Data Replication in Ad Mobile Databases

Abstract:

Increase in the mobile computing has forced to explore the possibilities of more efficient, reliable and cost effective resources for mobile computing. Mobile hosts run on battery power and communicate with each other via wireless links of limited bandwidth. Hence, it is necessary to incur low communication cost during the transfer of information from one mobile host to another. In an ad hoc environment, ensuring reliability of obtained information is a major problem due to frequent partitioning of the network as a result of the mobility of servers. Hence, it becomes absolutely necessary to improve the processing time for transactions while respecting battery power considerations and ensuring the reliability of information. Proper data replication can result in achieving the desired results.

1 Introduction:

Replicating data at servers increases the local availability of the data thereby resulting in an over all reduction of computing cost in a mobile environment. Data Replication in Ad hoc mobile databases is gaining importance due to the need for an effective replication scheme and various complications involved in replicating data for ad hoc system. Various replication schemes have been developed for data replication over the years for static distributed systems. Enhancements over the basic schemes devised for distributed systems have been proposed for data replication in mobile databases. However, there are very few techniques that capture the ad hoc mobile databases.

In a mobile environment a static network interconnects the stationary servers. Hence for data replication, it is essential to determine the location of replica mainly on the mobility pattern of the mobile client. However, in a mobile ad hoc system, not only the clients but also the servers are moving. Hence, additionally it is necessary to consider the mobility pattern of the servers for replica allocation. Depending on the mobility pattern of the client a replica might be allocated at a server currently at the location where the client is supposed to access a remote data item. However, since the server moves, the client may not be able to access the required data item locally, thereby making the data replication scheme futile. Thus, it becomes difficult to implement data replication for ad hoc mobile databases.

Both static and dynamic replication schemes have been proposed for distributed databases. In static replication schemes the number and location of replicas are determined earlier and are fixed, while in dynamic schemes the replicas are computed dynamically on the basis of access patterns [Wu, 1997]. In an ad hoc environment, as a result of the mobility of clients as well as the

servers, the access patterns often change. Hence, for an ad hoc mobile system some kind of dynamic replication scheme that adjusts itself to the changing access pattern is required.

The paper introduces a new data replication scheme that takes into consideration the mobility issues of the servers, the associated energy levels of each server before assigning replicas. The paper also focuses on maintaining the accuracy of queries, one of the main drawbacks of the technique proposed by [Karumanchi, 1999]. This paper reviews in section 2 some of the earlier data replication techniques proposed for mobile and ad hoc mobile databases. Section 3 proposes a technique for data replication in ad hoc mobile databases. The theoretical analysis of the proposed technique and the technique by [Karumanchi, 1999] is presented in section 4 and 5 respectively. Section 6 compares the proposed technique and the technique by [Karumanchi, 1999]. Finally section 7 presents the conclusions and recommendations for future work.

2 Review of Existing Techniques:

Various data replication techniques have been proposed for fixed distributed databases and mobile databases. All the replication techniques mainly aim at maintaining consistency of replicated data and reducing the cost of maintaining the replicas. The replication schemes compare the cost savings in maintaining a replica at a site as compared to the cost incurred in updating the replica. The cost savings in maintaining a replica at a site is the difference in the cost incurred in reading a data item from the remote server and the cost incurred in reading the same data item locally. If the cost savings is more as compared to the cost incurred in maintaining the replica at a site, a replica is allocated at the site; otherwise no replica is allocated.

[Wu 1997] has proposed an active replication scheme for mobile databases. The scheme aims at replicating a single(set) of data item(s) at a given server if the number of reads issued from the

server is greater than the number of writes/updates in the system on the same data item. Unlike other dynamic replication schemes, this scheme takes into consideration the number of read and write transactions presently active in the system on a given data item. Every user maintains a daily schedule in his/her profile that helps in actively deciding whether a replica has to be allocated at the current location. As specified in the daily schedule, when the user reaches the pre-specified location at a given time, a decision is made beforehand regarding replicating the data item at the current location. However, when the user does not reach the pre-specified location at a given time, like other dynamic algorithms, the replication decision is based on the past histories of reads and writes on the concerned data item. If the number of read histories is greater than the write histories, then a replica is allocated at the concerned site. This scheme relies heavily on submission of schedules. In a mobile ad hoc environment, as the servers are also mobile, submitting schedules ahead of time would not help.

[Huang, 1994] proposes a data replication scheme that aims at optimizing the communication cost between a mobile computer and the stationary computer. The stationary computer can be identified as a server that stores the online database and mobile computer can be considered as laptops carried by mobile users. The authors have presented and analyzed both the static and the dynamic data allocation schemes. In the static allocation scheme a copy is either stored only at the server or stored at both the server and the mobile computer. Accordingly, the static allocation scheme is either called a one-copy scheme or a two-copy scheme. The author does not discuss the concept of primary copy hence in a two-copy scheme any of the copy can be updated. This results in update of the other copy. In the dynamic allocation scheme, a single window of size *k* is maintained for each data item either at the mobile computer or the server. The window

maintains a list of read and write requests on a given data item. If the number of reads is greater than the number of writes and there is no copy at the mobile computer, a copy is allocated at the mobile computer. Also the window of requests is passed to the mobile computer. Any further write request to the stationary computer are propagated to the mobile computer. The mobile computer then checks if the number of writes is greater than the number of reads. If the number of reads is greater, the copy at the mobile computer is simply updated. If the number of reads is less than the number of writes and there is a copy at the mobile computer, the copy at the mobile computer is deleted and the window is passed to the stationary computer. Any further request to the stationary computer is not propagated to the mobile computer.

[Shivakumar, 1997] presents a technique called *per-user replication scheme* that maintains replicas at sites where the cost savings in maintaining the replicas is maximum. This technique is an improvement over the pure Home Location Register (HLR) scheme and the HLR/Visitor Location Register (VLR) scheme [Mohan 1994].

In the HLR scheme, a call placed by a user results in a remote lookup at the HLR of the callee. A home location refers to the site where a user is registered. A visitor location refers to the site where the user is presently in. This site is any site other than the user's home location. A call initiated in the HLR/VLR scheme results first in querying the local database for the callee. If the callee's profile is not found, the home location of the callee is queried.

The *per-user replication scheme* maintains an up-to-date copy of the user's profile at the user's HLR and also determines other sites where the user's profile has to be replicated. The scheme

assumes that the home location has the knowledge of all the sites where the replicas reside and update of all the replica profiles is initiated by the home location. The home location initiates the update of the replicas when the user crosses zones. This scheme first specifies the maximum number of replicas to be stored at the site and the maximum number of replicas that can be allocated to each user profile. The replication scheme computes the maximum number of replicas and the location of these replicas using the minimum-cost maximum-flow algorithm [Ahuja, 1993]. The minimum-cost maximum-flow algorithm finds the databases to which the user profiles should be assigned such that the number of replicas is maximum while minimizing the system cost in maintaining these replicas. For all the user profiles, the algorithm first computes the cost savings in maintaining a user replica at all the sites and the maximum number of distinct replicas that can be stored at that site.

Let Zi represent the i^{th} site and Pj the j^{th} user. The algorithm chooses a particular Zi and Pj pair such that the cost savings in maintaining a replica of the profile of Pj at Zi is highest among those computed and allocates a replica of the profile of Pj to Zi. The algorithm then chooses the pair with the next highest cost savings and allocates the replica to that site. The algorithm continues either till all the zones have been allocated the maximum number of replicas they are supposed to store, or for each user, the maximum number of replicas to be allocated has reached. The algorithm is periodically executed at a centralized site and the replication plan is propagated to other sites. When a user moves from one zone to another, the HLR updates the replicas. A centralized site computing and allocating replicas sounds reasonable for a mobile environment; however for a mobile ad hoc environment, this is not desirable as the central server might be in a partition different from the partition of some of the other servers who receive the replication

plan. This would prevent providing the replication plan to the servers in the different partition, thereby seriously affecting the usability of this technique.

Very few replication techniques have been proposed for mobile ad hoc databases. The main problem in mobile ad hoc databases is to maintain consistency of replicated data due to the mobility of servers. Also, since the servers run on battery power it is necessary to take into consideration the available energy levels before assigning replicas.

[Karumanchi, 1999] presents a data replication scheme for mobile ad hoc networks. To increase the availability of the data in the event of frequent network partitions, the authors have proposed three schemes for information dissemination. In all the three schemes, a mobile host randomly chooses a *quorum* and sends a read/update message to the servers represented in the selected quorum. A quorum consists of a group of servers, the exact of description of which is described later. A transaction succeeds if at least one server responds positively to the request.

The system model consists of *'N'* firefighters. *'n'* of the *'N'* (n << N) firefighters are officers who collectively manage the entire operation. Each firefighter has a small wireless communication device. The firefighter updates his/her location and state information, and is also able to obtain the latest information about all the other firefighters. The officers' devices have additional storage space that is utilized to maintain the state information and act as servers for the firefighters. The firefighter receives/gives update information from/to the officers. No firefighter is assigned to a particular officer in advance. The firefighter is free to choose any officer within its range.

The technique uses a quorum-based solution to achieve replication in ad hoc mobile networks. The servers are divided into quorums such that the union of all the quorums results in the whole server set. Additionally the intersection of any two quorums always has at least one server in common. The quorums are constructed in advance and all the nodes in the system are assumed to know to which particular quorum a given server belong.

Three strategies namely, 1) Select_then_Eliminate (STE), 2) Eliminate_then_Select (ETS), and 3) Hybrid have been suggested for identifying servers for updates and queries. In all the three strategies the servers maintain a disqualified list DQL for each server. The DQLx maintains a list of servers that are not reachable from server x.

The Select_Then_Eliminate (STE) strategy requires a node *x* to randomly select a quorum Si. Assuming that DQLx is initially empty, the node sends the query/update message to all the servers in the quorum. The node then waits a certain time $T_{timeout}$ for the reply from the servers. In case of update message the node receives an acknowledgement from the server. For queries the server would reply with a query answer or a NULL value with the timestamp for each. If at least one server responds with an acknowledgement or NULL/Non Null reply, the operation is said to be succeeded. If any server (s) does not respond within the time $T_{timeout}$, the node then adds this server to DQLx. The server (s) entry remains in DQLx for *disqualification duration* δ_{DQL}. Suppose after some time (assuming mean time *x* has issued some updates and queries), *x* again sends an update message and randomly picks up the quorum Si. Assuming *s* is still in DQLx, *s* is not reachable from node x. Hence, the node x does not have to send the update request to *s*. The node *x*, therefore eliminates *s* from the set of servers to whom the update message is sent. Thus *x* ends up sending update message to Si' = Si –DQLx.

