high

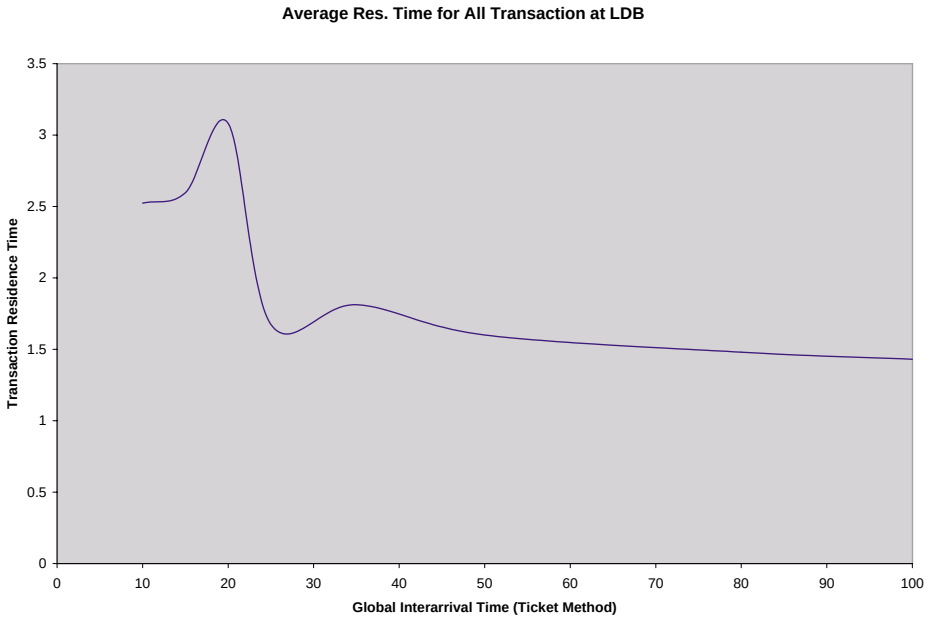

Figure 5: Transaction res. times at LDBS 1 vs. Global Interarrival Time for Ticket Method

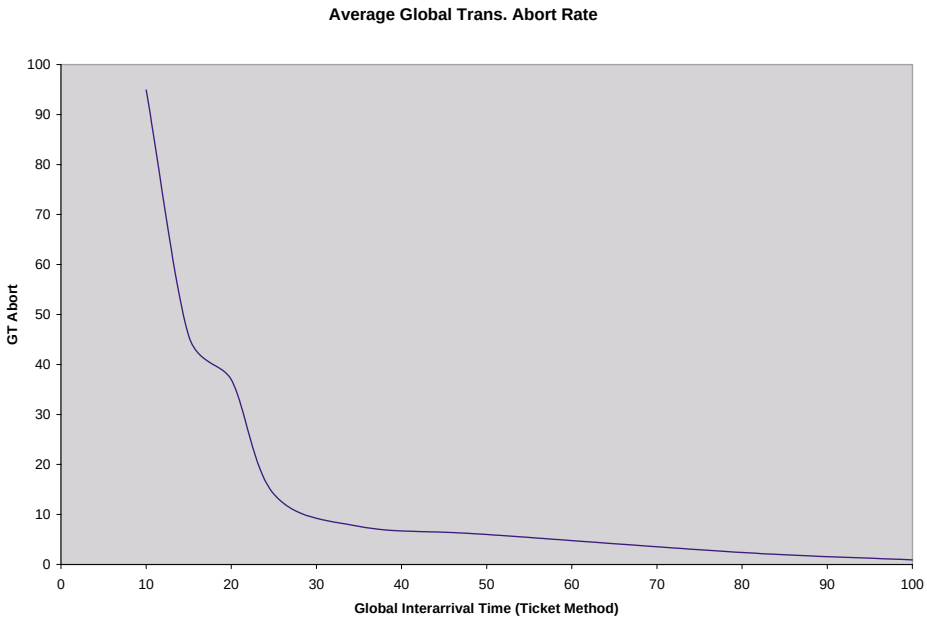

Figure 6: GT aborts vs. Global Interarrival Time for Ticket Method

variability explains the unexpected jump in residence time at LDBS 1 for a global interarrival time of 20 seconds. The Ticket Method algorithm is highly sensitive to timing issues, and in this case, suffered even worse performance than expected.

The tuning of the Ticket Method as described previously is difficult and has a great effect on its performance. Unfortunately, this tuning is not very robust and can easily fail as LDBSs loads increase/decrease, global transaction arrival rates change, or the query mix changes. Thus, the Ticket Method is not very robust and suffers from poor performance in the general case. Even though the algorithm is simple and offers the potential for higher concurrency, the lack of global-level flow control often overwhelms the LDBSs causing many global transaction conflicts and aborts leading to an overall weaker performance.

