A TOGGLE TRANSACTION MANAGEMENT TECHNIQUE FOR MOBILE MULTIDATABASES

Ravi A. Dirckze
School of Computer Science,
University of Oklahoma,
Norman, Oklahoma 73019
radirckz@cs.ou.edu

Le Gruenwald
School of Computer Science,
University of Oklahoma,
Norman, Oklahoma 73019
gruenwal@cs.ou.edu

1. ABSTRACT

Frequent disconnection and migrating transactions are new issues that need to be addressed by the Mobile Multidatabase System (MMDBS). Disconnection cannot be treated as catastrophic failures that result in aborted transactions. Disconnection and migration prolong the execution time of transactions which, in turn, affects concurrency control. Furthermore, limiting the MMDBS to purely ACID transactions may not be desirable. In this paper, we develop a Toggle Transaction Management Technique for the Mobile Multidatabase environment that addresses these issues. Two new states - Disconnected and Suspended - are introduced to address disconnection and migration. A toggle operation is introduced to handle the extended execution time of transactions. A Partial Global Serialization Graph commit algorithm that supports a wide range of correctness criterion with respect to the Atomicity and Isolation properties is introduced and its correctness is proved.

2. INTRODUCTION

The mobile computing model consists of two distinct sets of entities: a fixed network system and a continuously changing set of mobile clients (Figure 1). The fixed networking system consists of a collection of static computers connected by a wired network. Some units on the static network have the capability of communicating with mobile units through a wireless medium and are called mobile support stations (MSS). The area covered by an MSS is called a cell [11]. During the course of execution, the mobile user is likely to migrate across cells. The typical mobile computer has limited resources compared to its desktop counterpart [12]. Due to the unreliability of the wireless communication medium as well as limited resources available, the mobile user will be characterized by frequent disconnection. However, disconnection is predictable [9].

CIKM 98 Bethesda MD USA

The Mobile Multidatabase System (MMDBS) consists of a collection of autonomous databases connected to a fixed network and a Mobile Multidatabase Management System (MMDBMS). The MMDBMS is a set of software modules that resides on the fixed network system. The respective Database Management System (DBMS) of each independent database retains complete control over its database. These databases use different data models, transaction management mechanisms, etc. [1]. Each local database provides a service interface that specifies the operations accepted and the services provided to the MMDBMS. The MMDBMS cooperates with the service interfaces to provide extended services to its users. Existing (local) users of the autonomous databases will continue to access these databases through their respective DBMSs. Local transactions executed by local users are transparent to the MMDBMS. Global users - users connected to the MMDBMS - are capable of accessing multiple databases by submitting global transactions to the MMDBMS. Global users could be either static users or mobile users.

A global transaction consists of a set of operations, each of which is a legal operation accepted by some service interface. Any subset of operations of a global transaction that access the same site may be executed as a single transaction with respect to that site and will form a logical unit called a site-transaction. Site-transactions are executed under the authority of the respective DBMS. As mobile users migrate from one MSS to another, operations of a global transaction may be submitted from different MSSs. Such transactions are referred to as migrating transactions.

The Global Transaction Manager (GTM) is a software component of the MMDBMS that manages the execution of global transactions. A transaction is defined as an independent unit of consistent and reliable computing [11]. Ensuring that global transactions are consistent and reliable units of computing is the primary responsibility of the GTM. Generally, this can be achieved by enforcing the ACID (Atomicity, Consistency, Isolation, and Durability) properties [6].

As a consequence of autonomy, we can assume that no data integrity constraints are defined on data items residing at different sites [3]. As each local DBMS will ensure that the site-transactions executed by it do not violate any local integrity constraints, global transactions will, by default, satisfy the Consistency property. Similarly, the GTM can rely on the Durability property of the local DBMS to ensure Durability of committed global transactions. Thus, the GTM need only enforce the Atomicity and Isolation properties.

In [5], the authors make a compelling argument for providing unrestricted access to data in the MMDB environment.

"Returning dirty data tagged with appropriate warnings is much more useful than returning an ABORT message ...". This will make it necessary for the GTM to support a spectrum of correctness criterion with respect to the Atomicity and Isolation properties.

