POWER AWARE MANAGEMENT OF MOBILE REAL-TIME DATABASE TRANSACTIONS IN AD-HOC NETWORKS

Le Gruenwald
Shankar M. Banik

University of Oklahoma
School of Computer Science
Norman, OK 73019
(ggruenwald@ou.edu; smbanik@ou.edu)

Abstract

In an ad-hoc mobile network architecture, all mobile hosts (MHs) are connected with each other through a wireless network that has a frequently changing topology. This type of architecture is used in many applications such as battlefields and disaster recovery where it is difficult or not feasible to depend on a static wired communication infrastructure. These applications are usually time-critical where many of their transactions must not only be executed correctly but also within their deadlines. In addition, the MHs in this environment are not connected to unlimited power supplies and may store data that can be shared by other MHs. Existing mobile database transaction management techniques do not consider the ad-hoc network characteristics, real-time constraints, and energy limitation. This paper reviews these existing techniques, identifies issues that need to be addressed in this new environment, and propose approaches for solutions.

1. INTRODUCTION

Advances in connectivity and wireless communications have revolutionized the computer industry. Connectivity has enabled software available on independent systems at different locations to cooperate in order to provide a wider spectrum of services to users. Wireless communication technology makes it possible to extend these services to nomadic users. Mobile MultiDatabase Management Systems (MMDBMSs) are those that provide services to allow nomadic users to access databases conveniently and efficiently. Transaction Manager (TM) is a vital component of any MMDBMS. TM is responsible for providing reliable and consistent units of computing to its users. The wireless communication mediums introduce new issues that need to be addressed by the TM, i.e., frequent disconnection and migration. Unlike in wired systems, disconnection in wireless systems cannot be treated as catastrophic failures that result in aborted transactions. When a disconnection occurs, TM needs to determine the status of the user,

and if reconnection is expected, the transaction must not be aborted. However, even if reconnection is not expected, aborting a transaction should be postponed as long as possible as the status of the user can only be predicted. Also, a disconnected user may resume execution from a different location. Disconnection and migration prolong the execution time of transactions resulting in a higher probability of conflicts with other transactions. Thus, it is necessary to ensure that transactions of mobile users are not penalized due to their extended execution time. This means that Long-Live Transactions (LLT) must be supported. Limiting MMDBMSs to purely ACID (Atomicity, Consistency, Isolation, Durability) transactions has also been argued. Because it is expected that enforcing ACID to MMDBMS may lead to too many aborts which will result in a system that is perfectly consistent but gets only a small fraction of useful work done [Dunham, 1997]. This dictates that MMDBMSs will need to support a range of correctness criteria. In addition, any solution should conform to multidatabase design restrictions, i.e., the autonomy of the local databases should not be violated.

There are two typical mobile computing architectures. In the General Mobile Computing Architecture, there is a fixed Mobile Support Station (MSS) that supports all mobile hosts (MHs) roaming within its cell. When an MH moves out of a cell and enters a new cell, it can no longer communicate with the previous cell's MSS, and is under the control of the new cell's MSS. All MSSs communicate with each other via a fixed network. In the second architecture called an Ad-hoc Mobile Network Architecture, all MHs are roaming and the network that interconnects these MHs is a wireless network with a frequently changing topology, and there are no fixed infrastructure and fixed MSSs. This second kind of architecture is widely used in battlefields and in disaster recovery situations ([Hong, 1999][Liu, 1999]).

Much research in the area of mobile database transaction management was based on the first architecture ([Dunham,1997], [Madria,1998], [Dirckze,2000]), while none on the second one. Supporting database transaction services in an ad-hoc mobile network raises new issues. If an MH stores a database, then other MHs will try to submit transactions and get data from it. In this environment both the user and the data source will be moving. So finding a route from one MH to another MH is necessary

before submitting a transaction. Moreover applications like battlefields are time-critical which require their transactions to be executed not only correctly but also within their deadlines. Thus the Transaction Manager at the MH where the database is stored has to consider the mobility of the submitting MHs as well as the deadlines of the transactions. Another important issue in ad-hoc networks is power or energy restriction on MHs because MHs are not connected to direct power supplies and many of them are small and low-power devices. So energy-efficient solutions are needed for this environment.

The objectives of this paper are to provide the state of the art of mobile database transaction management, identify deficiencies of these techniques in terms of support for ad-hoc networks, real-time constraints, and energy efficiency, and propose a direction to find solutions to resolve these issues. The paper is organized as follows. Section 2 describes the Ad-hoc Mobile Network Architecture. Section 3 reviews some of the most recent mobile transaction management techniques. Section 4 presents our solution direction. Finally Section 5 concludes the paper.