It is noticeable that even at moderate LDBS loads, the residence times for global transactions are high and highly variable. This is because subtransactions of global transactions are constantly competing with each other for resources, especially the ticket resource. Also, the constant resubmission after aborts often overloads the LDBSs which further exasperates the problem. In total, the Ticket Method algorithm does not provide higher concurrency by submitting all global subtransactions because it often overloads the LDBSs causing long wait times and frequent global transaction aborts.

8.3 GSS Results

The Global Serial Scheduler was run on the MDBS configuration outlined 10 times for 100 global transactions. The results gathered indicate several desirable properties of the GSS algorithm. Figure 7 shows the residence time of global transactions, and Figure 8 shows the residence time of all transactions at local database 1.

The GSS algorithm is deadlock-free and does not cause global transaction aborts. This allows the average and maximum GT residence times to be fairly consistent until the arrival rate passes the threshold of about 5 transactions/sec. The average GT residence time is very low and tends to be around 2 seconds which is somewhere in between the average parallel execution time (1.5 sec) and the average serial execution time (3.1 sec). Also, transactions are committed in the order they arrive which further reduces the average residence time.

At the local level, the GSS algorithm causes no local aborts, and the global residence time at an LDBS does not depend on the global arrival rate. Thus, the total residence time of all LDBS transactions is not affected by the global arrival rate (except a little at very high arrival rates). The

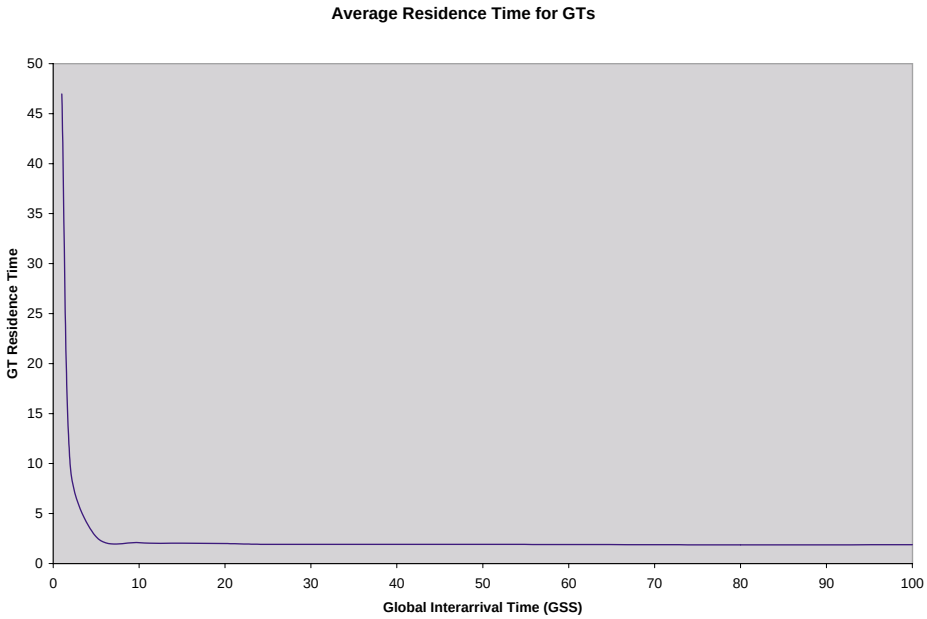

Figure 7: GT residence time vs. Global Interarrival Time for GSS algorithm

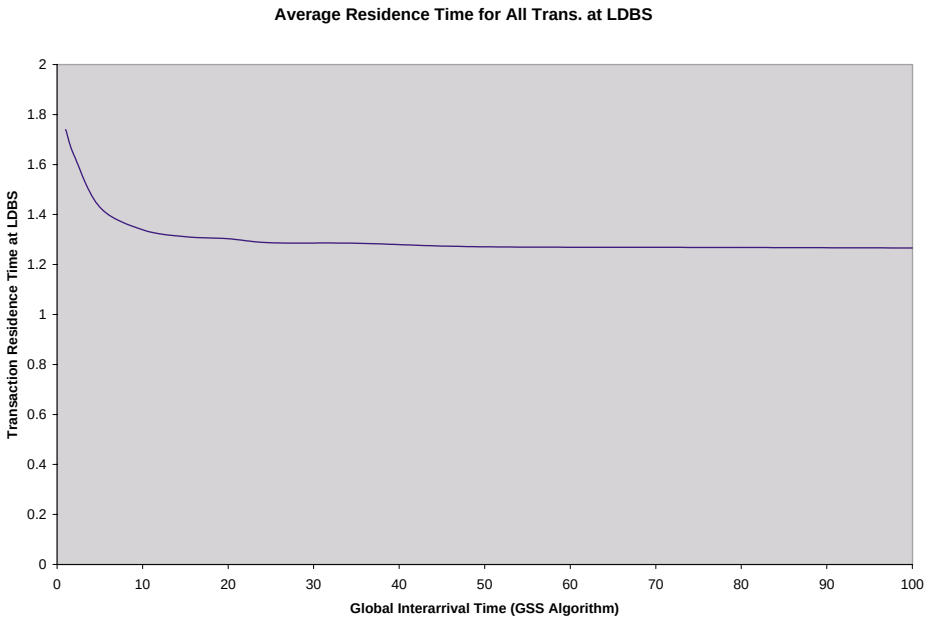

Figure 8: Transaction res. times at LDBS 1 vs. Global Interarrival Time for GSS algorithm

