Abstract— The requirements to access and manipulate data across multiple heterogeneous existing databases and the proliferation of mobile technologies have propelled the development of mobile multidatabase system (MDBS). In that environment, transaction management is not a trivial task due to the technological constraints. Agent technology is an evolving research area, which has been applied to several application domains. This paper proposes an Agent-based Transaction Management for Mobile Multidatabase (AT3M) system. AT3M applies static and mobile agents to manage the transaction processing in mobile multidatabase system. It enables a fully distributed transaction management, accommodates mobility of the mobile clients, and allows global subtransactions to process in parallel. The proposed algorithm utilizes the hierarchical meta data structure of Summary Schema Model (SSM) which captures semantic information of data objects in the underlying local databases at different levels of abstractions. It is shown by simulation that AT3M suits well in mobile multidatabase environment and outperforms the existing V-Locking algorithm designed for the same environment in many aspects.

Index Terms— Computer networks, Concurrency control, Database concurrency operations, Database systems, Mobile communication, Mobile agent, Parallel processing, Wireless LAN

I. INTRODUCTION

A continuous increase in amount of data and information overload has led to difficulties in exploring, sharing and manipulating data, and extracting underlying useful information from it. To overcome these problems, various database technologies and architectures have been developed and adjusted for various requirements, ranging from homogeneous centralized database, distributed database, heterogeneous databases (multidatabases -- MDBS) to mobile MDBS. Transaction management is known to be a core functionality of every database management system (DBMS), to achieve high system utilization and data integrity by handling many database transactions concurrently. Such functionality becomes more complicated in MDBS environment because of the following constraints:
- Local databases are heterogeneous (i.e. having different data representation and concurrency control scheme).
- Local databases are autonomous and do not reveal local transaction execution schedule to the global level.

Furthermore, when user mobility comes into the picture, it introduces additional constraints, which make transaction management even more complex.
- Network connectivity is intermittent and unreliable.
- Power constraints due to limited battery power.
- Low bandwidth and disconnections may make mobile global transactions long-lived transactions (LLTs), which would hold resources for longer period of time.

The existing solutions have several shortcomings such as allowing cascading aborts, generating a lot of communication messages and consuming long processing time. Some solutions have restricted assumptions; for example, global transactions are compensatable, and disconnections are planned or predictable.

We propose an Agent-based Transaction Management scheme for Mobile Multidatabase systems (AT3M) that addresses the aforementioned challenges and the deficiencies of the existing solutions.

Table I is a quick reference to the various acronyms used in the rest of our discussion.

Our approach uses agent-oriented design paradigm. An agent is a software program and a mobile agent can halt its execution at one host, migrate to another host in a network, and resumes its execution. We chose the Summary Schema Model (SSM) [10] as our MDBS organization model. It is semantic based hierarchical structure where the leaf nodes are

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT</td>
<td>Global transaction submitted to the global MDBS</td>
</tr>
<tr>
<td>GTAgent</td>
<td>Agent representing a global transaction</td>
</tr>
<tr>
<td>GTC</td>
<td>Global Transaction Coordinator; the GTC for a</td>
</tr>
<tr>
<td></td>
<td>global transaction is the node at which that</td>
</tr>
<tr>
<td></td>
<td>particular global transaction is resolved.</td>
</tr>
<tr>
<td>GST</td>
<td>Global subtransaction</td>
</tr>
<tr>
<td>GST&lt;x&gt;L&lt;y&gt;</td>
<td>A global subtransaction of global transaction x</td>
</tr>
<tr>
<td></td>
<td>which will be executed at local database y</td>
</tr>
<tr>
<td>GSTAgent&lt;x&gt;L&lt;y&gt;</td>
<td>Agent representing a global subtransaction of</td>
</tr>
<tr>
<td></td>
<td>global transaction x which will be executed at</td>
</tr>
<tr>
<td></td>
<td>local database y</td>
</tr>
<tr>
<td>LDB</td>
<td>Local database participating in the MDBS</td>
</tr>
<tr>
<td>LT</td>
<td>Local transaction submitted to a local database</td>
</tr>
<tr>
<td>MDBS</td>
<td>Multidatabase system</td>
</tr>
<tr>
<td>SSM</td>
<td>Summary schema model, Semantic based hierarchical</td>
</tr>
<tr>
<td></td>
<td>structure used as our MDBS organization model</td>
</tr>
<tr>
<td>SSN</td>
<td>Summary schema node, node participating in the</td>
</tr>
<tr>
<td></td>
<td>summary schema model</td>
</tr>
</tbody>
</table>
local databases (local nodes) and other nodes are summary schema nodes (SSNs). Local nodes join the MDBS federation by publishing their local schema. In order to reduce the amount of information held at high level SSNs, increasingly abstract view of the data, known as summary schema, is generated by summarizing the schemas of its child nodes. The relationships between terms in the SSM include synonyms, hypernyms (words with more general meaning) and hyponyms (words with more specific meaning), provided by a thesaurus. A sample schema hierarchy presented in the original paper of the SSM is shown in Fig. 1. In this example, the term “Wage” and “Salary” in node A and B are summarized to the hypernym term “Earnings” at node 4.A.