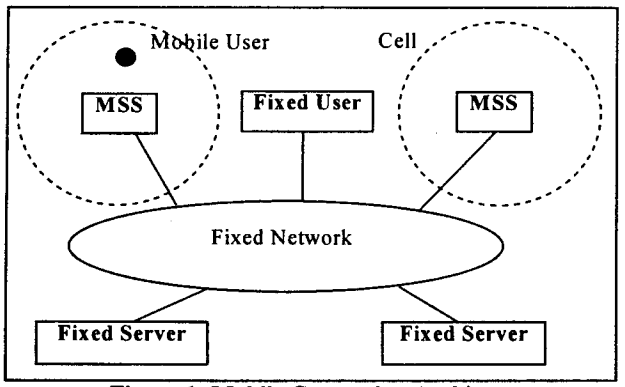

Figure 1: Mobile Computing Architecture

In addition to the Atomicity and Isolation properties, the GTM of the MMDBMS needs to address disconnection and migrating transactions. Unlike disconnection in the static environment, disconnection in the mobile environment cannot be treated as failures that result in aborted transactions. Disconnection and migration need to be viewed as events that occur during the normal course of execution of a global transaction. However, in some cases, disconnection will be caused by catastrophic failures, or a catastrophic failure may occur after a disconnection. Halted transactions are not resumed after a catastrophic failure and need to be aborted. As the MMDBMS can only predicted the nature of the disconnection, erroneous decisions are bound to be made resulting in unnecessary abortions. The GTM needs to minimize unnecessary abortions caused by erroneous decisions.

Some global transactions are expected to be interactive by nature, i.e., pause for input by the user [2]. The interactive nature of global transactions, as well as disconnection and migration prolong the execution time of global transactions. Such transactions are referred to as Long-Lived Transactions (LLT) [10]. To tolerate LLTs without disruption to transaction processing, site-transactions should be allowed to commit independently of the global transaction so that (local) resources held by the site-transaction may be released in a timely fashion. In addition, as the execution time of a transaction increases, the probability of that transaction conflicting with other transactions increases and, therefore, LLTs are more likely to be aborted due to Isolation property violations [4]. The GTM needs to minimize this ill-effect upon LLTs.

Furthermore, the GTM of the MMDBMS should conform to Multidatabase design restrictions, that is, the autonomy of the local databases *cannot* be violated. Also, no modifications can be made to the local DBMSs in order to support the MMDBS.

Transaction management in the MMDB environment has received considerable attention lately [12] [2] [13] [5]. However, none of the proposed techniques provide a complete solution [4]. In fact, the affects of the prolonged execution of LLTs and disconnection caused by catastrophic failures are not addressed by any technique. Further, no technique enforces the Isolation property. A detailed review of transaction management techniques in the MMDB environment can be found in [4].

In this paper, a transaction management technique called the Toggle Transaction Management (TTM) technique that addresses the deficiencies that exist in the current literature will be introduced. This technique is presented in the following section. Concluding remarks and our plans for future research are presented in section 4.

3. THE TOGGLE TRANSACTION MANAGEMENT TECHNIQUE

In the Toggle Transaction Management (TTM) technique, the GTM consists of two layers: The Global Coordinator layer that manages the overall execution and migration of global transactions, and a Site Manager layer that supervises the execution of site-transactions. Site-transactions are categorized as either vital or non-vital [2]. This provides the flexibility to support a range of Atomicity and Isolation correctness criterion. Each global transaction will be associated with a data structure that contains the current execution status of that transaction, and will follow the user from MSS to MSS.

Two new states - Disconnected and Suspended - are introduced to support disconnected operations. Upon disconnection, associated transactions are placed in the Disconnected state and execution is allowed to continue. If at any time during a disconnection the MMDBS determines that a catastrophic failure has occurred and that the user is unlikely to reconnect, transactions are placed in the Suspended state and execution is suspended. Suspended transactions will not be aborted until they obstruct the execution of other transactions. Therefore, needless abortions caused by erroneous decisions made by the MMDBS will be minimized.

In order to minimize the ill-effects of the extended execution time of mobile transactions, a global transaction can state its intent to commit by executing a toggle operation. If the operation succeeds, the GTM guarantees that the transaction would not be aborted due to Atomicity or Isolation violations unless the transaction is Suspended.

Whenever a global transaction requires to commit or be toggled, the TTM technique executes the Partial Global Serialization Graph (PGSG) commit algorithm to verify the Atomicity and Isolation properties. The algorithm first verifies the Atomicity property. If Atomicity has been violated, the transaction is aborted; else, the Isolation property is verified. If Isolation has been violated, the algorithm attempts to resolve the violation. If Isolation property violations cannot be resolved, the transaction is aborted; else, the commit or toggle operation succeeds. This algorithm is based on an optimistic approach. Next, a detailed description of the TTM Technique is provided.