2. ARCHITECTURE

In ad-hoc networks, MHs communicate with each other without the help of a static wired infrastructure. These types of networks are usually used in battlefields ([Hong, 1999][Liu,1999]). So we have defined our architecture considering the battlefield environment as illustrated in Figure 1. Depending on communication capacity, computing power, disk storage, size of memory and energy limitation, MHs in the proposed network architecture can be classified into two groups: 1) computers with reduced memory, storage, power and computing capabilities (e.g. soldiers equipped with portable computing and transceiver devices), which we will call *Small Mobile Hosts (SMHs),* and 2) classical workstations equipped with more storage, power, communication and computing facilities than the *SMHs*, which we will call *Large Mobile Host (LMHs)*. These *LMHs* can be classified into two subgroups – humvees and tanks. Humvees have high capacity communication links and relatively stable. Tanks have less storage, computing and energy capacities and move more often than humvees. Both humvees and tanks are more

static than *SMHs* [Liu 1999]. Soldiers (i.e. *SMHs*) can communicate with tanks and humvees via wireless LAN technology. One soldier can talk to several humvees or tanks at the same time.

Every MH has a radius of influence. In Figure 1, a circle with borders in dotted line represents the radius of influence of an MH. An MH can directly communicate with other MHs which are within its radius of influence. That means two MHs can communicate with each other if each MH is within the radius of influence of the other MH. The communication link between two MHs shown with dark dotted lines in Figure 1 means both the MHs can communicate with each other. In our proposed environment, if two MHs are outside each other's radius of influence, they will be able to indirectly communicate with each other in multiple hops using other intermediate MH*s* between them [Bandyopadhay, 1999]. For example, in Figure 1, *SMH 11* will not be able to communicate directly with *LMH 3* because *LMH 3* is residing outside the radius of influence of *SMH 11*, but it can indirectly communicate in multiple hops using *SMH 10* and *SMH 9* between them.

MHs in battlefields are not connected to unlimited power supplies and thus have energy limitation. To reduce energy consumption, the MHs can operate in three modes - Active mode, Doze mode and Sleep mode.

1. Active Mode: The MH performs its usual activities. Its CPU is working and its communication device can transmit and receive signals.
2. Doze Mode: The CPU of the MH will be working on a lower rate. It can examine messages from other MHs. The communication device can receive signals. So the MH can be awaken by a message from other MHs [Barbara, 1994].
3. Sleep Mode: Both the CPU and the communication device of the MH are suspended.

Due to energy and storage limitations, we will assume that only *LMHs* will store the whole Data Base Management System (DBMS) and *SMHs* will store only some modules of the DBMS (e.g. Query Processor) that allow them to query their own data, submit transactions to *LMHs* and receive the results.