In the Eliminate_Then_Select (ETS) strategy, a mobile node x first eliminates any quorums that have at least one server in the DQLx list. Servers in the DQLx list represent servers that are not reachable from node x. The mobile node x , then randomly selects one of the quorums from the remaining quorums for update/query. Hence the name Eliminate_then_Select.

Here the node *x* eliminates all the quorums that have at least one node in DQLx. Of the remaining quorums, one is randomly selected and update/query message is sent accordingly. If some servers do not respond with in the time $T_{timeout}$, then these servers are added to DQLx for δ_{DQL} time. For the next update/query, the quorums that have at least one server in DQLx are eliminated.

The ETS strategy eliminates those quorums that have at least one server not reachable from the mobile node x. According to the mobile node x's perception all the servers in the remaining quorums are reachable from x. This strategy thus tries to maximize the number of servers in one quorums that receives the update/query. As a result, the possibility of updating more number of servers increases. This in turn increases the possibility of getting more recent value of the data item.

The Hybrid strategy makes use of ETS for updates thereby increasing the accuracy of future queries. For queries this strategy uses the STE strategy thereby maximizing the availability of information. As explained earlier, the ETS strategy aims at maximizing the number of servers in a quorum that receives the update thereby maximizing the accuracy of queries. When updates are sent to more the number of servers, the probability that a query results in accurate information is also high. ETS strategy however reduces the choice of quorums for updates, thereby affecting the availability of information. In the STE strategy on the other hand, all the quorums in the system

are available thereby Increasing the availability. Within a partition, the queries should result in the most recent value of the data item. Since the ETS strategy maximizes the number of servers with in a quorum receiving an update and STE gives choice of all the quorums for queries the possibility of obtaining the most recent value of the data item increases.

The ETS strategy selects a quorum, which has all the servers in quorum with in its reach. Actually it may happen that one/more of the servers might not be reachable from the node, but is not reflected. This is because the server(s) has(have) moved away. Thus, the updates are sent to all the available servers in the quorum. Also when some other node initiates a query using the STE strategy, more accurate information is available regarding the update of the earlier node as each of the servers is represented in one of the pre-decided quorums. The hybrid strategy thus aims at increasing the number of servers that receive the update. This results in queries that retrieve more accurate information.

STE, ETS and Hybrid Strategy Comparison:

ETS aims at maximizing the servers that receive the updates, thereby increasing the accuracy of queries. In ETS, a quorum with no server in the disqualified list is obtained. This basically results in lesser quorums available for update, but maximizes the number of servers within a quorum that receive the updates/queries.

Meanwhile, STE eliminates only those servers in the quorum that are not within the reach of the node that initiates the update/query. Thus the node has more choice of quorums, thereby increasing the availability of the information.

In STE, a query/update might be sent to a quorum with very few servers reachable. Thus only few servers receive the update, thereby reducing the chances of obtaining accurate information

for queries. The possibility of choosing a quorum with fewer servers for querying can further degrade the accuracy of the quorum.

The hybrid strategy uses the positive features of STE and ETS. It increases the accuracy of queries by using the ETS strategy for updates and maximizes the availability of information for queries by using the STE strategy.

3 GBRMAD Technique

This section proposed an energy-efficient technique for data replication in ad hoc mobile databases. The technique by [Karumanchi, 1999] for ad hoc mobile databases does not take into consideration the energy efficiency issue. Also the consistency of replicas and accuracy of queries is not guaranteed in this technique. The proposed technique called the Group Based Replication for Mobile Ad Hoc Databases (GBRMAD) aims at achieving the accuracy of queries and decreasing the energy consumption of the mobile nodes.

The GBRMAD technique divides the servers into groups such that the intersection of any two groups does not have any servers in common. The technique uses a variation of the Majority Voting Strategy [Selinger, 1980] as explained later for an update/query. This helps in obtaining the most recent value of the data item for future queries. A replica is allocated at the server if the number of read transactions originated at the server is greater than the number of updates/writes in the system. Otherwise the data is read from the remote server where the primary copy resides. The technique introduces different modes of operations (active, doze and sleep) thereby addressing the energy efficiency related issue.

Subsections 3.1 and 3.2 present the system model and the assumptions respectively. The definition of groups and how to divide server into server groups is presented in subsection 3.3 and 3.4 respectively. Subsection 3.5 explains the concept of primary and secondary groups. Finally subsection 3.6 presents the replication plan.

3.1 System Model:

As proposed by [Gruenwald, 2000] the system model is described as follows. *"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. So, we have defined our architecture considering the battlefield environment as illustrated in Figure 1. Depending on communication capacity, computer power, disk storage, size of memory and energy limitation, MHs in the military network architecture can be classified into two groups: 1) computer 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 Hosts (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 often than humvees. Both humvees and tanks are more static than SMHs. 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 shape 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. The communication link between two MHs is shown with dark dotted lines in Figure 1. 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 intermediated MHs between them. For example, in Figure 1, SMH 11 will not be able to communicate directly with LMH 3 because their radii of influence are not overlapping, 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 the 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.*
3) *Sleep Mode: Both the CPU and communication device of the MH are suspended.*

Due to energy and storage limitations, we will assume that only LMHs will store the whole Database 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."

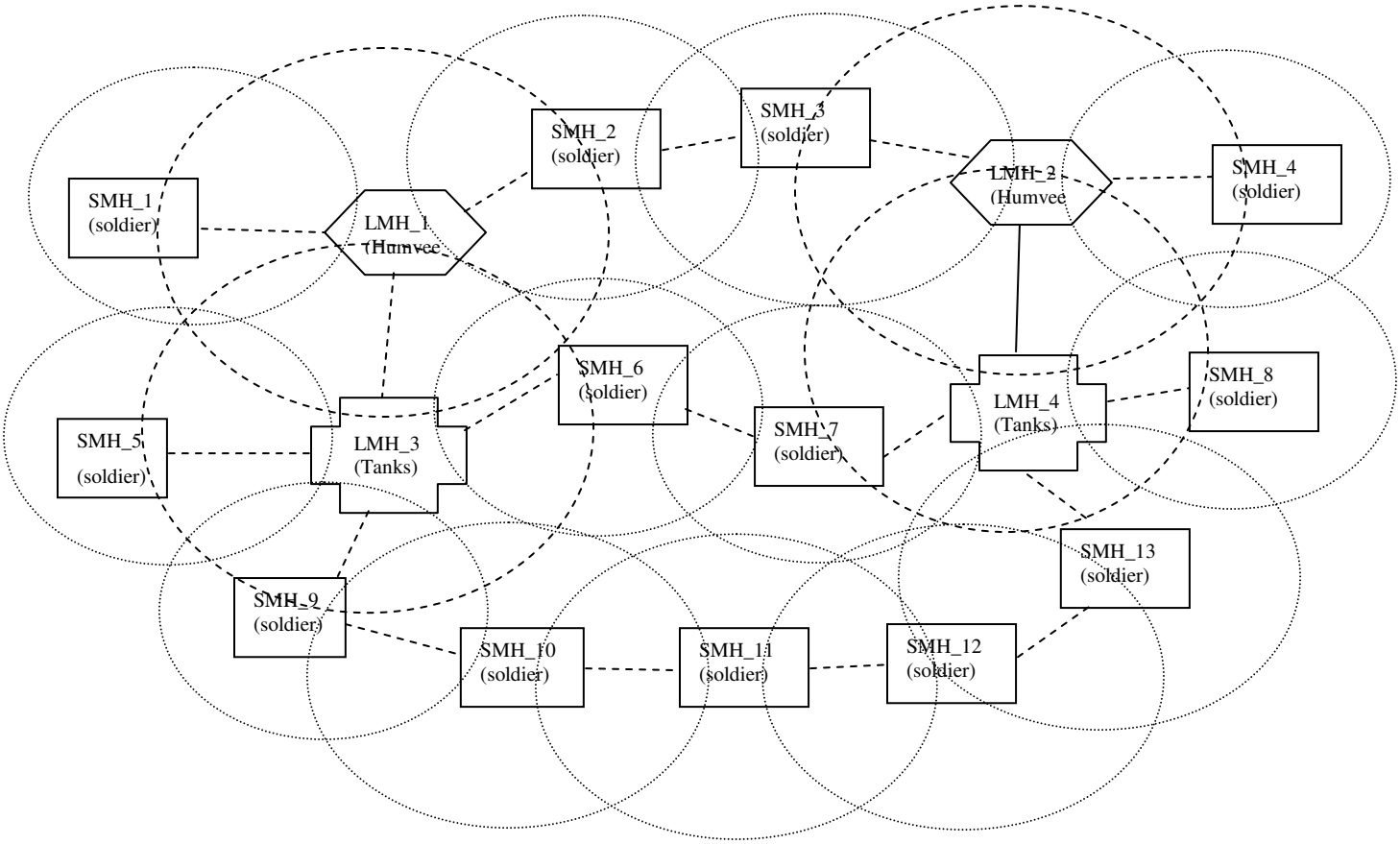

Figure 1: System Model

3.2 Assumptions

For any replica control protocol, the conditions to be studied are consistency, availability, low processing cost, and access frequency. Apart from these conditions, since the ad hoc mobile hosts have limited battery power, it is necessary to take into account the energy levels of the MHs before assigning replicas.

The GBRMAD technique makes the following assumptions.

1) The LMHs as explained later are divided into groups. For dividing LMHs into server group the energy levels of each server, the storage capacity of each server, and the number of

servers are taken into account. To simplify calculations all the LMHs are assumed to have same storage capacity and this capacity is sufficiently large.

2) Every transaction can be critical or non-critical. A critical transaction is capable of forcing a MH in doze mode to go into active mode.
3) A transaction can be classified as firm or soft [Gruenwald, 2000]. A firm transaction has to be aborted if its deadline is missed. A soft transaction is to be executed even if its deadline is missed.
4) Transactions are also be classified as normal and Most Recent Value (MRV) transactions. A MRV transaction requires the most recently updated value of the required data items (example, transaction that requires the position of a stock). A normal transaction can read the data item from the nearest available site (example, transaction that requires the weather forecast).
5) The number of read accesses for all LMHs is the same.
6) All transactions are compensatable [Dirckze, 1999]. When it is possible to undo the changes made by a transaction after committing the transaction, such a transaction is called a compensatable transaction.
7) The number of reads is much greater than the number of writes for all data items.