Our major contributions are highlighted as follows:
- Non-lock based scheme prevents the need to wait for global locks; thus, shorten the global transaction’s processing time.
- No cascade abort.
- The use of autonomous agents allows parallel processing of global subtransactions. The capability of the agents to make local decision avoids acknowledgement messages. As a result, the scheme consumes less network bandwidth and can achieve better processing time, which allows the resources to be released early.
- The use of agent to support user mobility allows disconnected computing (i.e. the user may be disconnected during the transaction processing). The result of the transaction will be saved until the user is reconnected. Thus, the user can turn off the mobile device to conserve energy.
- Simplified local transaction management because of the global order is enforced through out the hierarchy.

This paper is organized into six sections. Section II provides background on transaction management and mobile MDBS. Some related works are described in section III. Section IV details our AT3M algorithm, whereas simulation and its results are presented in section V. Finally, section VI concludes the paper and suggests some possible future works.

II. BACKGROUND

Transaction management involves scheduling transactions and interleave reads and writes operations from various transactions, while leaving the database in a consistent state. As noted in the literature, in order to maintain database consistency and reliability, the transaction management must maintain ACID properties; Atomicity, Consistency, Isolation, and Durability.

A. Serializability Theory

Classical issues in transaction management involve the scheduling of dependent transactions and effects of crashes resulted from interleaving those transactions. For example, when two transactions, T1 and T2 are executed concurrently, serializability requires that the final effect must be equivalent to their serial schedule, i.e. they are executed serially in arbitrary order. Transaction management schemes aim to achieve a schedule that is conflict equivalent to serializable schedule — a schedule that is equivalent to some serial execution of the transactions [13]. It must be conflict serializable (i.e. contains the same set of transactions and the conflicting operations of the committed transactions are in the same order as the serializable schedule). Several well-known concurrency control protocols include Two-phase Locking, Time-Ordering, Multi-Version Timestamp Ordering, and Serialization Graph Testing for transaction management [11].

B. Mobile Multidatabase

A multidatabase (MDBS) deals with multiple pre-existing heterogeneous and independent databases. A transaction, which may be submitted at any participating hosts, may involve access to several databases. There are two levels of control — two layers of transaction management. At the global level, each global transaction (GT) is decomposed into several global subtransactions (GSTs), each of which is to be sent to a local database to be executed as a local transaction. Interleaving global transactions results in the interleaving of subtransactions at the local level. Moreover, the heterogeneity and autonomy of participating local databases allow them to conceal the way they interleave the global subtransactions and local transactions. Thus, several local schedulers now control the global schedule. Figure 2 shows an abstract view of mobile multidatabase transaction management mechanism.

These constraints complicate the transaction management for MDBS in several ways. First, the global transaction manager must support various types of concurrency control schemes used by heterogeneous local databases, without violating their local autonomy. Second, the global transaction manager maintains a global history (GH) of the execution order of global transactions (GTs), while each local transaction manager maintains a local history (LH) of the execution order of both local transactions and global subtransactions executed at the corresponding database [7].

Serializable schedule at local level does not always guarantee serializability at global level. In addition, only conflicts between global transactions (GTs) are visible to the...
global transaction manager — direct conflict [5]. However, the
global schedule is generated from the local schedules of
participating databases. Therefore, it is possible that two
global transactions, which otherwise do not conflict, conflict
over local transactions, namely indirect conflict which is not
visible at the global level.