3.1 The Model

The GTM consists two layers: a Global Coordinator (GC) layer and a Site Manager (SM) layer (Figure 2). The Global Coordinator layer consists of a set of Global Transaction Coordinators (GTCs). Each GTC exists at an MSS and will manage the overall execution of global transactions of users currently connected to it. The Site Manager layer consists of a set of Site Transaction Managers (STMs). Each STM exists at a participating database site and will supervise the execution of site-transactions submitted to that site. The GTCs and STMs communicate with each other to coordinate the execution of global transactions.

Global transactions are based on the Multi-Level Transaction model [6] in which the global transaction consists of a set of compensatable sub-transactions. A compensatable transaction is a transaction whose effects can be undone after it has committed by executing a compensating transaction [8]. In the proposed model, all operations of a global transaction that access the same site constitute a site-transaction that is executed as a single transaction with respect to that site. All site-transactions are compensatable and, therefore, can be committed prior to the decision to commit the global transaction, releasing resources in a timely manner.

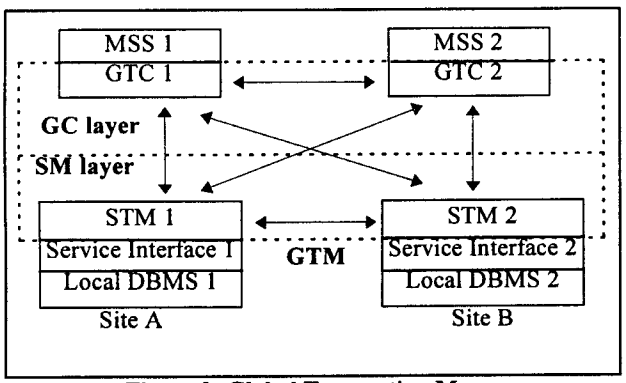

Figure 2: Global Transaction Manager

In addition, all site-transactions will be categorized as either vital or non-vital [2]. All vital site-transactions of a global transaction must succeed in order for the global transaction to succeed. The abort of non-vital site-transactions do not force the global transaction to be aborted. This categorization provides the flexibility to accommodate a wide range of Atomicity and Isolation criterion. The MMDBS can specify the levels of Atomicity and Isolation that need to be enforced in terms of either the number of vital and non-vital site-transactions, or restrictions placed upon non-vital site-transactions. Requiring non-vital site-transactions to be read-only is an example of such a restriction.

3.2 Disconnection and Migration Support

Global transactions are initiated at some GTC component of the GTM. The GTC will submit site-transactions to the STMs, handle disconnection and migration of the user, log responses that cannot be delivered to the disconnected user, enforce the Atomicity and Isolation properties, etc.

Each global transaction can be in one of five states: 1) Active - the user is connected and execution continues; 2) Disconnected - the user is disconnected, but the disconnection was predicted and re-connection is expected; 3) Suspended - the user is disconnected and the status is unknown; 4) Committed - the transaction committed successfully; and 5) Aborted - the transaction is aborted.

When a global transaction is initiated by a user, the respective GTC creates a Global (data) Structure to keep track of the information required to supervise its overall execution. The Global Structure is given in Table 1.

GTID	global transaction identifier
GT_Status	current state of global transaction
Commit_Intent	Boolean field - specifies commit intent
Site_List	respective site of each site-transaction
STID_List	respective STID of each site-transaction
Critical_List	vital/non-vital for each site-transaction
STID_Status_List	respective status of each site-transaction
Response_List	list of undelivered responses, if any

Table 1: Global Data Structure

The STM at each site supervises the execution of site-transactions submitted to that site. Each site-transaction could be in one of four states: 1) Active - the site-transaction is active; 2) Completed - the site-transaction has committed at the local database but the global transaction has not committed; 3) Aborted - the site-transaction is aborted; or 4) Committed - the site-transaction and the respective global transaction has committed. Each STM will maintain a Site Table containing information on all site-transactions submitted to it. For each site-transaction, the following information will be collected:

GTID	respective GTID
STID	assigned STID
MSS_ID	current MSS to which user is connected
STID_Status	current status of site-transaction
Comp_Transaction	compensating site-transaction

Table 2: Entry in Site Table

After submitting all operations of a site-transaction to the STM, the GTC will submit the respective compensating (site) transaction to the STM. Upon completion of the execution of a site-transaction, the STM will submit a commit operation to the local DBMS and update the STID_Status of that site-transaction to reflect the outcome of the local commit operation, i.e., marked Completed or Aborted.