Generally, for a fixed distributed or mobile environment, among the number of copies of a data item, one copy is assigned as the primary copy of that data item. Any update to data item is always first reflected in the primary copy of that data item. In a mobile ad hoc environment, due to the mobility of the nodes, it is possible that the area of influence of nodes may not overlap. Also, it is possible that any two MHs cannot communicate with each other via single or multiple

hops. In such a situation the two MHs are said to be in different partitions of the network. Since the occurrence of network partitions is often in a mobile ad hoc environment, there is a possibility that data items required by a MH in network partition 1 may actually lie in network partition 2. There is no way for the MH to obtain the required data item. So to increase the availability of a data item, instead of having one primary copy of the data item, a number of primary copies of the data item are stored at different servers. The set of servers that store these primary copies of the data item can be classified as a group. These groups are predefined an explained in the following sub-sections.

3.3 Group Definition:

Let S1, S2, …….., Sn be *n* LMHs.

Let C1, C2, ……., Cn and E1, E2, …… En be the initial storage capacity and energy level of n servers, respectively. Let SGi denote the i^{th} server group.

Let the servers be divided into m groups such that:

$$1) \bigcup_{i=1}^{m} SGi = \text{Set of all servers}$$

$$2) SGi \cap SGj = \varnothing, \text{ for } 1 \le i, j \le m$$

In other words, the union of all groups result in the whole server set, and the intersection of any two groups is always an empty set. This means that there is no server in common to any two groups. Servers are divided into groups such that the primary copies X_{prim} of a data item X are assigned to a given single group. The intersection of server groups is an empty set so that the primary copies of the same data item do not lie in multiple groups. This helps in identifying to what group does a required data item belong. Let X_{prim} be the primary copies residing at the servers in a group SGi. A replica/secondary copy of X called X_{sec} is dynamically allocated at any

server belonging to a server group SGj such that i≠j depending on the replication decision made by the replica control protocol. The servers can be divided into m groups depending on their individual storage capacity, energy levels, and number of times a given server group is accessed as explained in section 3.4. As mentioned before the initial storage capacity is identical for all servers and is sufficiently large.

3.4 Division of servers into server groups:

Let Ai be the average number of read accesses for all the data items in a given server group SGi. In our system, the number of reads is considered to be much greater than the number of writes for all the data items. If the number of update/write accesses were high, it would not be advisable to have multiple copies of the data item since for every update/write all the copies of the data item have to be updated. This in turn would increase the overhead on the system. In our system since the read accesses are much higher than the write accesses, only the number of read accesses are considered for computing groups. Since the copies are replicated at all the servers in a server group, Ai can also be viewed as the number of read accesses on a given server in the server group SGi. The greater the number of reads a server performs, the greater the necessity would be to allocate more replicas for the data items residing at these servers.

For a server group that has a higher value of Ai, it would therefore be advisable to have more replicas. The number of servers to be allocated to each group SGi can then be calculated using the ratio of Ai of the i^{th} group to the aggregate Aj's of all the groups. It is necessary that the servers of a given energy level have to be distributed according to the number read accesses of a given group. This helps in avoiding the possibility of giving all the high energy level servers to one group and low energy level servers to the other.

The algorithm for dividing servers among server groups first assigns the tentative number of servers of given energy levels to each server group. The process of assignment is later explained in detail with the help of an example. The algorithm then finds the remaining number of servers of the given energy levels and the additional number of servers required by each group. The un-assigned servers are then allocated first to the group having the highest read access ratio depending upon the server requirements of that group. Once the servers have been assigned to this group, the remaining un-assigned servers are then allocated to the group having the next highest read access ratio and so on. The requirements for the group having the highest read access ratio are satisfied first as the error ratio (ratio of assigned energy level to required energy level) would be less for servers requiring higher energy levels. This is because as explained earlier, the higher the read access ratio is, the greater is the energy level requirement. For example, by assigning an extra server of energy level 100 units to a group having high energy level requirements the error ratio would be less as compared to assigning an extra server of energy level 100 units to a group having a low energy level requirements. A positive error ratio indicates that the assigned energy level is greater than the required energy level and vice versa.

The process of assigning servers to group first starts with assigning the servers to classes. Each available energy level is considered to be a class. Arrange servers with the same energy levels to a particular class. Therefore if there are J different energy levels there will be J classes. For example, let there be five servers (S1, S2, S3, S4 and S5). Let 100 units be the energy level of servers S1 and S2, and 60 units be the energy level of servers S3, S4 and S5. Since there are two different energy levels (60 and 100 units), there are two classes ($class_{60}$ and $class_{100}$ respectively). Servers S1 and S2

Let E_L represent the energy level of each server in the L^{th} class and $N(E_L)$ be the total number of servers in the L^{th} class of servers, where L ranges from 1 to J. The number of servers of a given energy level to be assigned to the i^{th} group SGi is given by

$$N_{SGi}^{E_L} = \frac{Ai}{\sum_{k=1}^{m} Ak} \times N(E_L)$$

The group energy level of each group is the sum of the energy levels of individual servers in the group. The group energy level is directly proportional to the number of read accesses on a given server group. It is necessary that the group having higher number of accesses have higher energy levels, as the depletion of energy would be higher.

A J×m matrix with m group as columns and J energy levels as rows can be constructed. The groups are arranged in the decreasing order of their access ratios and the energy levels. The entries in the matrix represent the tentative number of servers of a given energy level for a given group. These entries are obtained from the above equation. These entries can either be an integer or a real number. The sum of the number of servers in a column gives the number of servers required for each group. This sum is rounded off to the nearest integer and is denoted by Nsum. For each column the decimal part is neglected and the sum of the integer part gives the number of servers of each class that are to be assigned to each group. The sum of the integer part in the column gives the number of servers that are assigned to each group and is denoted by Nint. The difference of Nsum and Nint gives the additional number of servers that are required by each group and is denoted by Req_servers. The remaining servers are then assigned to the groups that require additional servers.

The process of assigning the additional servers starts with assigning the required number of servers to the group having the highest read access ratio followed by the next highest read access

ratio and so on. Naturally the group having higher read access ratio shall have higher energy requirements and hence higher energy level servers are assigned to the group having a higher read access ratio. The algorithm and the example presented herein depicts the process.

Notations for the algorithm:

m = number of server groups

n = total number of servers in the entire system

J = number of different energy levels available

E_{ARRAY} = an array of length J storing the available energy levels

N_{ARRAY}= an array of length J storing the number of servers belonging to each class.

N_{ARRAY} [k] = array location storing the number of servers belonging to class k. In other words the number of servers having energy level Ek

N_{MATRIX}= a matrix of size J × m that stores the tentative number of servers of a particular energy level to be assigned to each group

A_{MATRIX} = a matrix of size J × m that gives the actual number of servers of a particular energy level to be assigned to each group

R_{ARRAY} = an array of size m that stores the number of read accesses for each group

A_{RATIO} = an array of size m that stores the access ratios (ratio of the read accesses for each group to the total number of read accesses for all groups) for each group

N_{SUM} = an array of size m that stores the total number of servers required for each group. This is obtained from N_{MATRIX}

N_{INT}= an array of size m that stores sum of the integer part of the total number servers required for each group. This is obtained from N_{MATRIX}

Add_{ARRAY} = an array of size m that stores the additional number of servers required for each group

Input:

m, n, J, E_{ARRAY}, R_{ARRAY} , N_{ARRAY}

Output:

A matrix A_{ARRAY} that indicates the number of servers of each energy level to be assigned to each group.

Algorithm for dividing groups

```
// Sort energy level in decreasing order such that E_ARRAY(1) represents highest available energy
// level and E_ARRAY(J) the lowest energy level
FOR k = 1 to J-1
        FOR p = 2 to J
                IF E_ARRAY(k)  <  E_ARRAY(p)          THEN
                        temp := E_ARRAY(p);
                        E_ARRAY(p) := E_ARRAY(k);
                        E_ARRAY(k) := temp;
                END IF;
        END FOR;
END FOR;

// Calculate the ratio of the number of read accesses for each group to the total number of read accesses of all
// groups
Asum := 0
FOR k = 1 to m
        Asum :=  Asum + R_ARRAY(k);
END FOR;
FOR k = 1 to m
        A_RATIO(k) = R_ARRAY(k) / Asum;
END FOR;

// Arrange the ratios in their decreasing order
FOR k = 1 to m-1
        FOR p = 2 to m
                IF A_RATIO(k) <  A_RATIO (p) THEN
                        temp := A_RATIO(p);
                        A_RATIO(p) := A_RATIO(k);
                        A_RATIO(k) := temp;
                END IF;
        END FOR;
END FOR;
```

```
// Compute the number of servers required of each energy level for each group
// The computed values are stored in a J ×m matrix. Each column indicates the number of servers of each energy
// level  required for each group. Each row indicates the division of  servers of given energy level among all groups.
FOR p = 1 to J
        FOR k = 1 to m
                N_MATRIX(p, k) := A_RATIO(k) * N_ARRAY(p);
        END FOR;
END FOR;

// Add the number of servers required for each group and round off to the nearest integer.
// For each column in the original matrix add the integer part leaving behind the decimal  fraction
FOR k = 1 to m
        N_INT(k) := 0; N_SUM(k) = 0;
        FOR p = 1 to J
                N_SUM(k) := N_SUM(k) + N_MATRIX(p,k);
                N_INT(k) := N_INT(k) +  INTEGER [N_MATRIX(p,k)];
                // Assign approximate number of servers of each energy levels to all groups
                A_MATRIX (p,k) := INTEGER[N_MATRIX(p,k)];
                // Subtract the number of assigned servers from total available servers for
                // each energy level.
                N_ARRAY(p) := N_ARRAY(p) – INTEGER[N_MATRIX(p,k)];
        END FOR;

        // round off to the nearest integer
        IF (N_SUM(k) - ⌊N_SUM(k)⌋) ≤ 0.49
                N_SUM(k) := ⌊N_SUM(k)⌋;
        ELSE
                N_SUM(k) := ⌈N_SUM(k)⌉ ;
        END IF;
        // subtract the integer part values from the round off values
        // This gives the number of servers required additionally in each group
        ADD_ARRAY(k) := N_SUM(k) – N_INT(k);
END FOR;

// Since the server groups are arranged in the increasing order of their read access ratios,
// start assigning servers to the server group with the highest read access ratio
p := 1;
FOR k = 1 to m
        // assign the required number of servers from the available servers
        //  servers with higher available energy levels are assigned to groups
        // having higher read access ratios
        WHILE ADD_ARRAY(k) > 0  AND p <= J
                IF N_ARRAY(p) > 0  AND  ADD_ARRAY(k) >= N_ARRAY(p) THEN
                        A_MATRIX(p,k) := A_MATRIX (p,k) + N_ARRAY(p);
                        ADD_ARRAY(k) := ADD_ARRAY(k) – N_ARRAY(p);
                        N_ARRAY (p) := 0;
                ELSE  IF N_ARRAY (p) > 0 AND ADD_ARRAY (k) < N_ARRAY (p) THEN
                        A_MATRIX (p,k) := A_MATRIX (p,k) + ADD_ARRAY (k);
                        N_ARRAY (p) := N_ARRAY (p) – ADD_ARRAY (k);
                        ADD_ARRAY (k) := 0;
                END IF;
                p++;
        END WHILE;
END FOR;
```

EXAMPLE:

An example showing the division of servers into groups is presented below.