As a result, multidatabase serializable schedule must
preserve the following serializability rules [5].
1. Every local history (LH) is conflict serializable.
2. For two global transactions GTi and GTj, if an operation
   of GTi precedes an operation of GTj in one LH, all
   operations of GTi must precede any operation of GTj in
   all LHs.

Some examples of transaction management algorithms for
MDBS include Site Graph Method [17], and the Forced Local
Conflict Method [3].

The application of mobile technology and demands to
access the information anytime, anywhere has motivated the
development of mobile multidatabases (e.g., the mobile data
access systems (MDAS)). In this platform, clients could
submit transactions to self-autonomous, heterogeneous, and
potentially mobile databases using wireless connection.
Mobility brings out new issues and new challenges in the
design of transaction management protocols as mentioned in
section I. In this work, we focus on user mobility.

III. RELATED WORK

Previous works on transaction management for mobile
multidatabase attempted to address some of the challenges in a
mobile environment without drastic performance degradation.

Pre-Serialization (PS) [12] is an optimistic approach, which
allows global transactions to build their serialization order
before completing their executions. The PS protocol
decomposes global transactions into vital and non-vital
subtransactions. When the vital portion is completed, all the
vital subtransactions are allowed to commit, the transaction is
toggled, and the resources are released. As noted before, the
PS protocol is an optimistic approach and hence, it checks for
conflict after transaction is committed. Consequently, it could
result in cascading aborts. However, the protocol assumes that
all transactions are compensatable, which makes the overhead
from cascading aborts small. PS was also designed to handle
disconnection. However, it is assumed that disconnections are
predictable. Our approach allows non-compensatable
transactions and unpredictable disconnections.

The V-Locking protocol [5] uses a global locking scheme
with 2PL along with the wait-for-graph scenario to enforce
serializability in a hierarchical fashion. Similar to our work, it
exploits the structure of the summary schema model (SSM).
The submission of global subtransactions to LDBs is delayed
until a lock is granted. A more conservative approach uses this
information to delay global operations until a global lock is
obtained. As with other lock-based schemes, v-locking
algorithm may suffer from deadlocks. Thus, the information in
the global locking table is used to create a global wait-for-
graph to detect or prevent global deadlocks. Nonetheless, it is
more difficult to detect and prevent potential deadlocks
resulted from indirect conflicts because the status of the locks
at LDBs is not visible to GTM. The problem can be resolved
by adding site information to the global locking tables and
constructing implied wait-for-graph which can detect all
potential deadlocks, including false deadlocks. Some
optimizations have been presented to reduce the number of
false deadlocks. The need to wait for a global lock for each
operation in the global transaction before the global
subtransaction can send that operation down to the local level
affects the processing time. V-locking also requires
acknowledgement from local level to progress the global
transaction, leading to a lot of communication messages.
Lastly, false deadlocks would result in excessive global
restarts. Our approach avoids these shortcomings; as our
experimental results will show, AT3M outperforms the V-
locking approach.

Other related work concerns primarily with database
mobility. A correctness criterion called Mobile Semantic
Serializability [1,2] was proposed for mobile database
transaction management in ad hoc networks. A mobile MDBS
is viewed as a collection of disjoint sets objects, each of which
represents a single mobile database and is a semantic unit
(SU). Objects in different SUs are independent. A transaction
is modeled as a sequence of modules; each of which consists of
operations on the objects in only one SU and is an atomic
unit of the transaction. Semantic Serializability is maintained
when the local schedule is serializable and there is no
interleaving within each module. However, the authors did not
detail how to guarantee serializability if there is no in-module
interleaving. This work claimed that it achieved better inter-
transaction parallelism but it did not provide any experimental
results or performance evaluation. Another scheme, Multi-
check out Timestamp Ordering Technique [21] was proposed
for distributed replicated database where nodes of mobile
databases are peers and can be replicated. The scheme handles
IV. AGENT-BASED TRANSACTION MANAGEMENT FOR MOBILE MDBS (AT³M)

In the context of mobile multidatabase system, transaction management faces two major challenges: i) It must conform to the two multidatabase serializability rules mentioned in section II.B, and ii) Its design must take intermittent network connectivity, reducing message traffic, and conserving energy into consideration.