Whenever a user disconnects, the GT_Status of all global transactions of that user are marked as Disconnected. Upon reconnection, the GT_Status of Disconnected transactions will be set to Active and execution is resumed. All responses received after disconnection are placed in the Response_List and delivered to the user upon re-connection. If the MMDBMS determines that a catastrophic failure has occurred, the GT_Status of global transactions related to that connection are marked as Suspended. Any site-transaction of the Suspended global transaction that obstructs the execution of another global transaction is aborted. If a user whose global transactions were Suspended re-connects, the execution of each Suspended transaction will be resumed only if none of its vital site-transactions have been aborted; else, the Suspended transaction is aborted. This will minimize the number of unnecessary abortions caused by erroneous decisions made by the MMDBMS.

When a user migrates to a new cell, the user will supply the current MSS with the identity of the previous MSS. The GTC at the current MSS will obtain the associated Global Structure from the previous GTC and take on the responsibility of overall execution.

In order to ensure that LLTs are not penalized due to their extended execution time, each global transaction will be associated with a Commit-Intent variable that is initially set to False. At any time during its execution, an LLT may execute a *toggle* operation. At this point, the respective GTC will determine whether the required levels of Atomicity and Isolation can be guaranteed for the current set of site-transactions of that global transaction by executing the PGSG commit algorithm to be

explained in section 3.3.1. If the required levels can be guaranteed, the toggle operation succeeds and the Commit_Intent field is set to True. Else, the global transaction has already violated the required levels and, therefore, is aborted. Once a global transaction is toggled, it may initiate only non-vital site-transactions. A successful toggle operation establishes the global transactions place in the serialization order and guarantees that the transaction would not be aborted due to Atomicity/Isolations violations unless it is Suspended. Although any site-transaction initiated after the toggle operation could cause a potential Atomicity/Isolation property violation, it cannot force the global transaction to be aborted as it is non-vital.

3.3 Enforcing the Atomicity and Isolation Properties

Whenever a global transaction is to be committed or toggled, the respective GTC will execute the PGSG commit algorithm to verify the Atomicity and Isolation properties. The algorithm first verifies the Atomicity property. If the required level of Atomicity can be guaranteed, the Isolation property is verified; else, it is aborted. If the required level of Isolation can be guaranteed, the transaction is toggled or committed; else, it is aborted.

The Atomicity property is based on *Semantic Atomicity* [8] which requires either all site-transactions to be committed or each site-transaction to be aborted or compensated for. This correctness criterion allows site-transactions to commit early, regardless of the future outcome of the global transaction.

In the PGSG commit algorithm, if a global transaction is to be aborted, all Completed site-transactions will be compensated for and all Active site-transactions will be aborted as required by Semantic Atomicity. However, as long as all vital site-transactions have successfully completed their execution, i.e., marked Completed, the global transaction is allowed to commit. A locally aborted non-vital site-transaction does not force the global transaction to be aborted. Although this is a violation of Semantic Atomicity, it is justified as follows: In section 1, it was stated that the GTM needs to support a wide range of correctness criterion with respect to the Atomicity property. If all site-transactions are categorized as vital, then Semantic Atomicity will be strictly adhered to. If all site-transactions are categorized as non-vital, non-Atomic global transactions will be supported. By allowing global transactions to consist of any combination of vital and non-vital site-transactions, the TTM technique supports the full range of Semantic Atomicity.

The Isolation property is based on serializability. In the MMDB environment, Serializability requires conflicting site-transactions of two global transactions to be serialized in the same order at all sites at which the two global transactions execute. In order to establish the serialization order of global transactions, each STM maintains an SSG that reflects the local serialization order of all site-transactions that execute at that site. As the serialization order of site-transactions within the local databases is transparent to the STM, this information is obtained implicitly by forcing conflicts among the site-transactions that execute at each site by using a data item called a ticket [7]. Each site-transaction is required to read the respective ticket, increment its value and write the new value back at the beginning of its execution. The Ticket value read by the site-transaction indicates its serialization order at that site [1] and will be used to construct the SSG.

The SSG at each site is a directed graph (or digraph) whose nodes represent global transactions and edges represent (forced) conflicts between their respective site-transactions executed at some site. For example, if $T_1 \rightarrow T_2$ exists in some SSG, then global transactions T_1 and T_2 access at least one common site where the ticket obtained by the site-transaction of T_1 is less than the ticket obtained by the site-transaction of T_2. The information contained within each node is given is Table 3. Each node in the SSG is categorized as either an Accessed node or a Propagated node. Whenever a site-transaction is submitted to a site, a node representing the global transaction is marked Accessed and added to the SSG using its ticket value to determine its serialization order. The Site_ID of the Accessed node is set to the ID assigned to that site. In addition, whenever a global transaction is committed or toggled by the PGSG commit algorithm, serialization information is propagated back to all participating SSGs. Nodes copied during this phase are marked as Propagated nodes.