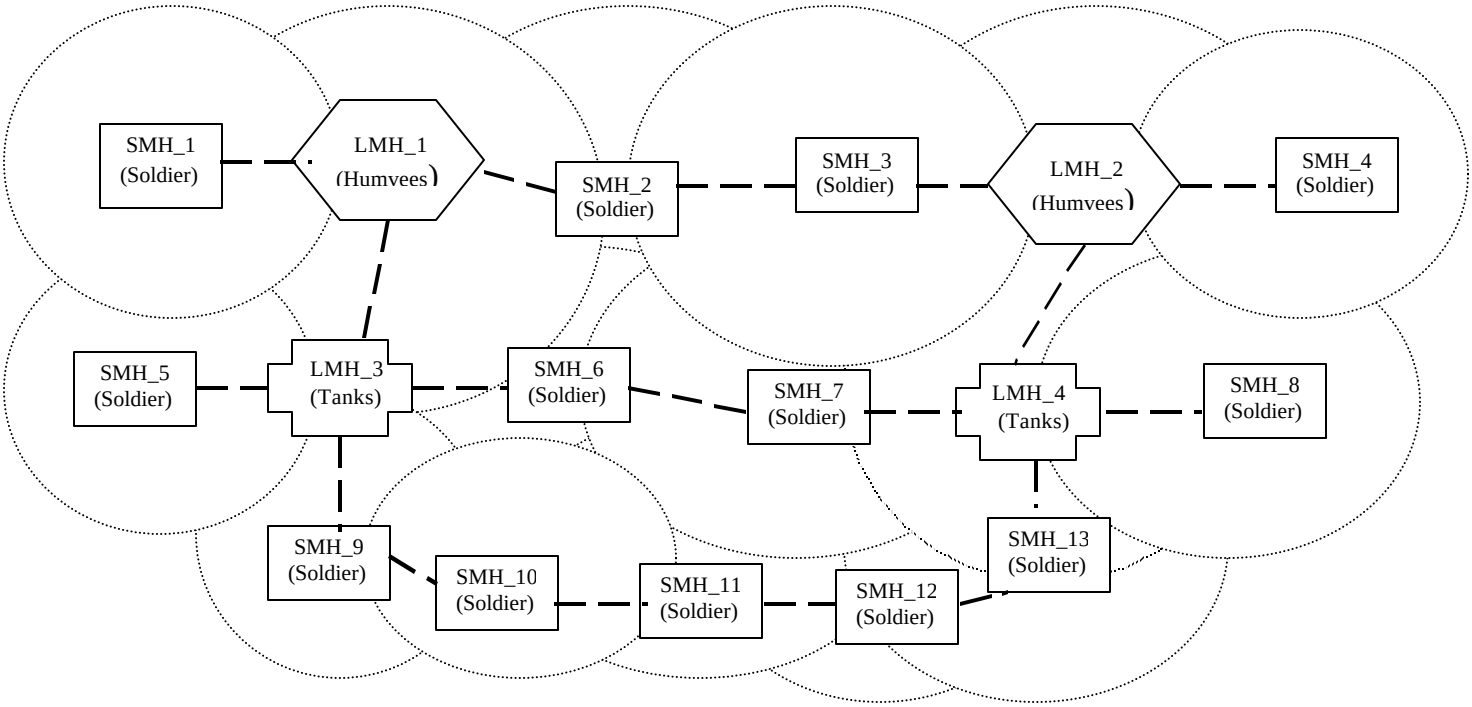

Figure 1: Architecture

3. REVIEW OF EXISTING MOBILE TRANSACTION MANAGEMENT TECHNIQUES

The Kangaroo model proposed in [Dunham, 1997] is based on the Open Nested model and captures the movement behavior of MHs. A global transaction (referred to as a Kangaroo transaction) consists of a set of Joey transactions, each consisting of all operations executed within the boundaries of one MSS. Each Joey transaction consists of one or more sub-transactions, and therefore, does not allow arbitrary migration that may occur in the middle of a sub-transaction. Joey transactions may be committed independently. Kangaroo transactions execute in two different modes: Compensating mode and Split mode. Under the Compensating mode, the failure of any Joey transaction causes all committed Joeys to be compensated and any other active Joeys to be aborted. Under the Split mode, all committed Joeys will not be compensated and the decision to commit or abort any active Joeys is left up to the component DBMSs. These modes provide a full spectrum of Atomicity. However, under the Split mode, component DBMSs may be left in an inconsistent state. Neither mode enforces the Isolation property. This model does not infringe upon the autonomy of any local DBMS.

The Pre-Commit model proposed in [Madria, 1998] introduces a pre-read, pre-write, and pre-commit operation to address the issues of mobile computing. Transactions of mobile users read or pre-read data values, manipulate the data that have been read and then pre-write modified values at the MH. Once all pre-write values have been declared, the transactions pre-commit at which point, all pre-write values are transmitted to the MSS. The MSS will then complete the transactions. After the pre-commit stage, the remaining part of execution of a transaction is shifted to the stationary host. Thus it reduces the computing cost at the MH. A pre-write does not update the state of the physical data object but only declares its modified value. Once a transaction pre-commits, its pre-write values are written to a pre-write buffer maintained in the MSS and are made visible to other concurrent transactions executing at that MH and the respective MSS. A transactions read will return a pre-read value if the latest value available has not been written to the database yet; otherwise, the value residing in the database (read value) will be returned. All pre-committed transactions are guaranteed to commit by the MSS. This transaction model does not address disconnection that represents catastrophic failures. It addresses the concurrency limitation caused by the extended duration of mobile transactions by maintaining a pre-write buffer and making the pre-write values visible upon pre-commit. However, the pre-write values are visible only to those transactions that are executing in that MH or MSS. This technique violates local autonomy as transactions are pre-committed by the MSS which guarantees that the pre-committed transaction will not be aborted; this cannot be achieved without the unilateral cooperation of the local databases. This technique enforces strict Atomicity and strict Isolation.

In the PSTM technique proposed by [Dirckze, 1998,2000], the Global Transaction Manager (GTM) consists of two layers: the Global Transaction Coordinator (GTC) which resides at each MSS, and the Site Transaction Manager (STM) which resides at each local database site. All local databases are connected to a fixed network. When an MH submits a global transaction to the GTC, the GTC creates a global data structure to supervise the overall execution of the global transaction. Then it submits all sub-transactions of the global transaction and their compensating transactions to the corresponding sites. The STM at each site supervises the execution of site transactions submitted at that site. After the completion

of each site transaction, the STM informs the GTC of the status of the site transaction. Global transactions of MHs are classified as LLTs (Long lived Transactions) and global transactions of static users are classified as non-LLTs. For a non-LLT, the GTC will execute the Partial Global Serialization Graph (PGSG) Algorithm to verify the Atomicity and Isolation properties of the non-LLT after its execution. If the properties are not violated the global transaction is committed; otherwise it is aborted. For LLT, the GTC will execute the PGSG at the end of its vital phase. If the properties have not been violated, it is toggled and its execution continues; otherwise it is aborted. If an MH migrates to a new cell, the MH will inform the new MSS of the identity of the previous MSS. The GTC of the new MSS will obtain the whole global data structure from the previous MSS and will be responsible for the execution of the global transaction of this user. If a user disconnects, its status is marked as disconnected but its transactions are not halted. But if the GTM determines that a catastrophic failure has occurred, then they are halted and marked as suspended. Suspended transactions are not aborted until they obstruct other global transactions. This technique takes care of the Disconnection and Migration issues of Mobile Database systems. It does not violate the autonomy of the local database sites. It enforces the Atomicity and Isolation properties of transactions.