We propose an agent-based mobile multidatabase transaction management scheme (AT³M). Our approach addresses the first challenge by using a time-stamp based ordering for global transactions. The second challenge is alleviated by taking advantage of the agent-oriented programming paradigm. An agent is created to act on behalf of each global transaction, making local decision without user intervention when performing transaction management tasks. Pessimistic approach is chosen to resolve conflicts before the actual execution of the transactions in order to avoid cascading aborts of the global transactions. Agents representing global transactions cooperate to agree on the serialization order to be used at the local level. When a global transaction is completed, the result is delivered to the user. When the mobile client is disconnected, the result of the transaction is not lost but will be stored and delivered to the user when it is reconnected.

A. Assumptions
- The Summary Schema Model (SSM) described in section I is utilized as the underlying MDBS platform.
- Each global transaction is decomposed into global subtransactions by using the query resolution process defined by the SSM [20].
- Local databases are in fixed network and receive transactions from both static and mobile clients.
- Each global transaction has only one subtransaction submitted to a local database.
- Each local database ensures local serializability and resolves local deadlocks.

B. System Design and Architecture

Our SSM-based multidatabase system has a hierarchical structure consisting of several levels of Summary Schema Nodes (SSNs) built on the top of the local nodes which are local databases. The interaction between the node and other external entities is performed through a stationary agent residing in the node called NodeManager. Each SSN maintains a Global Order Table, which keeps the order information of the global subtransactions (GSTs) with which it is involved during the transaction resolution process. The order of GSTs in the global order table reflects the global schedule seen by the SSN. The order information includes GST’s ID, timestamp, and status on whether it has entered the prepared-to-commit stage. The status at the lowest SSNs also records whether the GST has been submitted to the LDB. Figure 3 provides the overview of the architecture of the transaction management over SSM.

When a user submits a global transaction (GT) to the system, at any node, a GTAgent is created to act on behalf of that GT. As part of the transaction resolution process, based on semantic information captured by the summary schema, the GTAgent is launched to the designated Global Transaction Coordinator (GTC). A GTC is recognized as the lowest SSN, which semantically contains related information needed by the GT. For example, in Fig. 3, assume that GT1 is submitted at node 1.A and will be executed in LDB2, LDB3 and LDB4; the GTC of GT1 would be node 1.A. The GTC is where the GTAgent starts to resolve (decompose) the GT. The resulting global subtransactions (GSTs) are also represented by agents, called GSTAgent, which are dispatched by the GTAgent to the lower SSNs. Each GSTAgent is tagged with the ID of the LDB at which its GST will be executed. In this example, the resulting GSTAgents will travel from node 1.A to LDB2 through node 2.B and 3.B, to LDB3 through node 2.B and 3.C, and to LDB4 through node 2.C and 3.D.

Conflicts between global subtransactions are resolved during their propagation down to the local level in accordance with the following timestamp ordering rules.
1. Each GT is uniquely identified by the ID of the SSN that is its GTC and the time at which it is resolved.
2. When a GT is resolved at a GTC, all of the global subtransactions (GST) represented by GSTAgents will have the same timestamp from the GTC upon their creation.
3. The NodeManager at the Summary Schema Node (SSN) assigns a timestamp to the GST on its arrival at the node. At each SSN, GSTs of the same GT will have the same timestamp although they arrive at the different time.
4. When each GST is given the timestamp, an entry for it is inserted to the global order table. Then, the GSTAgent will be given the global order, which is an ordered list of the ID
of all GSTs preceding it in the global order table. The GSTAgent will carry this order information to the next SSN it will visit. The current SSN determines the next SSN for the GSTAgent.

5. When the GSTAgent arrives at the next SSN, the information in the global order it is carrying can be included in the global order table of the next SSN. By this means, the global order information carried by one GSTAgent can be transferred to another GSTAgent that arrives the SSN after it via the SSN’s global order table. With this knowledge, if GSTAgenti arrives at the SSN at level k before GSTAgentj, thus, has smaller timestamp and results in the global order GSTi \rightarrow GSTj; but GSTAgenti arrives at the next SSN at level k+1 (which GSTAgentj must also visit) before GSTAgenti, it will wait for GSTAgenti, before being assigned a new timestamp and inserted to the global order table to preserve the global order GSTi \rightarrow GSTj. Note that the Global Order carried by each GSTAgent becomes more specific to the target LDB as it moves closer to the data.