Let n = 100, the number of servers. Let m = 3

Let the energy levels of 40 servers be 100, 30 servers be 60 and remaining 30 servers be 40.

Since there are three different energy levels, J = 3 such $E_{ARRAY}[1] = 100$, $E_{ARRAY}[2] = 60$, $E_{ARRAY}[3] = 40$.

Let the read accesses for each server group be $R_{ARRAY}[1] = 5$, $R_{ARRAY}[2] = 4$, $R_{ARRAY}[3] = 3$.

Asum := 3 + 4 + 5 = 12.

$A_{RATIO}[1] := 5/12$; $A_{RATIO}[2] := 1/3$; $A_{RATIO}[3] := 1/4$.

$N_{ARRAY}[1] = 40$; $N_{ARRAY}[2] = 30$; $N_{ARRAY}[3] = 30$.

The matrix N_{MATRIX} of size $J \times m$ is as follows:

	Group 1	Group = 2	Group = 3
$E_{ARRAY}[1] = 100$	16.67	13.33	10
$E_{ARRAY}[2] = 60$	12.5	10	7.5
$E_{ARRAY}[3] = 40$	12.5	10	7.5

$N_{SUM}[Group1] = \lceil 41.67 \rceil = 42$; $N_{SUM}[Group\ 2] = \lfloor 33.33 \rfloor = 33$; $N_{SUM}[Group\ 3] = 25$;

$N_{INT}[Group1] = 40$; $N_{INT}[Group2] = 33$; $N_{INT}[Group3] = 24$;

$ADD_{ARRAY}[Group\ 1] = 2$; $ADD_{ARRAY}[Group\ 2] = 0$; $ADD_{ARRAY}[Group\ 3] = 1$;

The matrix A_{MATRIX} is as follows after assigning N_{INT} to each groups

	Group 1	Group 2	Group =3
$E_{ARRAY}[1] = 100$	16	13	10

$E_{ARRAY}[2] = 60$	12	10	7
$E_{ARRAY}[3] = 40$	12	10	7

The new values of $N_{ARRAY}[1] = 1$; $N_{ARRAY}[2] = 1$; $N_{ARRAY}[3] = 1$;

Assign values $N_{ARRAY}[1]$ and $N_{ARRAY}[2]$ to Group 1 and assign value of $N_{ARRAY}[3]$ to Group 3.

The final A_{MATRIX} is as shown below:

	Group 1	Group = 2	Group = 3
$E_{ARRAY}[1] = 100$	17	13	10
$E_{ARRAY}[2] = 60$	13	10	7
$E_{ARRAY}[3] = 40$	12	10	8

Data Placement:

The next question that arises is to decide which data items go to which groups. For the infrequent write request, ideally it would be suitable to dynamically select servers for update. [Karumanchi, 1999] adopts this approach. However, if a MH has to read/update a data item, in the worst case scenario all the servers have to be searched to obtain the data item. This would result in very high communication costs. [Karumanchi, 1999] adopts the intersecting quorum based approach, but as explained in section 2, this approach eventually results in replicating the entire database at all the sites. This approach is not suitable from the point of view of storage space considerations. The GBRMAD technique assumes initial static allocation of data. Typically in a military environment, the army is divided into different divisions. Each division has some soldiers assigned to it. The soldiers belonging to each division can be assigned to a particular group. The

union of all the groups will result in the complete army. For each division the number of times the divisional data has to be accessed (read) can be estimated and the divisions can allocate the required number of servers with the calculations presented before. For this system the number of reads are considered to be much higher than the number of writes, as a result of which only the read accesses are considered. The number of divisions present represents the number of groups m. For updating the position of a soldier, the server determines the group to which the soldier belongs and as explained later updates the position of the soldier at majority of the servers in that group. Hence any update to the data item takes place only at the servers in the group to which the data item belongs. This makes the GBRMAD technique more static, but aims at achieving a balance between the communication costs and storage space requirements.

3.5 Primary and Secondary Groups:

A group can be classified as a primary group or a secondary group. The primary group of a data item X is the group where the primary copies of X reside. The primary copies are the pre-assigned copies of the data item X for which at least the majority of them should be updated for a successful update. Pre-assignment of the primary copies of X is explained in the earlier paragraph. Any update to the data item should result in at least an update of the majority of the primary copies. Since, there is a possibility that more than one server is allocated to a group, we have multiple primary copies of X.

The group other than the primary group is considered a secondary group with respect to the data item X. A server in the secondary group stores the copy of the data item X only if it is determined that there is a large number of reads originating from this server and hence it would be beneficial to store a local copy of X at the server in the secondary group. If a copy of the data

item X resides at any servers in the secondary group, the copy is referred to as the secondary copy of X.

Let S11, S12, S13 be servers belonging to group SG1 and S21 be a server belonging to group SG2. Xprim represents the primary copy of data item X and Xsec the secondary copy. The primary copies of X reside at servers S11, S12 and S13. To store a secondary copy of X at S21, S21 obtains the primary copies of X from at least two of the servers in group SG1. The most recently updated copy of two copies obtained is considered as Xsec. It is possible that S11 store Xprim, and S12 and S13 store Xprim', the most recently updated copy. If S21 obtains primary copies from S11 and S12, value Xprim' is stored as the value of Xsec at S21. The secondary copy may be different from some of the primary copies, however as explained later it is always the most recent copy. With respect to the data item X, S11, S12 and S13 are the primary servers of X and S21 the secondary server. The group SG1 would then be called the primary group with respect to X and SG2 the secondary group with respect to X. Figure 2 shows the idea of Primary and Secondary groups.

The primary groups hold the primary copies of a given data item and the servers in a secondary group maintain a temporary replica of this data item if the replication plan determines that it is beneficial to store a copy of X at this server in the secondary group. Every data item X has only one Primary group however can have more than one secondary group.

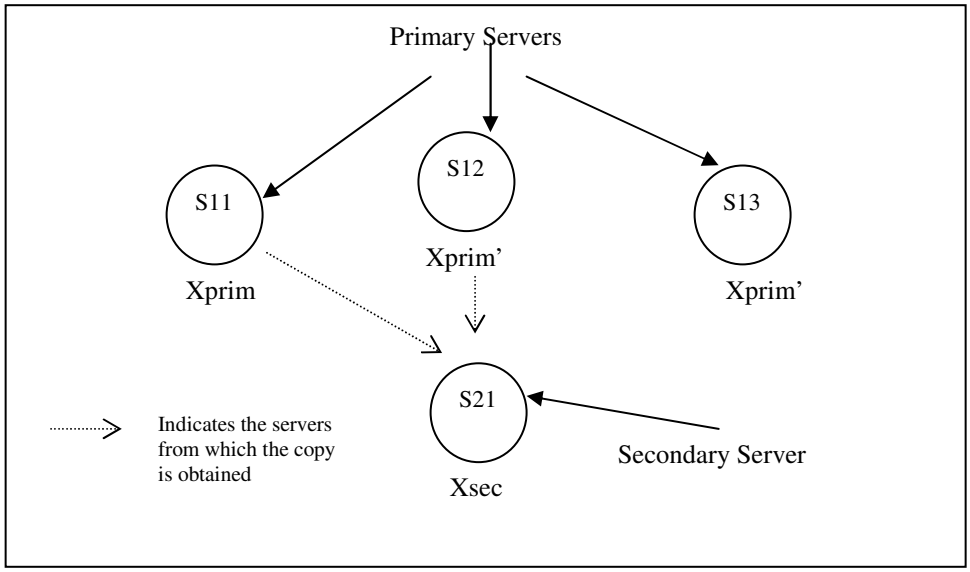

Figure 2: Primary and Secondary Servers

3.6 Replication Plan:

The replication plan uses a variation of the Majority Voting Strategy [Draffan 1980] for READS and WRITES. For the Majority Voting Strategy (MVS), once the majority of locks are obtained the update takes place. However, for this variation of the MVS, a transaction waits for all the servers to respond within a time out period and selects only the majority of servers for updates. The reason why the variation is adopted would be clear in the following paragraphs.

Let a transaction Tm that READS/UPDATES X originate at a server Sj. Let SGi be the Primary server group with respect to X. Let N_{SGi} be the number of servers in the group SGi. To obtain majority of locks for updates, any server Sj has to obtain at least $\lfloor (N_{SGi} / 2) + 1 \rfloor$ locks. For example if $N_{SGi} = 6$, majority of locks would be obtained if at least $\lfloor (6/ 2) + 1 \rfloor = 4$ locks are obtained. The variation of the MVS strategy waits for a certain timeout period to obtain locks on all the servers belonging to group SGi. If all the 6 locks are obtained the primary copy of the data item is updated at only four selected servers. The selection of servers is explained in the later

subsection. The variation of the MVS strategy does not update the remaining two primary copies. It is necessary that when a data item is being updated at SGi, no other transaction should be able to read the data item being updated. Hence, to read a data item it is necessary to obtain $\lceil NSGi / 2 \rceil$ locks. For NSGi = 6, the minimum number of read locks needed are $\lceil 6/2 \rceil = 3$. Since the read and write locks are incompatible, when a transaction Tm is updating a data item at SGi, no other transaction can read the same data item. Similarly, when a transaction Tm is reading a data item at SGi, no other transaction can update the data item.

Replica Allocation:

Each primary server maintains an update/write counter that keeps track of the number of updates on the data item X. This counter helps in finding how often the primary copy is updated. Each secondary server Sj maintains a read counter that keeps track of the number of reads that are originated from Sj for the data item X. This read counter helps in deciding whether it is beneficial to allocate a replica of the data item X at the secondary server.