All the above reviewed techniques are based on the General Mobile Computing Architecture. So they address the mobility of users only. But in an ad-hoc mobile network environment the servers that store the data sources are also MHs; there are no fixed MSSs. And the precise positions of the users and the data sources cannot be located in advance. So before submitting a transaction, an MH has to find the MH which has the data. After processing the transactions the MH has to find the requester and submit the results. So routing should be a part of transaction management in our environment. From the energy point of view, since all MHs will be running on limited power, they can go into the doze mode or sleep mode at any time to reserve energy. The reviewed techniques do not address the energy-related issues (e.g. if an MH goes into the doze mode then how the MSS will take care of the transaction submitted by that MH). In our environment the *LMHs* can also go into the doze mode and sleep mode and are different

from the MSSs which are servers with unlimited power. The techniques also do not deal with real-time transactions. Associating deadlines with transactions will have an impact on each of the proposed techniques. We will need a Real Time Transaction Scheduler and a commit protocol which takes transaction types (firm and soft) and transaction deadlines into consideration in order to minimize the number of transactions that must be aborted due to deadline violations. An efficient sub-transaction deadline assignment is also needed to carefully distribute global transactions' deadlines among their sub-transactions. In summary, none of the reviewed techniques can be applied directly to our proposed environment.

4. PROPOSED SOLUTION APPROACHES

4.1 Key Information Stored at Mobile Hosts

In our presented architecture, each MH will store some key information in its local database. The *ID* field will uniquely identify an MH. Every MH will get its coordinates from GPS (Global Positioning Scheme) [Ko, 1998] periodically and store them in the *Position* field. This position information will be used at the time of routing a transaction from a source MH to a destination MH. Each MH will also store the its *Radius of transmission range* in its local database. The *Energy_availability* field will record the amount of energy available at that time. This information is needed to identify the MH with the highest available energy at any point in time.

Apart from the above information, each *LMH* will maintain a *Global Schem*a, which is the integration of all local schemas from all *LMHs*. It will also store the corresponding *ID* of the *LMH* for each local schema. This *Global Schema* is required to identify which data object is stored in which *LMH*. Each *SMH* and *LMH* will maintain a field *LMH_List* which records the *Position*, *ID* and *Energy_availability* of each *LMH*. The *LMH*s will periodically broadcast their *ID*, *Position* and *Enery_availability,* and the *SMH*s and other *LMHs* will update their local databases after listening to the broadcast channel. *SMHs* will use the *LMH_List* to identify the nearest *LMH* and the *LMH* with the

highest available energy when needed. The key information stored at *LMHs* and *SMHs* are described in Table 1.

LMH	***SMH***	**Description**
ID	ID	Unique Identifier
Position	Position	Coordinator (obtained using GPS)
Radius of Influence	Radius of Influence	Radius of transmission range
Energy_availability	Energy_availability	Energy available at that point of time
Global_Schema	----------	Integration of all local schemas at each LMH and the corresponding ID of the LMH.
LMH_List	LMH_List	ID, Position and Energy_available of each LMH.

Table 1: Key Information Stored at Mobile Hosts

4.2 Transaction Property and Classification

In our real-time environment, transactions have deadlines and are classified into two categories: firm and soft [Gruenwald, 1997,1999]. Firm transactions must be aborted if they miss their deadlines while soft transactions still can be executed after their deadlines have expired. For firm transaction, the value of the transaction becomes zero after the deadline expires. From the value function describing tasks with soft deadlines in [Abbott, 1988], we can define soft transactions with two deadlines. A soft transaction still can be executed after its first deadline expires, but its value decreases after the first deadline and becomes zero after the second deadline. For example, in a battlefield environment, querying the position of enemy can be treated as a soft transaction with two deadlines. Since the position of the enemy will be changing frequently in this environment, the value of this transaction will decrease over the period of time and will become zero after a certain time period. Each transaction also has an attribute that records its run time estimate.

4.3 Transaction / Data Flow between MHs

In our architecture, there can be different types of transaction flows between the *LMHs* and the *SMHs*. The first case is when an *SMH* submits an entire transaction (from begin to commit) to an *LMH*

before it moves. The *LMH* executes the transaction and returns the results to the requesting *SMH*. This is the simplest case. The second case will be if we remove our assumption, that is, the *SMH* submits some sub-transactions of the transaction to an *LMH* and then moves to some other place. An *LMH* can also submit sub-transactions to other *LMHs*. Since the *SMHs* are not storing the whole DBMS in our architecture, we are not considering the transaction flow between two *SMHs*. But they will be able to communicate and send/receive messages to/from each other. Since routing in a mobile ad-hoc network uses multi-hops, an *SMH* can be a transaction initiator or a transaction forwarder. But the *LMH* can be a transaction initiator or a transaction forwarder or a transaction executer.