Since each SSN issues timestamp value independently, time synchronization is not a concern. As the global serialization order is determined before the subtransactions are actually executed at the local databases, the global subtransactions that arrive the local databases are global conflict-free if they reserve the global serialization order agreed during the transaction resolution.

From our running example using Fig. 3, the following scenario illustrates our algorithm.

- GSTAgent1 representing GT1 dispatches GSTAgent1L2, GSTAgent1L3 and GSTAgent1L4 to LDB2, LDB3 and LDB4 respectively. Each of which is given the same timestamp from node 1.A.
- GSTAgent1L2 and GSTAgent1L3 migrate to node 2.B, where as GSTAgent1L4 migrates to node 2.C.
- Either GSTAgent1L2 or GSTAgent1L3 that arrives at node 2.B before the other will add both GST1L2 and GST1L3 to the global order table at 2.B. Both are given the same timestamp, say TS_{2B1}. If there is no other entry prior to their entries in the global order table, the global order given back to them would contain only their own ID.
- Assume another global transaction GT2 having node 2.B as its GTC. Its GTAgent2 dispatches GSTAgent2L2 to LDB2 and GSTAgent2L3 to LDB3. Both GSTAgents are given the same timestamp from node 2.B, say TS_{2B2} (>TS_{2B1}).
- On the creation of GSTAgent2L2, the NodeManager at node 2.B observes that its next SSN will be the same SSN as GSTAgent1L2’s and then gives global order GST1L2 \rightarrow GST2L2 to it.
- Both GSTAgent1L2 and GSTAgent2L2 travel to node 3.B. If GSTAgent2L2 arrives node 3.B before GSTAgent1L2 and gives its global order (GST1L2 \rightarrow GST2L2) to the NodeManager to merge to the existing global order table at node 3.B. As it is shown in the global order that GST1L2 comes before GST2L2, the GSTAgent2L2 will have to wait until GSTAgent1L2 arrives, adds GST1L2 to the global order at the node, and receives its timestamp, say TS_{3B1}.

Then, GST2L2 would be added to the global order after GST1L2, and GSTAgent2L2 would receive TS_{3B2} (>TS_{3B1}) to preserve global order GST1L2 \rightarrow GST2L2. The same scheme occurs at node 3.C at which GSTAgent3 and GSTAgent3G would visit to preserve the global order GST1L2 \rightarrow GST2L2.

At the local level, NodeManager at each LDB will ensure that the defined global order is respected at each local database. We proposed three different approaches to work with LDBs with different concurrency control schemes.

1. Timestamp-ordering: As the order in which GSTs must execute is defined by the timestamps assigned to them at the global level, LDBs using timestamp ordering as their concurrency control scheme can directly utilize the existing timestamps. NodeManager maintains the LDB’s global schedule. When a GST agent arrives, the global order carried by the agent is merged to the existing global schedule. The NodeManager guarantees to submit GSTs to the local database in the global order. If the arrival of GSTj results in global schedule GST1 \rightarrow GSTj but GSTj has not been submitted to the NodeManager, it will wait for the arrival of GSTj and submit GSTj to the LDB before GSTj, to ensure that GSTj always receives lower timestamp from the local transaction manager than GSTj.