The primary servers of X already hold a copy of the data item X. The replication plan aims at determining whether it is beneficial to hold a copy of the data item X at the secondary server. It is beneficial to allocate replica at the secondary server Sj, if the number of reads on X from Sj is greater than the number of writes. Let Xsec be the secondary copy of X stored at Sj. The server Sj maintains a list of all the primary servers from whom the copy of X is obtained. If any of the primary servers is not reachable from Sj, it is possible that some other transaction can obtain a majority locks on the primary servers without the knowledge of Sj and update X. Thus if Xsec stored at Sj is not deleted, Xsec would be an outdated value. Hence, Xsec is promptly deleted from Sj. This helps in maintaining an up-to-date copy of X at Sj. Each of the primary servers has

knowledge where the replica of X is stored. When an update takes place at the primary servers of X, if Sj is not reachable from the primary servers then primary servers are not able to delete Xsec. However, Sj finds that the primary servers are not reachable from it and deletes the Xsec stored at Sj. This two way procedure of deleting Xsec helps in maintaining an up-to-date copy of Xsec.

Update/Write Query:

For updating a data item, the idea is to use a variation of the MVS on the server group. A data item has to be updated at majority of the servers in its primary server group. When a transaction Tm that needs to update a data item X is originated at Sj, Tm requests an exclusive lock on all the servers in the group SGi where the primary copy of data item X resides. Tm then waits for a predetermined time $T_{timeout}$ to receive a response from all the servers in the primary group. If at least the majority of locks are obtained Tm continues; otherwise Tm is aborted.

Once the majority of locks are obtained, if Tm is a firm transaction, the nearest $\lfloor (NSGi / 2) + 1 \rfloor$ servers are updated. Here since not all the servers are updated it leads to inconsistency in data. However, as seen in the next section, for a read query we obtain $\lceil NSGi/2 \rceil$ locks thereby obtaining the most recently updated value of the data item X. Hence, updating all the servers would increase the overhead unnecessarily. [Karumanchi, 1999] also allows inconsistency in the data. Also in a mobile ad hoc network, it may be difficult to obtain locks on all the servers. As a result a transaction might get aborted too often. So for a firm transaction only the nearest majority of servers are updated. For a firm transaction, if the deadline of the transaction is missed

at any of the servers, the transaction is compensated at all the servers that participated in the update.

However, if Tm is a soft transaction, the servers in the primary group are arranged in the decreasing order of their energy levels. The $\lfloor (NSGi / 2) + 1 \rfloor$ servers having the best energy levels are then chosen for update. This helps in energy conservation at servers that have less energy level. Thus it is seen that by varying MVS a choice can be made to choose servers for updates depending on the type of transactions.

The update of X at the primary servers is followed by the deletion of all the secondary copies of X that existed at the servers in the secondary group. The secondary copies of X are the temporary replicas of X that are stored at the secondary servers as there is a cost savings in allocating a replica at the secondary servers. Suppose a secondary server Sk has a secondary copy of X. A transaction originating at some other server (primary/secondary) updates the data item X at the servers in the primary group. Transactions that originate at Sk and are scheduled after the update of X, may no longer be local to Sk due to the mobility of the server Sk. A transaction is called a local transaction if the MH that originated the transaction is within the direct radius of influence of the LMH. Once an update of X takes place at the primary servers, the secondary copies of X are deleted. Any further allocation of a secondary copy of X to a secondary server is based on the number of reads that presently exist local to that server. We can thus limit the number of secondary copies in the system by not maintaining secondary copies at the secondary servers. For an update query the transaction does not have to worry about whether it originated at the primary or secondary server. Every update is done on the primary servers. The transaction finds

the membership of the primary servers that hold the data item X. If the transaction originated at a primary server, one of the primary servers would be local to it. On the other hand if the transaction originated at a secondary server, all the primary servers would be remote to it. In any case the transaction has to obtain a majority of locks on the data item X at the primary servers for an update to take place.

The algorithm for an update query is given below.

Transaction Tm originated at server Sj updates a data item X

// Initialize a counter to keep track of the number of servers on which locks are obtained
Count := 0;
Identify and find the number of servers (N_{SGi}) belonging to the primary group of X
FOR each server in the primary group
    Request a lock on X at the primary server, distance from Sj to the server and energy level
        for each server in the primary group within the $T_{timeout}$ period;
    Count ++;
END FOR;
// Check if locks are obtained for X on the majority of the servers
IF Count >= $\lfloor (N_{SGi}/2) + 1 \rfloor$ THEN
    IF Tm is a firm transaction THEN
        Order servers in the increasing order of their distances from Sj;
        Update/write to the nearest $\lfloor (N_{SGi}/2) + 1 \rfloor$ primary servers;
    ELSE *// Tm is a soft transaction*
        Order servers in decreasing order on energy levels;
        Update/Write to the $\lfloor (N_{SGi}/2) + 1 \rfloor$ primaryservers having highest energy levels;
    END IF;
    COMMIT Tm;

ELSE *// if majority locks are not obtained.*
    ABORT Tm;
END IF;
// Check if the deadline is missed for a firm transaction
IF (Tm is committed) and (deadline missed for firm transaction Tm at any of the servers) THEN
    COMPENSATE Tm;
END IF;

Every primary server of X is aware of the location of all the secondary copies of X. As explained earlier, an update to the primary server of X is followed by the deletion of all the secondary copies of X associated with this primary server. If the secondary server is not reachable from the primary server, the primary server does not delete these copies. The secondary server when finds that the primary server is no longer reachable it deletes Xsec thereby avoiding inconsistencies.

The algorithm for deletion of secondary copies is presented below.

When a primary server Sk updates its data item X

```
WHILE there are secondary copies of X
        IF Secondary server storing X is reachable from Sk THEN
                Delete the secondary copy;
        END IF
END WHILE;
```

Read Query:

As explained earlier, for any update on data item X a minimum of $\lfloor (N_{SGi} /2) + 1 \rfloor$ primary servers of X are required. Hence, for any future read transaction to obtain the most recently updated value of the data item X, it is sufficient to read X from $\lceil N_{SGi} / 2 \rceil$ primary servers of group SGi. When a read transaction is in process, no majority copies can be obtained for updates on the same data item. Also in case of an ongoing write transaction, read locks cannot be obtained on $\lceil N_{SGi} / 2 \rceil$ primary servers for the same data item.

A transaction Tm originated at server Sj reads the data item X. If Sj belongs to the primary group of X and Tm is a transaction requiring the MRV of X, a lock request is send to all the servers in the primary group of X. The transaction then waits for a pre-specified time period $T_{timeout}$ for the primary servers to respond.

In the event that more than $\lceil N_{SGi} / 2 \rceil$ locks are obtained, Sj arranges all the primary servers in the increasing order of their distances from Sj. The data item X is then read from $\lceil N_{SGi} / 2 \rceil$ primary servers that are nearest to Sj. If exactly $\lceil N_{SGi} / 2 \rceil$ locks are obtained, data is immediately read from these servers. Reading data from $\lceil N_{SGi} / 2 \rceil$ ensures that the most recently updated value of the data item X is read. For a MRV transaction if less than $\lceil N_{SGi} / 2 \rceil$ locks are obtained the transaction is aborted.

A normal transaction does not necessarily require the most recently updated value of the data item X. Hence, for a normal transaction, a lock is requested from a single primary server that is closest to Si. The data item X is then read from this primary server.

A firm/soft transaction can either be a MRV or a normal transaction. If a firm transaction Tm misses its deadline, Tm is aborted. For soft transaction Tm, Tm continues to execute even if the deadline is missed.

A transaction Tp originated at a secondary server Sk with respect to the data item X checks for the presence of the secondary copy of X at Sk. If no copy of exists at Sk and it is beneficial to allocate a replica of X at Sk, a replica is allocated at the server Sk. The replica is obtained from the primary copies of X at the primary servers on which locks are obtained. The most recent value of all the primary copies is taken as the value of the replica. Tp then reads the data item X locally. If a copy exists at Sk, Tp reads the data item X locally from Sk and commits. As explained earlier, since the secondary copy of X is always an up-to-date copy, Tp always reads the most recently updated value of X. Thus, if a replica exists at Sk, Tp reads the data item X locally thereby reducing lot of overhead.

The algorithm for a read query is given below.

Transaction Tm Originating at server Sj, Reads the data item X

```
// Initialize a counter to keep track of the number of servers on which locks are obtained
Count := 0;
// Initialize a Boolean value to false. Only if this Boolean is set to true a transaction is
// committed
SUCCESS := FALSE;
Identify and count the number of servers (N_SGi) belonging to the primary group of X
// The following IF condition is executed if the server at which Tm originates is the primary
// group of X  and if Tm requires the most recently updated value.
IF (Sj belongs to the primary group of X) AND (Tm is MRV) THEN
        // Here the transactions request locks and relevant information from all the primary
        // servers holding the data item X
        FOR each server in the primary group
                Request a lock on X at the primary server, distance from Sj to primary server and
                        energy level for each server in the primary group within the T_timeout period;
                Count ++;
        END FOR;
       // Check if locks are obtained on minimum number of servers required to read X. If locks
       // are  obtained continue.
        IF Count >= ⌈ N_SGi / 2 ⌉ THEN
                // The servers are ordered in the increasing order of their distances from Sj for a
                // firm transaction and in the decreasing order of their energy levels for soft
                // transaction. The value of X is read from the first ⌈N_SGi / 2 ⌉servers
                IF Tm is a firm transaction THEN
                        Order servers in the increasing order of their distances from Sj;
                ELSE IF Tm is a soft transaction THEN
                        Order servers in decreasing order of their energy levels;
                END IF;
                Read X from the first ⌈ N_SGi / 2 ⌉ ordered servers;
                SUCCESS:= TRUE;
        END IF
// This condition is executed if the server at which Tm originates is the primary group of X
// and if Tm does not require the most recently update value.
ELSE IF (Sj belongs to the primary group of X) AND (Tm is a normal transaction) THEN
        IF Tm is a firm transaction THEN
                Request lock on nearest primary server of X;
        ELSE IF Tm is a soft transaction THEN
                Request lock on the primary server of X having highest energy level;
        END IF;
        //Check if the required lock is obtained. If so set success to true.
        IF required lock obtained
                Read X;
                SUCCESS:= TRUE;
        END IF;
```

```
// This condition is executed if the server at which Tm originates belongs to the secondary group
// of X and if a replica of X already exists at the secondary server.
ELSE IF (Sj belongs to the secondary group with respect to X) AND (replica of X, i.e., Xsec is
        present at Sj) THEN
        Request lock for Xsec on Sj;
        //Check if the required lock is obtained. If so set success to true.
        IF required lock obtained
                Read Xsec;
                SUCCESS:=TRUE;
        END IF;

// This condition is executed if the server at which Tm originates belongs to the secondary group
// of  X  and if there is no replica of X at the secondary server.
ELSE IF (Sj belongs to the secondary group with respect to X) AND (replica of X, i.e., Xsec is
        absent at Sj) THEN
        // Check if the Tm is MRV transaction
        IF (Tm is MRV) THEN
                FOR each server in the primary group
                        Request a lock on X at the primary server, distance from Sj to the primary
                        server and energy level for each server in the primary group within the
                        T_timeout period;
                        Count ++;
                END FOR;
                // Check if locks are obtained on the minimum number of servers required to
                // perform read on data item X. If not abort the transaction
                IF Count >= ⌈N_SGi / 2⌉ THEN
                        // The servers are ordered in the increasing order of their distances from
                        // Sj for a firm transaction and in the decreasing order of their energy
                        // levels for soft  transaction. The value of X is read from the first ⌈N_SGi / 2⌉
                        //servers
                        IF (Tm is a firm transaction) THEN
                                Order servers in the increasing order of their distances from Sj;
                        ELSE IF (Tm is soft transaction) THEN
                                Order servers in the decreasing order of their energy levels;
                        END IF;
                        //Here the copy is allocated if the number of reads is greater than the
                        // writes
                        IF (number of reads from Sj > number of writes) THEN
                                Obtain Xprim from the first ⌈N_SGi / 2⌉ ordered servers;
                                Allocate Xprim with the highest time stamp as Xsec at Sj;
                                Read Xsec;
                                SUCCESS:= TRUE;
                        ELSE
                                Read X from the first ⌈N_SGi / 2⌉ ordered servers;
                                SUCCESS := TRUE
                        END IF;
```

```
                END IF

        // If Tm is not a MRV transaction then execute the following section
        ELSE
                IF Tm is a firm transaction THEN
                        Request lock on nearest primary server;
                ELSE IF Tm is a soft transaction THEN
                        Request lock on the primary server having highest energy level;
                END IF;
                IF required lock obtained
                        Read X;
                        SUCCESS:=TRUE;
                END IF;
        END IF;
END IF;
// If success is true, the transaction can be committed else it is aborted
IF (SUCCESS) THEN
        COMMIT Tm;
ELSE
        ABORT Tm;
END IF;
// If a deadline of the committed firm transaction is missed compensate the transaction
IF (Tm is committed) and (deadline is missed for for firm transaction Tm) THEN
        COMPENSATE Tm;
END IF;
```