GTID	respective global transaction ID
GT_Status	status of global transaction
Commit_Intent	commit intent of global transaction
Node_Category	Accessed or Propagated
Site_ID	Tthis Site_ID or ID of Propagated Site

Table 3: SSG Node

To verify serializability of a global transaction, the PGSG commit algorithm constructs a Partial Global Serialization (PGS) graph using the SSGs maintained at the sites at which the global transaction executed. The algorithm will then look for cycles containing Completed site-transactions of that global transaction in the PGS graph. If cycles are detected, the algorithm will attempt to break the cycles by aborting site-transactions of Suspended global transactions and/or non-vital site-transactions of that global transaction. If the cycles cannot be resolved, the global transaction is aborted; else, it is committed or toggled. As only Completed site-transactions of that transaction are taken into account when looking for cycles, the PGSG algorithm enforces the full range of Isolation as well. A detailed description of the algorithm follows.

3.3.1 The PGSG Algorithm

First, it is necessary to define certain terms used in the algorithm.

Definition 1: We say that T_j is *reachable* from T_i in graph G if there is a path from T_i to T_j in G.

Definition 2: Reachable(T_j,) is a (directed) sub-graph of an SSG that contains node T_j and all nodes T_i such that T_j is reachable from T_i in the SSG.

Definition 3: a *Candidate* node is any Propagated node whose GT_Status is not Committed.

Definition 4: Let G_i and G_j be two graphs with node sets N_i and N_j and edge sets E_i and E_j respectively. The operation $G = Merge(G_i, G_j)$ results in a new graph G(N, E) such that $N = N_i \cup N_j$ and $E = E_i \cup E_j$ where $\cup$ is the union operator.

Definition 5: The graph *Predecessor(T_j, S_m)* is the sub-graph Reachable(T_j) of the SSG at site S_m Merged with the graphs Predecessor(T_k, S_n) where T_k is a Candidate node in Reachable(T_j) and S_n is the respective propagated site of T_k. Formally,

Predecessor(T_j, S_m) = { G = Reachable(T_j) | Merge (G, Predecessor(T_k, S_n)) $\forall$ Candidate nodes T_k in Reachable(T_j) where S_n is the Site_ID of T_k }

Definition 6: The list *PList(T_j, S_m)* is a list of sites maintained at site S_m that contains the list of sites from which the Predecessor graphs were obtained in order to construct Predecessor(T_j, S_m).

Whenever a global transaction, say T_j, is to be committed or toggled, the PGSG algorithm will first verify Semantic Atomicity by inspecting the Global Structure of T_j. If the STID_Status_List indicates that all vital site-transactions have completed successfully, then the required level of Semantic Atomicity can be guaranteed; else, T_j is aborted.

If Semantic Atomicity can be guaranteed, the PGSG algorithm verifies the Isolation property by determining serializability. In order to construct the PGS graph, this algorithm initiates the Request Predecessor(T_j, S_m) algorithm at all sites S_m at which T_j executed successfully. The Predecessor(T_j, S_m) graphs returned by the sites are Merged to construct PGS graph. Next, the algorithm verifies whether T_j violates serializability with respect to Committed and Toggled transactions by checking for cycles amongst the relevant nodes in the PGS graph, i.e., Completed site-transactions of T_j and all nodes representing Committed and Toggled global transactions. The GT_Status and Commit_Intent fields of the nodes are used to determine whether a transaction is Committed or Toggled. If a cycle is detected, the algorithm attempts to break the cycle by first aborting site-transactions of Suspended global transactions and then aborting non-vital site-transactions of T_j. If the cycle cannot be broken, the global transaction is aborted. If the cycle can be broken, then Isolation can be guaranteed and the operation succeeds. Each STM, except the STMs at which non-vital site transactions need to be aborted in order to break cycles, is informed that Tj is to be committed or toggled. In addition, these STMs receives a copy of the PGS graph so that serialization information contained in the PGS graph will be propagated.

After a toggled transaction completes its execution, it needs to execute the PGSG algorithm a second time in order to determine if any non-vital site-transactions initiated after the toggle operation violate the serialization order established by the toggle operation. Any site-transaction that violates this order needs to be aborted. The PGSG algorithm is given below. The Operation field specifies whether the requested operation is a Commit or a Toggle operation.