2. Rigorous Concurrency Control: It has been shown that when the LDBs produce rigorous schedules or at least recoverable schedule, only maintaining uniform order in which the GSTs commits is sufficient to ensure global serializability [18,19]. The strict two-phase locking protocol (S2PL), which is used in most systems also produces rigorous schedule. An example of the scheme utilizing this property is Implicit Tick Method (ITM) [3]. ITM observes the order which the GSTs enter prepare-to-commit state. If the order produced by all LDBs is not identical, it will restart all involved global transactions. Nevertheless, in our approach, all NodeManagers already agree on the global order. Thus, when the NodeManager receives a prepare-to-commit from the GST, it can immediately determine whether the prepare-to-commit operation of that particular transaction would violate the global order. If the GST performing prepare-to-commit is not the next GST that should prepare-to-commit based on the global order held by the NodeManager, it will locally restart only that GST. Although this might be an unnecessary restart, it is less expensive than restarting the entire GT. To restart the GST with wrong prepare-to-commit order, say GST_{wrong}, the NodeManager will hold the GST_{wrong} for a threshold period of time proportional to the number of GSTs which should enter prepare-to-commit state before the GST_{wrong}, say its predecessors. If all of the predecessors of the GST_{wrong} have finished before the threshold period ends, the GST_{wrong} will prepare-to-commit after them; otherwise, it is assumed that the GST_{wrong} indirectly conflict with its predecessors and should be aborted and restarted. It is also possible that a
large number of GSTs are ready to prepare-to-commit but are waiting for a single GST. In this case, when the number of these GSTs reaches a threshold value, the GSTAgent of the blocking GST would return to its GTAgent and request for a global restart. Thus, other waiting global GSTs can process to prepare-to-commit.

3. Other Concurrency Control Schemes: For fully autonomous LDBs whose local concurrency control scheme is globally unknown, forced conflict method is a practical solution [3]. It applies take-a-ticket scheme to force additional conflict, which reflects the serialization order based on the ticket’s value. Provided that every local transaction manager always produces serializable schedule, by controlling the order in which the GT take a ticket, the NodeManager is able to reserve the defined global serialization order. The global order is always respected even though the global transactions do not conflict. Indirect conflicts which are invisible to the global level will not affect the correctness of our algorithm. Thus, our approach can address both direct and indirect conflicts. In any of the three aforementioned approaches, if the local transaction manager aborts one GST, its corresponding GSTAgent will inform its GTAgent to abort its other GSTs to preserve atomicity. The GT will be removed from the global order, while other succeeding GTs in the schedule remain unaffected. The aborted GT will be re-submitted and obtain a new position in the global order.

C. Proof of Correctness

Lemma1: Global order which the NodeManager receives from the GST always synchronizes with the global order it maintains.

Proof: Proofing by contradiction, let the global order at the NodeManager be GST_1  GST_2, which depicts that GST_1 receives smaller timestamp than GST_2 at the SSN above the NodeManager and at the NodeManager. If the NodeManager later receives a global order contains GST_1  GST_2  GST_3 from the GST, it means that GST_3 arrives the SSN above the NodeManager after GST_2 (otherwise, it should have included in its current global order). However, GST_3 has lower timestamps than GST_2; thus, violates timestamp ordering rule of the summary schema node. Thus, case i) Cannot occur. ii) Let the global order at the NodeManager be GST_1  GST_2. Later, the NodeManager receives a global order contains GST_2  GST_1 from GST. It means that there are at least two GSTs receive different global order from the same SSN. This case cannot happen because they received information from the same global order. The proposed scheme does not violate local autonomy because it does not require any modification to the local systems.

Lemma2: When local transaction manager uses timestamp ordering concurrency control protocol, by submitting GSTs to the local transaction manager in the same order as the global order, the NodeManager maintains global serializability.

Proof: For any timestamp order based local transaction manager, the serialization order of the transactions submitted to it is determined by the timestamps the transactions received at upon their arrivals. Thus, when the GST are submitted to LDB in the same order as the common global order, the local serialization order of the LDB is the same as the global order, which in turn depicts the global serialization order.

Lemma3: When local transaction manager uses rigorous concurrency control scheme, NodeManager guarantees to maintain global serializability.

Proof: Based on the proof that the global serialization order under rigorous concurrency control is depicted by the order which the GSTs commit. Each NodeManager ensures that order conforms to the global order maintained by all other NodeManagers. If the global order contains GST_1  GST_2, the NodeManager ensure that GST_1 reports prepare-to-commit and is committed before GST_2. Thus, the NodeManagers produce local history that is synchronized with the global order, leading to global serializable schedule.

Lemma4: When local transaction manager uses any concurrency control protocol, NodeManager guarantees to maintain global serializability using Forced Conflict Method.

Proof: Forced conflict method guarantees that local history produced by the local transaction manager follows the order of the ticket value. The NodeManager issues the ticket to the GSTs according to the commonly agreed on global order. Therefore, it produces local history that is synchronized with the global order, preserving global serializability.