4 Theoretical Analysis for the GBRMAD Technique:

Let

$T_L \rightarrow$ Average lookup time for a data item at the local server. It is also assumed as the time required to obtain lock and read the data.

$T_R \rightarrow$ Average communication time required in sending the request/data and receiving acknowledgement from a remote server for a data item.

$T_C \rightarrow$ Time required in checking at the transaction-issuing server if majority locks are obtained

$T_{group} \rightarrow$ Time required to find the group membership

$T_O \rightarrow$ Time required to order the servers on whom locks are obtained either according to their available energy levels or their distance from server Sj

$T_{CREATE} \rightarrow$ Time required to create a secondary copy at server Sj

$P_L \rightarrow$ Probability that the copy of a data item exists at the secondary site

$P_R \rightarrow$ Probability that a given server is reachable from the server where the transaction originates.

$P_{MR} \rightarrow$ Probability that majority servers in a group are reachable from the server where the transaction originates

$P_{NLR} \rightarrow$ Probability that majority servers in a group are reachable but required number of locks not obtained for a read transaction

$P_{NLW} \rightarrow$ Probability that majority servers in a group are reachable but required number of locks not obtained for a read transaction. P_{NLW} is different from P_{NLR} since for the system the number of reads is considered to be more than the number of writes. Also Read-Read locks are compatible while Write-Write locks are not compatible.

$P_{FIRM} \rightarrow$ Probability that the transaction is firm

$P_{DEADLINE} \rightarrow$ Probability that a deadline is missed at a server by the firm transaction

SGi $\rightarrow$ The primary group of data item X

$N_{SGi} \rightarrow$ The number of servers in group SGi

Let Pr be the probability that a given signal sent out to locate a particular server, finds that server. Pr is assumed to have the same value for every server in the system. According to the model assumed to describe the server system and its interactions, only if majority of total number of servers are reachable then it is possible to obtain a lock. The probability that a majority of the servers will be reachable is determined as follows:
Given a total of "n" servers, probability of successfully locating a majority (n/2 + 1) of servers is unity minus the sum of probabilities of finding exactly 1 to n/2 servers,

A transaction can proceed only if the majority of servers in the primary group are reachable from the server that originated the transaction. The probability that a majority of the servers are reachable can be determined in terms of P_R as follows:

If there are N_{SGi} servers in a group SGi, probability of successfully locating a majority of servers ($[N_{SGi} + 1] / 2$) is unity minus the sum of the probabilities of finding exactly 1 to $[N_{SGi} / 2]$ servers. As an example the probability of finding exactly 2 servers is given by

$$P_2 = \sum_{k=1}^{2} P_R^2 \ (1\text{-}P_R)^{N_{SGi}-2} \ {}^{N_{SGi}}C_2$$

Hence, the probability that the majority of the servers are reachable is given by

$$P_{MR} = 1 - \sum_{k=1}^{N_{SGi}/2} P_R^k \ (1\text{-}P_R)^{N_{SGi}-k} \ {}^{N_{SGi}}C_k$$

The theoretical analysis includes calculating the execution time of the transaction and the energy consumption at the server that issues the transaction.

4.1 Execution Time for the GBRMAD Technique:

Case 1:

When a transaction Tm wants to reads a data item X from a secondary site (Sj):

The total execution time is given by the summation of the time required to do the following functions.

a) Time to find out to which group Sj belongs (Tgroup)

b) Time to check if a copy of X resides at Sj and to obtain lock on X and read X ($2T_L$)

c) If local copy does not exist, time required between requesting a lock and granting a lock. This time also includes the time required to search the database of the remote primary servers for the existence of copy of the data item X ($T_R + T_L$)

d) Time to check if $\lceil N_{SGi} / 2 \rceil$ locks are obtained for read (T_C)

e) If $\lceil N_{SGi} / 2 \rceil$ locks are obtained then the time required to order the servers either according to energy levels (for soft transactions) or distance of the servers from Sj (T_O)

f) Time to read data (T_R+T_L). This is the communication time required for sending a request for read and obtaining the data item.

g) If it is beneficial to obtain a copy at Sj, then the time required to create a local copy (T_{CREATE})

h) If transaction is firm, time required to check if the deadline is missed (T_C). For simplicity this time is assumed equal to the time required to check if the necessary number of locks are obtained.

i) If the deadline is missed then the time required to compensate the transaction ($T_R + T_L$)

Hence the total execution time is given by:

Execution Time = **(**Time to find out to which group Sj belongs to**) +**

(Probability that local copy exists * [Time for local look up + Time for obtaining locks and reading the data]**) +**

((Probability that local copy does not exist) * [Time to find the primary group of X + Time to obtain locks on X at the primary sites + Time to check if $\lceil N_{SGi} / 2 \rceil$ locks are obtained + (Probability that the majority servers are reachable)*(Probability of obtaining locks on $\lceil N_{SGi} / 2 \rceil$ servers * {(Probability that transaction is MRV) *Time to order servers + Time to read data + Time to create copy}]**) +**

Probability that the transaction is firm * Probability that the deadline is missed * [T_C +T_R + T_L]

$$\text{Execution Time} = [T_{group}] + [P_L(T_L + T_L)] + [(1-P_L)[T_{group} + (T_R + T_L) + T_C + P_{MR} * (1-P_{NLR}) * (P_{MRV} * T_O + T_R + T_L + T_{CREATE})]] + [P_{FIRM} * P_{DEADLINE} * (T_C + T_R + T_L)]$$

→ (4.1)

In the above equation P_{MR} can be expressed in terms of P_R as explained before.

Case 2:

When a transaction Tm wants to reads a data item X from a Primary site (Sj):

The total execution time is given by the summation of the time required to do the following functions.

a) Time to find out to which group Sj belongs (Tgroup)

b) Time required between requesting a lock and granting a lock. This time also includes the time required to search the database of the remote primary servers for the existence of copy of the data item X ($T_R + T_L$)

c) Time to check if $\lceil N_{SGi} / 2 \rceil$ locks are obtained for read (T_C)

d) If $\lceil N_{SGi} / 2 \rceil$ locks are obtained then the time required to order the servers either according to energy levels (for soft transactions) or distance of the servers from Sj (for firm transactions) (T_O)

e) If locks are obtained, time to read data ($T_R + T_L$). This is the communication time required for sending a request for read and obtaining the data item.

f) If transaction is firm, time required to check if the deadline is missed (T_C). For simplicity this time is assumed equal to the time required to check if the necessary number of locks are obtained.

g) If the deadline is missed then the time required to compensate the transaction ($T_R + T_L$)

Hence the total execution time is given by:

Execution Time = **(**Time to find out to which group Sj belongs to**) + (**Time to obtain locks) +

(Time to check for majority locks**)** +

((Probability that the majority servers are reachable)*(Probability of obtaining locks on $\lceil N_{SGi} / 2 \rceil$ servers * {(Probability that transaction is MRV) *Time to order servers + Time to read data}]**)** +

(Probability that the transaction is firm * Probability that the deadline is missed * [Time to check if deadline is missed + Time to compensate transaction]**)**

$$\text{Execution Time} = T_{group} + (T_R + T_L) + T_C + [P_{MR} * (1-P_{NLR}) * (P_{MRV} * T_O + T_R + T_L)]] + [P_{FIRM} * P_{DEADLINE} * (T_C + T_R + T_L)] \rightarrow \textbf{(4.2)}$$

Case 3:

When Tm wants to write/update a data item X from a server Sj:

The total execution time is given by the summation of the time required to do the following functions.

a) Time required to identify the primary group of X

b) Time required between requesting a lock and granting a lock. This time also includes the time required to search the database of the remote primary servers for locating the copy of the data item X ($T_R + T_L$)

c) Time to check if majority locks are obtained (T_C)

d) If $\lceil N_{SGi} / 2 \rceil$ locks are obtained then the time required to order the servers either according to energy levels (for soft transactions) or distance of the servers from Sj (for firm transactions) (T_O)

e) If majority locks are obtained, time to update data ($T_R + T_L$). This is the communication time required for sending a request for update and update the data item.

f) If transaction is firm, time required to check if the deadline is missed (T_C). For simplicity this time is assumed equal to the time required to check if the necessary number of locks are obtained.

g) If the deadline is missed then the time required to compensate the transaction ($T_R + T_L$)

Hence the total execution time is given by:

Execution Time = (Time required to identify the primary group of X) + (Time to obtain locks) +

(Time to check for majority) +

((Probability that majority servers are reachable) * (Probability of obtaining majority locks) * Time to update data) +

(Probability that the transaction is firm * Probability that the deadline is missed * [Time to check if deadline is missed + Time to compensate transaction])

$$\text{Execution Time} = T_{group} + (T_R + T_L) + T_C + [P_{MR} * (1-P_{NLW}) * (P_{MRV} * T_O + T_R + T_L)]] + [P_{FIRM} * P_{DEADLINE} * (T_C + T_R + T_L)] \rightarrow \textbf{(4.3)}$$

4.2 Division of Access Time for Computing Energy Consumption:

For every transaction Tm, it is necessary to determine the energy consumption at the server where Tm is originated. The following energy consumption per second is assumed for the active and doze mode [Imielinski, 1994].