4.4 Transaction Management

Our proposed transaction management approach addresses the first case of transaction flow, i.e., the *SMH* will submit the entire transaction to the *LMH* before it moves. The transaction flow has two parts addressing how an *SMH* submits a transaction to an *LMH* and how an *LMH* executes a transaction submitted by an *SMH* and returns its result to the *SMH*.

4.4.1 How an SMH submits its transactions to an LMH

Since in our architecture, MHs have energy limitation and transactions have timing constraints, we need to have a transaction management policy that reduces the overall energy consumption while at the same time reduces the number of transactions that must be aborted due to missing their deadlines. Here we consider time as the most important factor in handling firm transactions and energy in handling soft transactions. So the *SMH* will submit its firm transactions to the nearest *LMH* so that the transactions can meet their deadlines. But for soft transactions, the *SMH* will submit them to the *LMH* which has the highest available energy. Here we are sacrificing the first deadlines of soft transactions in favor of energy consumption because soft transactions can still be executed after their first deadlines have expired. Thus when an *SMH* initiates a transaction, first it will check the type of the transaction. If the transaction is a

firm transaction, it will search the field *LMH_List* in its local database to find the *Position* and *ID* of the nearest *LMH*. Then it will find the route to this nearest *LMH* using some route discovery scheme. Here we can use the LAR Scheme 2 proposed in [Ko, 1998] because this technique uses only the position of the destination to find the route, and the local database of each *SMH* will store the position of each *LMH*. After finding the route, the *SMH* will submit the transaction along with its own *Position* and *ID* to the corresponding *LMH* using this route. The *LMH* needs the *Position* and *ID* of the requesting *SMH* in order to submit the result of the transaction after execution.

Now if the nearest *LMH* is in the active mode, it will receive the transaction and start processing the transaction. If it is in the doze mode, it will receive the transaction and examine the type of the transaction. If the transaction is firm, it will wake up and start processing the transaction in order to reduce the chance that the transaction will miss its deadline. After processing the transaction it will submit the result to the requesting *SMH*. But if the *LMH* is in the sleep mode, it will not be able to receive the transaction. In this case the requesting *SMH* will wait for some time period. If it does not receive the result of the transaction in this time period, it will assume that the nearest *LMH* is either in the sleep mode or disconnected. So it will again check its local database to find the next nearest *LMH*, find a route to this *LMH* and submit the transaction. The *SMH* can determine the length of the time period from the runtime estimate of the transaction, communication overhead and possible delay due to disconnection. If the transaction is soft, the *SMH* will find the *Position* and *ID* of the *LMH* with the highest available energy by searching the *LMH_List* in its local database. Then it will find a route to this *LMH* and submit the transaction to it. Again after waiting for a certain time, if the *SMH* does not get the result of the transaction, it will search the local database to find the *LMH* with the next highest available energy. If the requesting *SMH* moves after submitting a transaction to an *LMH*, it will inform the *LMH* of its new position. The algorithm is captured in Figure 2.

```
Let the Transaction be T1 with type T1_type, deadline T1_d, and runtime estimate T1_e.
Let the position of the requesting SMH be SMH_Position
Begin
        SMH initiates a transaction T1 with type T1_type, deadline T1_d and runtime estimate T1_e.
        Mark all LMHs in the LMH_List as not visited.
        While SMH has not received the result of T1 do
            Search the LMH_List.
            If T1 is a firm transaction
                Get the Position and ID of the LMH which is the nearest to the SMH and not yet visited.
            Else // T1 is a soft transaction
                Get the Position and ID of the LMH which has the highest energy available and not yet visited
            End if
            Find a route to the found LMH using LAR Scheme 2 [Ko, 1998].
            Submit T1 to the found LMH (SMH_ID, SMH_Position, T1, T1_type, T1_d, T1_e).
            Set WaitingTimePeriod = value. // maximum time for SMH to wait for the result
            While WaitingTimePeriod is not equal to 0 and SMH has not received the result of T1
                If SMH has not received the result of T1 // the found LMH is in the sleep mode //
                        If T1 is a firm transaction
                                If T1 missed its deadline
                                        Abort T1
                                End if
                        End if
                        Mark the found LMH as visited
                Else
                        Send an acknowledgement to the found LMH
                End if
            End while
        End while
End
```

Figure 2. SMH Execution Algorithm

4.4.2. How an LMH executes a transaction submitted by an SMH and returns its result to SMH

An *LMH* can receive two types of transactions: global transactions from an *SMH* or sub-transactions from other *LMHs*. Each *LMH* has three parts:

a. Transaction Scheduler (TS): schedules all global transactions and sub-transactions.

b. Transaction Coordinator (TC): divides the global transaction into sub-transactions and submits them to corresponding *LMHs*, and returns the results to the requesting MH.

c. Transaction Manager (TM): manages the execution of sub-transactions.