PGSG Algorithm (Operation, T_j)
/* This algorithm verifies Atomicity and Isolation */
/* first, verify required level of Semantic Atomicity */
If any critical site-transaction has been aborted /* violation */
  Send Abort(T_j) to all sites in Site_List
Else
/* verify required level of Isolation */
Request Predecessor(T_j, S_m) from all site S_m in Site_List where
-- T_j is marked as Completed
Generate PGSG by Merging all Predecessor(T_j, S_m)
Check for cycles w.r.t. T_j, Committed nodes and Togged nodes
If cycles are detected
  If cycles can be broken by aborting Suspended and/or
  -- non-vital site-transactions of T_j
    Mark GT_Status of respective nodes as Aborted in PGSG
    If node represents non-vital site-transaction of T_j
      Send Abort (T_j) to respective site and remove associate
      -- site from Site_List
    End If
  Else /* Isolation violated */
    Send Abort (T_j) to Site_List
    Exit Algorithm
  End If
End If
/* required level of Atomicity and Isolation can be guaranteed */
Mark GT_Status of T_j = Committed (or toggle Commit_Intent)
-- in Global Structure
Send Committed (or Toggled) to Site_List
/* Propagate Serialization information to SSGs */
Send PGSG to all sites in Site_List
End {PGSG Algorithm}

In order to construct the PGS graph for global transaction T_j, the PGSG algorithm will initiate the Request Predecessor algorithm to obtain Predecessor(T_j, S_p) from all sites S_p at which T_j executed successfully. These sites are referred to as *Primary* sites. To construct Predecessor(T_j, S_p), each Primary site obtains Predecessor(T_k, S_s) for all Candidate nodes T_k in Reachable(T_j) where S_s is the Site_ID of T_k. In turn, each site S_s may obtain Predecessor graphs from other sites. All non-Primary sites that participate in the algorithm are referred to as *Secondary* sites. At each site, PList contains the list of sites from which it obtained Predecessor graphs. Each Primary site submits Predecessor(T_j, S_p) to the respective GTC and awaits the outcome of the operation. If T_j is to be committed, then the respective STID_Status of T_j in the Site Table and the GT_Status of the respective node in the SSG is marked as Committed. If T_j is to be toggled, then the Commit_Intent field of the respective node in the SSG is set to True. In addition, each Primary site, except the Primary sites at which non-vital site-transactions were aborted in order to eliminate cycles, will receive a copy of the PGS graph so that serialization information will be propagated to its SSG. The Primary site will then submit the PGS graph to all sites in its PList so that the serialization information is propagated to all participating sites. As each participating site, where Predecessor(T_y, S_x) represents the graph submitted by that site, receives the PGS graph, all nodes that appear before T_y in the serialization order in the PGS graph are propagated to its SSG by Merging Reachable(T_y) of the PGS graph with the SSG at that site. Any Suspended nodes that were aborted in order to commit T_j will be aborted at the respective sites. This information is contained in the propagated PGS graph as well.

If the response received by the STM at the Primary site is to abort T_j, then the respective STID_Status of T_j in the Site Table is marked as aborted and the respective node is deleted from the SSG. If T_j is marked Completed, then the associated compensating transaction is executed; else it is Active and T_j is aborted. All Secondary sites in PList are informed that T_j was aborted. No serialization information needs to be propagated as, in essence, the execution of T_j is erased from that site. Note that a Primary site at which T_j executed a non-vital site-transaction may receive an abort response even though T_j is to be committed, as it may have been necessary to abort that site-transaction in order to eliminate cycles in the PGS graph. The Request Predecessor algorithm is given below:

Request Predecessor(T_j, S_m)
Construct Predecessor(T_j, S_m), PList(T_j, S_m)
Submit Predecessor(T_j, S_m)
If Reply is Abort(T_j) /* site-transaction is to be aborted */
  If T_i is Accessed node in SSG /* this is a Primary site */
    Abort or compensate T_j
  End If
  /* inform all Secondary sites */
  Send Abort (T_j) to all sites in PList(T_j, S_m)
Else /* global transaction is to be committed or toggled */
  If T_i is Accessed node in SSG /* this is a Primary site */
    Mark GT_Status as Committed or Commit_Intent as True

If T_j is committed
Mark STID_Status as committed
End If
End If
/* Propagate serialization information */
SSG = Merge(SSG, Reachable(T_j) of received PGSG)
Abort any Suspended nodes marked as Aborted in PGSG
Send Committed/Toggled and PGSG to all sites in PList(T_j, S_m)
End If
End { Request Predecessor}

2.3.2 Proof of Correctness

In this section, it will be proven that all cycles will be detected by the PGSG algorithm. First, it is necessary to present the following lemma:

Lemma 1: Let $T_i \rightarrow T_j$ be in the SSG at some site S_j. Then, T_i began its execution at S_j prior to the completion of T_j's execution at S_j and therefore, prior to the (global) commit of S_j.