stability of the residence time at the LDBSs arises because the GSS algorithm implements both concurrency control and flow-control at the global level. Global subtransactions are only submitted when no conflicts can arise which limits the number of global subtransactions active at any LDBS, and consequently prevents the MDBS from overloading a LDBS with global subtransactions. Although performance may be limited slightly by executing some STs serially, this performance is more than made up for by limiting the burden placed on the LDBSs by global transactions. Thus, global subtransactions that are submitted to a LDBS can execute faster than if they were competing for the same resources with other global subtransactions. Global-level flow-control is especially important when the global transaction arrival rate is high and when one or more LDBSs are heavily loaded.

8.4 Comparison between the two GTMs

In terms of performance, there is no comparison between the two GTMs. The possible higher concurrency for which the Ticket Method algorithm was designed to allow ends up being a drawback to its performance. It suffers from frequent transaction aborts, local database overloading, and performance loss through global deadlocks. All these factors significantly reduce the concurrency and performance.

The GSS has better performance because it controls the flow of global transactions entering the LDBSs. Although this may reduce concurrency in some situations, it does not cause global deadlock, LDBS overloading, or global transaction aborts. Thus, the GSS has much better performance than it would first appear.

In terms of implementation, the GSS is also much easier to build and configure. It is highly robust and is only slighted effected by increases in LDBS load, query mix changes, or varying global transaction submission rates. The Ticket Method algorithm is highly susceptible to performance concerns if the deadlock detection time and the restart time are not properly configured. Unfortunately, the performance varies greatly even within the same configuration in multiple runs. Also, it is next to impossible to properly configure the system to handle changing MDBS conditions. For example, the GSS can easily handle local transaction arrival rates of less than one second for the given MDBS configuration. The Ticket Method algorithm does not even complete the simulation for such values as it gets stuck in long cycles of global aborts and restarts. The Ticket Method with its high abort/resubmission rate and use of prepare-to-commit is slightly favored in this simulation environment because of the zero communication time over the GSS algorithm which does not abort

GTs or use prepare-to-commit.

9 Conclusion and Future Work

In conclusion, this paper presented two different global transaction management algorithms and argued about their performance in the MDBS environment. A general simulation system was constructed to prove claims about algorithm performance and characteristics. The data gathered by the simulation studies yielded important results on the desirable features of a GTM algorithm.

The simulations showed that the GSS algorithm is superior because it robustly handles different global transaction and local transaction arrival rates. It has better performance in all situations and offers more consistent global transaction residence times. The Ticket Method algorithm is highly dependent on arrival rates and abort frequencies and suffers from poor performance and highly variable global transaction residence times.

The difference between the algorithms is that the GSS algorithm implements global-level flow control by restricting the number of global subtransactions submitted to the LDBSs. This insures that the MDBS does not overload the LDBSs. The Ticket Method submits all global subtransactions which continually compete for resources. This results in more conflicts and overloading of the LDBSs. It is most likely the case that algorithms which do not implement some form of global-level flow control will suffer from the same performance liabilities as the Ticket Method algorithm.

Future work involves extending the simulator to handle different local transaction managers and database models. Also, different global transaction managers can be simulated. It would be interesting future work to determine if other global transaction algorithms which do not provide global-level flow control have the same performance liabilities as the Ticket Method GTM.

References

[1] K. Barker. *Transaction Management on Multidatabase Systems.* PhD thesis, University of Alberta, 1990.

[2] D. Bolier and A. Eliens. Simulation modeling support for discrete event simulation in C++. Technical report, Vrije Universiteit, Departement of Mathematics and Computer Science, October 1995.

[3] D. Georgakopolous, M. Rusinkiewicz, and A.P. Sheth. On serializability of multidatabase transactions through forced local conflicts. In *Proceedings of the Seventh International Conference on Data Engineering*, pages 286–293, April 1991.

[4] D. Georgakopoulos, M. Rusinkiewicz, and A.P. Sheth. Using tickets to enforce the serializability of multidatabase transactions. *IEEE Transactions on Knowledge and Data Engineering*, 6(1), February 1994.

[5] R. Lawrence, K. Barker, and A. Adil. Simulation MDBS transaction management protocols. Technical report, University of Manitoba, Department of Computer Science, March 1998.

[6] D. Shasha, F. Llirbat, E. Simon, and P. Valduriez. Transaction chopping: Algorithms and performance studies. *ACM Transactions on Database Systems*, 20(3):325–363, 1995.

[7] P. Triantafillou. An approach to deadlock detection in multidatabases. *Information Systems*, 22(1):39–55, 1997.