1) Active mode energy consumption per second: 250 mW = 0.25 W
2) Doze mode energy consumption per second: 0.05 mW = 0.00005 W

The execution time for each transaction Tm is calculated in the earlier section. Product of the energy consumption per second in Watts and the execution time will give the total energy consumption per transaction at each server that originated the transaction. Every server has to remain in the active state while receiving and transferring data. The server can go into doze mode while it waits for the remote server to respond for the lock request or waits while a data item is updated/read from the remote server. The server that originated the transaction Tm utilizes a fraction of the time T_R in sending and receiving the data. Let $(\alpha \times T_R)$ be the fraction of the time required to receive and send data by any server, where α is a constant multiplier that indicates the percentage of time (T_R) required for transferring and receiving information. During this time the server that originated the transaction has to be in active mode.

$(1-\alpha)\times T_R$ represents the fraction of the time spent in waiting for the remote server to respond. During this time the server can go into doze mode to conserve energy. The aforementioned three cases shall be considered for calculating the energy consumption at the transaction originating sites. In each of the cases the amount of time the server has to be in active mode and the amount of time the server has to be in doze mode are calculated. These time values are then multiplied by the assumed energy consumption per second to obtain the energy consumed at each server where the transaction Tm originated.

Case 1:

When Ti wants to reads a data item X from a secondary site (Sj),

From equation (4.1)

$$\text{Execution Time} = [T_{group}] + [P_L(T_L + T_L)] + [(1-P_L)[T_{group} + (T_R + T_L) + T_C + P_{MR} * (1-P_{NLR}) * (P_{MRV} * T_O + T_R + T_L + T_{CREATE})]] + [P_{FIRM} * P_{DEADLINE} * (T_C + T_R + T_L)]$$

Since $(\alpha \times T_R)$ is the time during which the server has to be in active mode for receiving and transferring data, the total T_R is broken into $(\alpha \times T_R)$ and $(1-\alpha)\times T_R$ as shown in the following equation:

$$\text{Execution Time} = T_{group} + 2P_LT_L + (1-P_L)[T_{group} + (\alpha T_R + (1-\alpha) T_R + T_L) + T_C + P_{MR} * (1-P_{NLR}) * (P_{MRV} * T_O + \alpha T_R + (1-\alpha) T_R + T_L + T_{CREATE})] + [P_{FIRM} * P_{DEADLINE} * (T_C + \alpha T_R + (1-\alpha) T_R + T_L)] \rightarrow \mathbf{(4.4)}$$

Equation (4.7) can be rearranged in two parts. One part shows the time spent by Sj (where the transaction originated) in performing local lookups, finding groups, checking majority and receiving and transferring data. The second part shows the time spent by Sj in waiting for remote servers to respond.

Rearranging the terms, the access time can be written as

$$\text{Execution Time} = \underbrace{\left[T_{group} + 2P_L T_L + (1-P_L)(T_{group} + \alpha T_R + T_C + P_{MR} * (1-P_{NLR}) * (P_{MRV} * T_O + \alpha T_R + T_{CREATE}\right] + \left[P_{FIRM} * P_{DEADLINE} * (T_C + \alpha T_R)\right]}_{\text{part 1}}$$

$$\underbrace{\left[(1-P_L)\ \{(1-\alpha)T_R + T_L + (P_{MR} * (1-P_{NLR}) * (1-\alpha)T_R)\}\right] + \left[P_{FIRM} * P_{DEADLINE} * ((1-\alpha)T_R + T_L)\right]}_{\text{part 2}}$$

→ (4.5)

$$= A_{Tl} + A_{Tr}$$

Where $A_{Tl} = \left[T_{group} + 2P_L T_L + (1-P_L)(T_{group} + \alpha T_R + T_C + P_{MR} * (1-P_{NLR}) * (P_{MRV} * T_O + \alpha T_R + T_{CREATE}\right] + \left[P_{FIRM} * P_{DEADLINE} * (T_C + \alpha T_R)\right]$, represents the time spent by Tm in looking up its local database, the time spent in checking if the majority locks are obtained and time spent in receiving and transferring information.

And $A_{Tr} = \left[(1-P_L)\ \{(1-\alpha)T_R + T_L + (P_{MR} * (1-P_{NLR}) * (1-\alpha)T_R)\}\right] + \left[P_{FIRM} * P_{DEADLINE} * ((1-\alpha)T_R + T_L)\right]$, represents the time spent by Tm in waiting for the remote sites to respond.

Case 2:

When Tm wants to reads a data item X from a Primary site (Sj),

As shown in case 1, total execution time can be rewritten as:

Execution Time = $[Tgroup + \alpha T_R + T_C + P_{MR}* (1-P_{NLW})* (P_{MRV} * T_O + \alpha T_R)] + [P_{FIRM} * P_{DEADLINE} * (T_C + \alpha T_R)]$

part 1

$[T_L + (1-\alpha)T_R + P_{MR}* (1-P_{NLW}) \{(1-\alpha)T_R + T_L] + [P_{FIRM} * P_{DEADLINE} * ((1-\alpha)T_R + T_L)]$ → **(4.6)**

part 2

Part 1 represents the time during which the server Sj should be active mode and part 2 indicates the time during which the server can go into doze mode.

Hence here $A_{Tl} = [Tgroup + \alpha T_R + T_C + P_{MR}* (1-P_{NLW})* (P_{MRV} * T_O + \alpha T_R)] + [P_{FIRM} * P_{DEADLINE} * (T_C + \alpha T_R)]$

And $A_{Tr} = [T_L + (1-\alpha)T_R + P_{MR}* (1-P_{NLW}) \{(1-\alpha)T_R + T_L] + [P_{FIRM} * P_{DEADLINE} * ((1-\alpha)T_R + T_L)]$

Case 3:

When Tm wants to write/update a data item X from a server Sj

As shown in case 1, total execution time can be rewritten as:

Execution Time = $[Tgroup + \alpha T_R + T_C + P_{MR}* (1-P_{NLW})* (P_{MRV} * T_O + \alpha T_R)] + [P_{FIRM} * P_{DEADLINE} * (T_C + \alpha T_R)]$

part 1

$$\underbrace{[T_L + (1-\alpha)T_R + P_{MR}* (1-P_{NLW}) \{(1-\alpha)T_R + T_L] + [P_{FIRM} * P_{DEADLINE} * ((1-\alpha)T_R + T_L)]}_{\text{part 2}} \rightarrow \textbf{(4.7)}$$

Part 1 represents the time during which the server Sj should be active mode and part 2 indicates the time during which the server can go into doze mode.

Hence here $A_{Tl} = [Tgroup + \alpha T_R + T_C + P_{MR}* (1-P_{NLW})* (P_{MRV} * T_O + \alpha T_R)] + [P_{FIRM} * P_{DEADLINE} * (T_C + \alpha T_R)]$

And $A_{Tr} = [T_L + (1-\alpha)T_R + P_{MR}* (1-P_{NLW}) \{(1-\alpha)T_R + T_L] + [P_{FIRM} * P_{DEADLINE} * ((1-\alpha)T_R + T_L)]$

4.3 Total Energy Consumption for ηNumber of Transactions at a Server

Naturally the time required for performing local operations (A_{Tl}) is much smaller than the time required for remote operations (A_{Tr}). The server has to remain in active mode during the time A_{Tl} and can go into doze mode during the time A_{Tr}. A server can go into doze mode only if no other transactions are being processed/originated at that server. To determine the total energy consumption for η transactions, it is necessary to consider the time lag between any two consecutive transactions. The time lag between any two consecutive transactions can decide the total energy consumption for η number of transactions. More the time lag between any two consecutive transactions, more is the possibility that a server can go into doze mode. For

simplicity of calculations, the time lag is considered to be constant between any two consecutive transactions.

Let Δ be the time lag between any two consecutive transactions. Three cases arise:

a) $\Delta \leq A_{Tl}$, b) $A_{Tl} < \Delta \leq A_{Tl} + A_{Tr}$, and c) $\Delta > A_{Tl} + A_{Tr}$

CASE a) $\Delta \leq A_{Tl}$

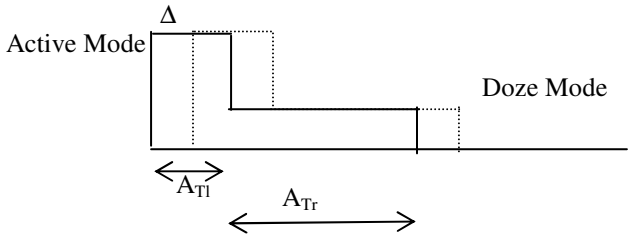

Figure 3: Time lag for case a)

Figure 3 shows the active and the doze mode of a server when two transactions are originated at the server for case a. For two transactions, the server has to remain in active mode for the time (2* A_{Tl}) and go into doze mode when the second transaction is being processed at the remote servers. Thus, if there are η transactions, then server has to remain in active mode for (η* A_{Tl}) time and go into doze mode when the transaction is processed at the remote server. Hence the total energy consumption for case a) is given by

$$E_T = 0.25(\eta * A_{Tl}) + 0.0005\ A_{Tr} \quad \rightarrow \textbf{(4.8)}$$

CASE b) $A_{Tl} < \Delta \leq A_{Tl} + A_{Tr}$

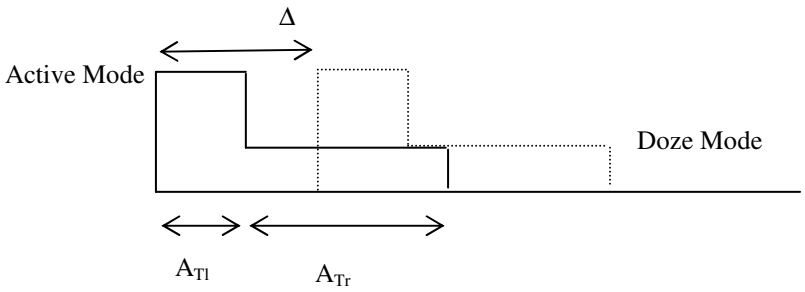

Figure 4: Time lag for case b)