Proof: In order for $T_i \rightarrow T_j$ to exist, T_i must have obtained a ticket that is less than the ticket obtained by T_j. Therefore, T_i began its execution at S_j prior to T_j completing its execution at S_j.

Theorem 1: Let $T = \{T_1, T_2, ..., T_n\}$ be a set of transactions that cause a cycle. Assume that T_2 through T_n commit successfully and that T_1 is the last transaction in T to attempt to commit. Then T_1 will be an Accessed node as well as a Candidate node in some Predecessor(T_1, S_x) used to construct the PGSG. Thus, the cycle will be detected.

Proof: For simplicity, let us assume that each transaction executes at exactly two sites such that the cycle
$C \equiv T_1 \rightarrow T_2 \rightarrow ... \rightarrow T_n \rightarrow T_1$ is produced.
By Theorem 1, for all T_q that have completed their execution, there exists $T_p \rightarrow T_q$ for some T_p in T in the SSG at some site S_q at which T_q executed. When T_q executes the PGSG commit algorithm, T_p is in Predecessor(T_q, S_q) used to construct the PGSG. Now, either T_p is committed, or not committed.
If T_p is not committed then T_p will be added as a Candidate node to all the SSGs at which T_q executed.
If T_p is committed, then, by Theorem 1, there exists a T_m in S such that $T_m \rightarrow T_p$ at some site S_m at which T_p executed. Once again, either T_m was committed or not committed at the time of T_p's commit. If T_m was not committed, then T_m was propagated to S_m at the time of T_p's commit and, as a result, will be in Predecessor(T_q, S_q) at the time of T_q's commit and will be added as a Candidate node to all SSGs at which T_q executed. If T_m was committed, we may repeat this argument. As the conflicts are cyclic, Predecessor(T_q, S_q) used to construct the PGSG when T_q attempts to commit will always contain a non-committed node from T which will then be added as a Candidate node to all SSGs at which T_q executed.
Now let T_1 attempt to commit at site S_1 and Sn where $T_1 \rightarrow T_2$ and $T_n \rightarrow T_1$ exist respectively. Then, as T_n is committed, the SSG at S_n will contain a Candidate node - say T_x with respective site S_x - in its Predecessor(T_1, S_n). If T_x committed after its propagation to site S_n, then the SSG at S_x would, in turn, contain a Candidate node. Finally, As the only node in the cycle that is currently active is T_1, the Predecessor(T_1, S_n) constructed at site S_n will contain T_1 as an Accessed node as well as a Candidate node. Therefore, Predecessor(T_1, S_n) will contain the entire cycle. Thus, the PGSG will contain the cycle. As all nodes except T_1 are committed, the cycle will be detected.

3.3.3 A Sample execution of the PGSG algorithm

The following example is used to illustrate the PGSG algorithm. In this example, the MMDBS consists of 4 sites labeled S_1 through S_4. There are 4 active global transactions labeled T_1 through T_4 in the system. For simplicity, we assume that each transaction accesses two sites and that all transactions have completed their execution but have not yet committed. The algorithm is illustrated in Table 4. The initial SSGs are given in row one. Initially, all nodes are Accessed nodes. Each of the next 4 rows reflects the SSGs after the completion of the PGSG algorithm of the specified transaction. In each row, only the sites that participate in the commit algorithm will have corresponding entries. For example, as S_1, S_2 and S_3 participate in the commit phase of transaction T_2, these columns will contain the new SSGs. The column representing S_4 will not contain an entry as it does not participate in the commit phase of T_2. The associated site labels of all Propagated nodes will be specified in brackets below the respective node (transaction) in each SSG. The last column in the table reflects the PGSG that is constructed at each stage. A [C] below the respective node in the PGSG states that the node is to be committed and an [A] states that the node is to be aborted.

In this example. transactions T_1, T_2, and T_3 commit successfully. During the commit phase of T_1, node T_4 will be propagated from the PGSG to the SSG at S_2. During the commit phase of T_3, node T_2 will be propagated from the PGSG to the SSG at S_4. During the commit phase of T_2, nodes T_4 and T_1 will be propagated from the PGSG to the SSG at S_3. Finally, when transaction T_4 executes the PGSG commit algorithm, it will be forced to abort as the PGSG contains a cycle and all other transactions involved in the cycle have already committed.