After receiving a transaction from an *SMH*, if the *LMH* is in the active mode, it will pass the transaction to the TS. If the *LMH* is in the doze mode, it will check the type of the transaction. If the transaction is firm, it will wake up and pass the transaction to TS. But if the transaction is soft, the *LMH*

will not wake up. Here we are sacrificing soft transactions for energy consideration because the *LMH* will usually go into the doze mode to reserve its energy, and for soft transactions, they still will be executed after they missed their first deadlines.

The TS at *LMH* will use a real-time energy-efficient dynamic scheduling algorithm (discussed in Section 4.4.3) to schedule transactions. The scheduling algorithm will organize the transactions in a queue that reflects their priorities of execution. Each time the *LMH* receives a new transaction (global transaction or sub-transaction), the TS schedules the transaction and places it in a proper position in the queue. Then the first transaction from the queue is taken by the TC. The TC checks the required data items for this transaction after consulting the *Global Schema*. If all the data items required by the transaction are available in this *LMH*, it will pass the transaction to its TM because the transaction is a local transaction. If all the data items are not available in this *LMH*, the TC will find the *LMHs* which contain the required data items from the *Global Schema*. Then it will divide the global transaction into sub-transactions and distribute the deadline of the global transaction among the sub-transactions using a deadline distribution algorithm. Here we can use the EQF Strategy proposed in [Kao, 1993] since it has been shown to perform better than other techniques. The type of the sub-transactions (i.e. firm or soft) will be the same as that of the global transaction. The transaction coordinator will find the routes to *LMHs* and submit the corresponding sub-transactions to them.

If we assume that the databases at *LMHs* are homogeneous, then we can use the two-phase commit protocol [Gray, 1993] to commit or abort the global transactions. But this protocol is two restrictive for a real-time mobile environment because the sub-transaction at each site has to wait for the decision of the TC for committing or aborting the transaction, and in the mobile environment, there will be frequent disconnections. So the TC can be in the disconnected state or can go into the doze mode or sleep mode. As a result there will be transaction blocking at the sites where the sub-transactions are executed. Moreover, it does not take care of deadlines of transactions. A lower priority transaction can block the data item which is required by a higher priority transaction and the higher priority transaction

can miss its deadline. During the time a site is waiting for the TC to make a global commit/abort decision, its sub-transaction may miss its deadline.

If the databases at *LMHs* are heterogeneous, we can adopt the PGSG Algorithm from PSTM Model [Dirckze, 1998,2000] presented in Section 3. But we have to tailor it for our environment with energy, ad-hoc networks, and real-time considerations. The requesting *SMH* has to define the vital and non-vital [Chrysanthis,1993] sub-transactions when it submits a transaction to an *LMH*. In that case the coordinator *LMH* (TC) will wait for the completion of all vital sub-transactions. If all the vital sub-transactions are completed, it can run the PGSG Algorithm [Dirckze,1998,2000] to verify the Atomicity/Isolation (A/I) properties. If the A/I properties are verified, then the *LMH* can decide to commit the transaction and submit the result to the requesting *SMH*. But if the A/I properties are violated and the transaction has not yet missed the deadline (for soft transactions second deadline), the *LMH* will restart the transaction. But if the A/I properties are violated and the transaction has missed the deadline (for soft transaction second deadline), the *LMH* will decide to abort the transaction and send the result to the requesting *SMH*.

After the TC has decided to commit a transaction, it will find a route (using LAR Scheme 2) [Ko, 1998] to the requesting *SMH* for submitting the result of the transaction. If the requesting *SMH* is in the active mode, it will receive the result and send an acknowledgement. If it is in the doze mode, it will examine the type of the transaction. If the transaction is firm, then in order to meet the deadline of the transaction the *SMH* will wake up, receive the result and send an acknowledgement to the *LMH*. But if the transaction is soft, it is up to the *SMH* whether the *SMH* should come into the active mode and receive the result or remain in the doze mode in order to reserve its energy for firm transactions. Another alternative for the soft transaction could be that the *SMH* will calculate the remaining slack time for the transaction using the equation (*Slacktime = d – t)* [Abbott, 1992] for deciding how long it can wait to receive the result, where 'd' is the deadline of the transaction and 't' is the current time. The slack time indicates how much time a transaction has left before it misses its deadline. The *SMH* will wait until the slack time is less than some time period value. Then it will wake up, receive the result and send an

acknowledgement. In the second approach, the deadline of the soft transaction is penalized in order to reserve energy in the requesting *SMH* because the *SMH* can receive the result of the soft transaction after its first deadline has expired. The value of the waiting period can be determined from the energy level of the requesting *SMH*. If the energy level of the *SMH* is sufficient, then the value of the waiting period will be less.