V. Performance Evaluation

The proposed system was simulated and compared against the V-Locking algorithm because i) To our knowledge, it is the most recent concurrency control scheme for mobile MDBS which local databases in fixed network and ii) It has been shown to outperform Potential Conflict Graph (PCG), forced conflict, and site-graph algorithm [5]. The simulator was developed using SimJava 2.0 [15]. The mobile multidatabase system consists of 7 local databases. Each of which contains 100 data items, 20 items are hot-spot, which is a large number of GSTs are ready to prepare-to-commit but are waiting for a single GST. In this case, when the number of these GSTs reaches a threshold value, the GSTAgent of the blocking GST would return to its GTAgent and request for a global restart. Thus, other waiting global GSTs can process to prepare-to-commit.

A. Simulation Parameters

Each simulation runs for 1000 time units. The global system and local system parameters used in the simulation are presented in table II and III respectively. We chose to simulate rigorous concurrency control as it is the most commonly used. Most of the simulation parameters we used are identical to those used in the simulation of the V-locking algorithm [5], except that we allowed more operations per GT and included hot spot. Parameters for agents were from measurements using IBM’s Aglets agent platform.
B. Simulation Results

We use processing time of the transaction and the number of communication messages to compare the performance of AT3M and V-Locking algorithms. The processing time is measured from the time the user submits the transaction to the system to the time the user receives the result, which depicts the system performance perceived by the user. The number of communication messages implies the bandwidth requirement of the system. First, both systems process only global transactions under different probabilities of read-only transactions. As two read-only transactions do not conflict, lower probability of read-only transaction depicts higher probability of conflicts. As shown in figure 4, the processing time of AT3M algorithm is about half of the V-Locking algorithm because i) The AT3M does not have to wait for global locks at each SSN level, ii) GSTAgents enable parallel processing of all global subtransactions, and iii) GSTAgents and NodeManager try to locally restart the potentially conflict global subtransaction before requesting for global restart. Moreover, the processing time of AT3M across various probabilities of read-only transaction is more stable. Since

AT3M always organizes the global subtransactions into the commonly agreed on global order regardless of the degree of conflicts between the global transactions, the change in probability of read-only transactions only impacts the processing time at the local databases, not at the global level.

Figure 5 describes the average number of messages in the system per global transaction. With AT3M algorithm, each GSTAgent is responsible for each global subtransaction. It contacts the GTAgent only when prepare-to-commit, after successful commit, and to request for global restart. In contrast, V-Locking algorithm requires explicit acknowledgement from the local database back to the GTC after each operation. Therefore, number of communication messages under AT3M is significantly lower than V-Locking. Although the V-Locking algorithm provides options to send acknowledgement only after each write operation or after
commit/abort of the global subtransaction, such alternatives would be traded off with higher degree of fault deadlocks which would degrade other performance metrics.

Figure 6 illustrates the impact of changing the local to global transaction ratio (LT:GT), which shows the extent to which the global users would be affected from the load at the LDB. AT3M still give better processing time for global users. It is shown that the processing time of the local transaction is not affected by the LT:GT ratio. As the local transactions access only single LDB using wired connection, they always have shorter processing time than the global transactions.

VI. CONCLUSION AND FUTURE DIRECTIONS

A. Conclusion

This paper proposed AT3M as a new approach for transaction management in mobile multidatabase. It introduces the use of agent technology. Each global transaction is carried out by an agent, which dispatches child agents representing the corresponding global subtransactions. The protocol is a non-locking approach. The global serialization order is determined before the global subtransactions are submitted to the local databases. The use of the agents allows fully distributed transaction management and parallel processing of all global subtransactions, and also addresses user’s mobility. From the simulation, the proposed AT3M algorithm provides better average processing time and lower number of communication messages than the V-Locking algorithm which has been designed for the same environment.

B. Future Directions

The current work covers the main functions of transaction management for mobile multidatabase. There are still a lot of opportunities to enhance the proposed algorithm. First, the algorithm can be enhanced to operate under mobile ad hoc environment, which the local databases are mobile. Second, the system can use some caching scheme to give better processing time. Sharing cached information among users may be performed using the concept of co-operate caching [4]. Third, the protocol can be improved to handle user’s migration (i.e. disconnected user who is reconnected to a different access point).

REFERENCES


ACKNOWLEDGMENT

Notice: This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.