4. Concluding Remarks

In this paper, a transaction management technique for the Mobile Multidatabase environment called the Toggle Transaction Management (TTM) technique is presented. In TTM, site-transactions are allowed to commit independently so that resources may be released in a timely manner. Two new states - Disconnected and Suspended - are introduced to address disconnection and migration. Unnecessary abortions caused by erroneous decisions made by the MMDBMS w.r.t. the status of a disconnection are minimized. A toggle operation is used to minimize the ill-effects of the prolonged execution of Long-Lived transactions. A PGSG commit algorithm that enforces a wide range of correctness criterion with respect to the Atomicity and Isolation properties is proposed and its correctness is proved.

In the TTM technique, concurrency is limited as all site-transactions that execute at each site are forced to conflict with each other. In our future research, the artificial conflicts generated by the algorithm will be eliminated by exploiting semantic information of site-transactions. Each service interface will need to provide conflict information on all operations accepted by that site. This information will be used to generate conflicts amongst site-transactions that actually conflict with each other. The current transaction model will also be extended to include compensatable and retriable site-transactions in the same global transaction in order to expand the application domain. Finally, we intend to carry out a performance evaluation and comparison study.

	S_1	S_2	S_3	S_4	PGSG
Initial SSGs	$T_4 \rightarrow T_1$	$T_1 \rightarrow T_2$	$T_2 \rightarrow T_3$	$T_3 \rightarrow T_4$	
Commit of T_1	$T_4 \rightarrow T_1$	$T_4 \rightarrow T_1 \rightarrow T_2$ [S_1]			$T_4 \rightarrow T_1 \rightarrow T_2$ [C]
Commit of T_3			$T_2 \rightarrow T_3$	$T_2 \rightarrow T_3 \rightarrow T_4$ [S_3]	$T_2 \rightarrow T_3 \rightarrow T_4$ [C]
Commit of T_2	$T_4 \rightarrow T_1$	$T_4 \rightarrow T_1 \rightarrow T_2$ [S_1]	$T_4 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3$ [S_1] [S_2]		$T_4 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3$ [C]
Commit of T_4	T_1	$T_1 \rightarrow T_2$	$T_1 \rightarrow T_2 \rightarrow T_3$ [S_2]	$T_2 \rightarrow T_3$ [S_3]	$T_4 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$ [A]

Table 4: Sample execution of PGSG algorithm

5. REFERENCES

[1] Breitbart Y., Garcia-Molina H., Silberschatz A. "Overview of Multidatabase Transaction Management" , TR-92-21, University of Texas at Austin.

[2] Chrysanthis P.K. "Transaction Processing in Mobile Computing Environments", IEEE Workshop on Advances in Parallel and Distributed Systems, October 1993.

[3] Du W., Elmagarmid A. K. "Quasi Serializability: A Correctness Criterion for Global Concurrency Control in InterBase". Proceedings of the 15th International Conference on VLDB, Amsterdam, The Netherlands, August 1989

[4] Dirckze R., Gruenwald L. "Disconnection and Migration in Mobile Multidatabases", The World Conf. on Design and Process Technology, Germany, July 1998

[5] Dunham M., Helal A., Balakrishnan S. "A Mobile Transaction Model that Captures Both the Data and Movement Behavior ". Mobile Networks and Applications, Vol. 2, No. 2, October 1997.

[6] Gray J., Reuter A. "Transaction Processing: Concepts and Techniques". Morgan Kaufmann Publishers, Inc. 1993.

[7] Georgakapolous D., Rusinkeiwicz M., Sheth A. "On Serializability of Multidatabase Transactions through Forced Local Conflicts", Proceedings of the 7th International Conference on Data Engineering, Kobe, Japan, 1991

[8] Mehrotra S., Rastogi R., Silberschatz S, Korth H. F. "A Transaction Model for Multidatabase Systems", Proc. 12th International Conference on Distributed Computing Systems, Japan 1992

[9] Pitoura E., Bhargava B. "Dealing with Mobility: Issues and Research Challenges". Technical Report CSD-TR-93-070, Purdue University 1993.

[10] Pitoura E., Bhargava B. "Revising Transaction Concepts for Mobile Computing", Proceedings of the IEEE Workshop on Mobile Systems and Applications, Santa Cruz, CA, December 1994.

[11] Pitoura E., Bhargava B. "Maintaining Consistency of Data in Mobile Distributed Environments". 15th International Conference on Distributed Computing Systems, Canada, June 1995.

[12] Pitoura E., Bhargava B. "A Framework for Providing Consistent and Recoverable Agent-Based Access to Heterogeneous Mobile Databases". SIGMOD Record, September 1995

[13] Yeo L. H., Zaslavsky A. "Submission of Transactions from Mobile Workstations in a Cooperative MDB Processing Environment". 14th International Conference on Distributed Computing Systems, Poland, June 1994