If the requesting *SMH* is in the sleep mode, it will not be able to receive the result. So if the TC of the *LMH* which has processed the transaction does not receive any acknowledgement till the deadline of the transaction, it will assume that the requesting *SMH* is in the sleep mode. Then it will check the type of the transaction. If the transaction type is firm, it will abort the transaction because a firm transaction has no values after its deadline has expired. But if the transaction type is soft and the requesting *SMH* is in the sleep mode, TC will calculate the slack time for the second deadline of the transaction. If this slack time is zero, it will abort the transaction. If it is not zero, it will divide the slack time into some time intervals and will submit the result again to the requesting *SMH* during those intervals. The motivation behind this technique is that since the transaction is soft and it consumes sufficient energy to transmit, the *LMH* will not continuously keep sending the result to the sleeping *SMH* and lose its energy. The length of the time-interval will depend on the remaining slack time of the transaction and the energy level of the *LMH*. If the energy level is low, the interval will be large and the number of transmissions will be small. The *LMH* execution algorithm is captured in Figure 3.

Begin

LMH receives a transaction T with ID T_ID, transaction type T_type, deadline T_d, Runtime estimate T_e, Requester ID R_ID, Requester position R_pos, Requester energy R_energy, Data item List L.
Schedule (T_ID, T_type, T_d, T_e, R_energy) and take the first transaction from the queue.
(Let the first transaction be Tf with id Tf_ID)
Check Tf_ID of the transaction.
If *Tf_ID is for a sub-transaction // Tf is a local transaction*
Execute the sub-transaction.
Submit the result to the coordinator LMH.
Else *// Tf is a global transaction*
Search the global schema.
Get the LMH_list for Data item List L.
Divide the Global Transaction into sub-transactions.
(Refer to this list of sub-transactions as S_list)
Distribute deadline Tf_d among S_list using EQF Strategy [Kao,1993].

```
For each LMH in the LMH_list
        Find a route to LMH using LAR Scheme 2 [Ko, 1998]
        Submit the corresponding sub-transaction from S_list to this LMH
End for
Wait until all the vital subtransactions are completed
Run PGSG Algorithm [Dirckze, 1998,2000].
If PGSG Algorithm's outcome is to abort Tf due to A/I violations
        If slack time of Tf  is greater than 0  // For soft transactions, calculated using second
                                                // deadline
                Restart Tf by executing this LMH execution algorithm again
        Else
                Abort Tf
                Send the result to requesting SMH
        End If
Else  // PGSG Algorithm's outcome is to commit Tf
        Submit result to the requesting SMH
        While LMH has not received an acknowledgement  (Ack) from requesting SMH
                                                and  slack time of Tf is greater than 0 do
                If LMH has received an Ack
                        Remove transaction from active transaction list
                End if
        End while
        If LMH has not received an Ack from requesting SMH  // SMH is in sleep mode
                If Tf_type is firm
                        Abort the transaction.
                        Submit result to  requesting SMH
                Else            // Transaction type is soft
                        Find the slack time using the second deadline
                        If slack time =0
                                Abort the transaction
                                Submit result to requesting SMH
                        Else
                                Divide the slack time into i intervals
                                While this loop  has not been executed i times and LMH has
                                        not received an Ack from requesting SMH do
                                    Submit the result to the requesting SMH
                                    Set WaitingTimePeriod = interval
                                            // maximum time that LMH waits
                                            // for an Ack from requesting SMH
                                    While LMH has not received an Ack  and
                                                        WaitingTimePeriod ¹0 do
                                        If LMH has received an Ack from requesting SMH
                                                Commit transaction
                                                Remove transaction from
                                                        active transaction list.
                                                Submit result to requesting SMH
                                        End if.
                                    End while
                                End While
                        End if
                End if
        End if
End if
End
```

Figure 3 : LMH Execution Algorithm

4.4.3 Scheduling Algorithm

An *LMH* will be receiving transactions from *SMHs* and other *LMHs*. It has to assign priorities among transactions in order to schedule them. In our environment the scheduling algorithm has to consider not only transaction types (firm and soft), transaction deadlines, but also the energy limitations of the *MHs*. Here we can use the Least Slack (LS) cognizant technique proposed in [Abbott, 1992] with certain modifications with respect to energy constraints, disconnections and transaction types. In the LS technique, transaction with less slack time is scheduled before transaction with more slack time. In our scheduling algorithm, first we will sort all the transactions with respect to their slack times irrespective of their transaction types. Here to calculate the slack times, we will use the equation used in [Abbott, 1992] with certain change. The slack time, 's' of a transaction can be calculated using the following equation,

$$s = d - (t + c + Pd*Td) \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad (1)$$

where 'd' is the deadline, 't' is the current time, 'c' is the runtime estimate, 'Pd' is the probability of disconnection during execution and 'Td' is the average time loss due to disconnection. Now if two firm transactions have the same slack time, then a higher priority will be given to the one whose requester has less amount of available energy. The reason is that because the requester which has less energy will exhaust its energy earlier, it is better to schedule its transactions earlier. We assume that MHs while submitting their transactions/sub-transactions to the LMH will also send their energy-level to the *LMH*. The same technique will be adopted if two soft transactions have the same slack time. If the slack time of a firm transaction is equal to the slack time of a soft transaction, then a higher priority will be given to the firm transaction considering that firm transaction will be aborted if it misses the deadline. Now if the slack time of a soft transaction is found to be negative, then its slack time will be recalculated considering its second deadline. If the recalculated slack time is again found to be negative, then the transaction will be discarded. The algorithm is captured in Figure 4.

Begin

Calculate the slack time for all transactions using Equation 1.
Sort all the transactions according to their slack times.
Assign higher priorities to transactions with shorter slack times.
If *two firm transactions or two soft transactions have the same slack time*
Then give priority to the one whose requesting MHs has less energy.
End if
If *the slack time of a firm transaction is equal to the slack time of a soft transaction*
Then give a higher priority to the firm transaction.
End if
If *the slack time of a soft transaction is negative*
Recalculate the slack time using its second deadline and equation 1.
End if
If *the recalculated slack time of a soft transaction is negative*
Discard the soft transaction.
End if

End

Figure 4. Energy-Efficient Real-Time Transaction Scheduling Algorithm

5. CONLUSIONS

In this paper, we have introduced an architecture for a real-time mobile ad-hoc environment that typically exists in battlefields and disaster recovery situations. We have addressed the new issues i.e. mobility of servers and users, time criticalness and energy limitations related to transaction management for this environment. We have also given partial solutions to these issues. More detailed study of transaction management for this environment is required. For example, we need to determine how long an *SMH* should wait after submitting its transactions to an *LMH* to determine the mode of the *LMH*, how an *SMH* should deal with its own data, and whether a cache should be used at an *SMH* to improve the performance of transactions it has initiated. Another important issue in environments like battlefield is that there will be some transactions which cannot be compensated. So we need to treat this type of transactions in a different way. As part of our future work, we will develop a complete transaction management technique to solve all the addressed issues. Then we will build a simulation model to analyze the performance of our technique in terms of energy optimization in mobile hosts and number of transactions meeting their deadlines.

REFERENCES

[Abbott, 1988] Abbott R., H. Garcia-Molina, "Scheduling Real Time Transactions", SIGMOD RECORD, Vol. 17, No. 1, March 1988.

[Abbott, 1992] Abbott R., H. Garcia-Molina, "Scheduling Real Time Transactions: A Performance Evaluation', ACM Transactions on Database Systems, Vol. 17, No. 3, September 1992.

[Bandvopadhvav, 1999] Bandyopadhyay, S., and K. Paul, "Evaluating the Performance of Mobile Agent-Based Communication among Mobile Hosts in Large Ad-Hoc Wireless Network", MSWIM 1999.

[Barbara, 1994] Barbara D., T. Imielinski, "Sleepers and Workaholics: Caching Strategies in Mobile Environments", ACM SIGMOD, may 1994.

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

[Dirckze, 1998] Dirckze, R., and L. Gruenwald, "A Toggle Transaction Management Technique for Mobile Multidatabases", *ACM Conference on Information and Knowledge Management*, November 1998.

[Dirckze, 2000] Dirckze, R. and L. Gruenwald, "A Pre-serialization Transaction Management Technique for Mobile Multi-databases ", To appear : Special Issue on Software Architecture for Mobile Applications, MONET 2000.

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

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

[Gruenwald, 1999] Gruenwald, L., et al., "Database Research at The University of Oklahoma", ACM SIGMOD RECORD, Vol. 28, No. 3, September 1999.

[Hong, 1999] Hong X., et al, "A Group Mobility Model for Ad Hoc Wireless Networks", MSWIM, 1999.

[Kao, 1993] Kao B., H. Garcia-Molina, "Deadline Assignment in a Distributed Soft Real-Time Systems", Proceedings of the 13th International Conference on Distributed Computing Systems. May 1993.

[Ko, 1998] Ko, Y., N. Vaidya , "Location-Aided Routing (LAR) in Mobile Ad-Hoc Networks", MOBICOM 1998.

[Liu 1999] Liu, M., et al., "Modeling and Simulation of large Hybrid Networks", Proceeding of 2nd Annual Advanced Telecommunications/ Infrastructure Distribution Research Program (ATIRP) Conference 1999.

[Madria, 1998] Madria, S. K., B. K. Bhargava, "A Transaction Model for Mobile Computing", International Database Engineering and Application Symposium (IDEAS 1998), July 1998.