Figure 4 shows the active and the doze mode of a server when two transactions are originated at the server for case b. For two transactions, the server has to remain in active mode for the time (2* A_{Tl}) and go into doze mode during 1) the time lag between first and second transaction and 2) when the second transaction is being processed at the remote servers. Thus, if there are η transactions, then server has to remain in active mode for (η* A_{Tl}) time and go into doze mode during the time [(η-1)* (Δ - A_{Tl}) + A_{Tr}]. Hence the total energy consumption for case a) is given by

$$E_T = 0.25(\eta * A_{Tl}) + 0.0005(\eta - 1) * (\Delta - A_{Tl}) + 0.0005\, A_{Tr} \quad \rightarrow \textbf{(4.9)}$$

CASE b) $\Delta > A_{Tl} + A_{Tr}$

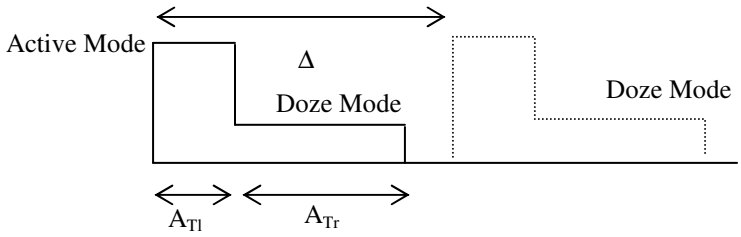

Figure 5: Time lag for case c)

Figure 5 shows the active and the doze mode of a server when two transactions are originated at the server for case c. For two transactions, the server has to remain in active mode for the time (2* A_{Tl}) and go into doze mode during 1) the time lag between first and second transaction after the first transaction has finished processing locally, and 2) when the second transaction is being processed at the remote servers. Thus, if there are η transactions, then server has to remain in active mode for (η* A_{Tl}) time and go into doze mode during the time [(η-1)* (Δ - A_{Tl} - A_{Tr}) + ηA_{Tr}]. Hence the total energy consumption for case a) is given by

$$E_T = 0.25(\eta^* A_{Tl}) + 0.0005(\eta-1)^* (\Delta - A_{Tl} - A_{Tr}) + 0.0005\, \eta A_{Tr} \quad \rightarrow \textbf{(4.10)}$$

5 Theoretical Analysis for the technique by [Karumanchi, 1999]:

[Karumanchi, 1999] has proposed three techniques, namely Select_then_Eliminate (STE) strategy, Eliminate_then_Select (ETS) strategy and Hybrid strategy. This section shows the theoretical analysis for these three techniques.

Let,

$T_L \rightarrow$ Average lookup time for a data item at the local server.

$T_R \rightarrow$ Average time required in sending the request/data and receiving acknowledgement

$T_{SQ} \rightarrow$ Time required in selecting a quorum

$T_{DQL} \rightarrow$ Time required to check the DQLx list

$T_{EQ} \rightarrow$ Time required in eliminating quorums

$T_{ADD} \rightarrow$ Time required in adding servers to the DQLx list

$P_R \rightarrow$ Probability that a given server is reachable from the server that originates the transaction

$P_Q \rightarrow$ Probability that a quorum gets selected

$P_S \rightarrow$ Probability that at least one server in the selected quorum is not in the DQLx list

Ns $\rightarrow$ number of servers in the selected quorum

The theoretical analysis includes calculating the execution time required by a transaction and the energy consumption at the server that issues the transaction.

5.1 Execution Time:

STE strategy:

The total execution time is given by the summation of the time required to do the following functions.

a) Time to randomly select a quorum Qi for read/update (T_{SQ})

b) Time to check the DQLx list for eliminating servers to whom read/update is sent (T_{DQL})

c) Time required in sending read/update message to the selected servers in the Qi. This time includes the communication time and the time to search the local database of all the servers in the Qi ($T_R + T_L$)

d) Time to add servers that did not respond within the timeout period to the DQLx list (T_{ADD})

e) If transaction is firm, time required to check if the deadline is missed (T_C). For simplicity this time is assumed equal to the time required to check if the necessary number of locks are obtained.

f) If the deadline is missed then the time required to compensate the transaction ($T_R + T_L$)

Hence the total execution time is given by:

Execution Time = **(**Time to randomly select a quorum Qi for read/update**) +**

(Time to check the DQLx list**) +**

((Probability at least one server in the selected quorum is not in the DQLx list) *

{Time required in sending read/update message to servers in Qi **+**

(Probability that at least one server does not respond) * (Time to add servers to DQLx list)}) +

(Probability that the transaction is firm * Probability that the deadline is missed * [Time to check if deadline is missed + Time to compensate transaction])

$$\text{Execution Time} = T_{SQ} + T_{DQL} + P_S * [(T_R + T_L) + [1 - (1- P_R)^n]T_{ADD} + [P_{FIRM} * P_{DEADLINE} * (T_C + T_R + T_L)] \rightarrow \textbf{(5.1)}$$

ETS strategy:

The total access time is given by the summation of the time required to do the following functions.

a) Time to check the DQLx list for eliminating quorums (T_{DQL})

b) If any quorums are left, time to randomly select a quorum Qi for read/update from the remaining quorums (T_{SQ})

c) Time required in sending read/update message to the servers in the Qi. This time includes the communication time and the time to search the local database of all the servers in the Qi ($T_R + T_L$)

d) Time to add servers that did not respond within the timeout period to the DQLx list (T_{ADD})

e) If transaction is firm, time required to check if the deadline is missed (T_C). For simplicity this time is assumed equal to the time required to check if the necessary number of locks are obtained.

f) If the deadline is missed then the time required to compensate the transaction ($T_R + T_L$)

Hence the total execution time is given by:

Execution Time = (Time to check the DQLx list) +

((Probability that at least one quorum exists) * [Time to randomly select a quorum Qi for read/update + Time required in sending read/update message to servers in Qi + (Probability that at least one server does not respond) * (Time to add servers to DQLx list)]) +

(Probability that the transaction is firm * Probability that the deadline is missed * [Time to check if deadline is missed + Time to compensate transaction])

$$\text{Execution Time} = T_{DQL} + P_Q * [T_{SQ} + (T_R + T_L) + [1 - (1 - P_R)^n] T_{ADD}] + [P_{FIRM} * P_{DEADLINE} * (T_C + T_R + T_L)] \rightarrow (5.2)$$

Hybrid Strategy:

The hybrid strategy uses the STE strategy for reads and ETS strategy for updates. Hence the access time for reads and updates is given by equation (5.1) and (5.2) respectively.

5.2 Energy Consumption:

The technique proposed by [Karumanchi, 1999] does not discuss the concept of active mode and doze mode. Hence, the servers are always assumed in active mode. The active mode energy consumption per second: 250 mW = 0.25 W [Imelinski, 1994] is assumed.

The product of energy consumption per second and the execution time of a transaction Tm gives the total energy consumed by a node x at which Tm originated.

For the STE strategy, the total energy consumed at node x per transaction is given by:

$$E_T = 0.25\{T_{SQ} + T_{DQL} + P_S * [(T_R + T_L) + (1 - P_R)^n T_{ADD} + [P_{FIRM} * P_{DEADLINE} * (T_C + T_R + T_L)]\}$$

→ (5.3)

For the ETS strategy, the total energy consumed at node x per transaction is given by:

$$E_T = 0.25\{T_{DQL} + P_Q * [T_{SQ} + (T_R + T_L) + (1- P_R)^n T_{ADD}] + [P_{FIRM} * P_{DEADLINE} * (T_C + T_R + T_L)]\}$$

→ (5.4)

For the Hybrid strategy, since the STE strategy is used for reads, equation (5.3) gives the total energy consumption for a read transaction. Equation (5.4) represents the total energy consumption for an update transaction as the ETS strategy is used for updates. The next section compares the GBRMAD technique and the technique by [Karumanchi, 1999].

6 Comparison of Techniques:

This section compares the GBRMAD technique with the earlier developed techniques. Since the active replication scheme, the per user replication scheme, and the data replication scheme between the mobile computer and the stationary computer are not developed for ad hoc mobile environment, this section compares only the GBRMAD technique and the technique developed by [Karumanchi, 1999].

The data replication technique proposed by [Karumanchi, 1999] does not aim at maintaining the consistency of the replicas. All the replicas can have different values and there is no way of ensuring that the data item read is the most recently updated. The GBRMAD technique does not ensure consistency of data at all the primary servers either. However, this technique aims at obtaining the most recently updated value of a data item. In the GBRMAD technique the transactions are classified as MRV transactions (transactions that require the most recently updated value) and normal transaction (transactions that do not necessarily require the most recently updated value). For read transactions requiring the most recent value, the transactions either read the data item from the up-to-date secondary copy of the data item or obtain the majority locks on the primary copies. This ensures reading the most recent value of the data item.

For normal transactions, the transactions either read the data item from the nearest server or the server with the highest energy level depending whether the transaction is firm or soft respectively.

The existing technique does not take into consideration the available energy levels of the server for maintaining replica. However, the GBRMAD technique takes into consideration the available energy levels for soft transactions. This ensures that the updates/reads are done on servers having the highest energy levels in case of soft transaction. However, for firm transactions the nearest servers are accessed irrespective of their energy levels.

The GBRMAD technique uses the variation of the majority voting strategy for updates. This ensures maintaining consistency of replicas in at least majority of the primary servers. An update succeeds only if the majority locks are obtained. However, in the existing technique a transaction succeeds even if a single server responds.

For the theoretical analysis carried out in section 4 and 5, the execution time of each transaction and the total energy consumption at the server originating transactions are compared. The probability that a primary server is reachable from the server originating a transaction is the only common parameter for the execution time between GBRMAD technique and techniques proposed by [Karumanchi, 1999]. For the purpose of analysis the number of servers in each server group are considered to be constant. The number of servers in a quorum for [Karumanchi, 1999] is assumed to be equal to the number of servers in the server group for the GBRMAD technique. This was to study the effect of increasing the number of servers on the execution time and energy consumption. Table 1 shows the assumed value of all parameters introduced in sections 4 and 5.

Parameters	Values
T_L	0.003 seconds [Dirckze, 1999]
T_R	0.07 seconds [Dirckze, 1999]
T_C , T_{group}, $T_{SQ,}$ T_{DQL}, T_{CREATE}	0.003 seconds
P_L	0.5
P_{NLR}	0.3
P_{NLW}	0.5
P_S, P_Q, P_{FIRM}, P_{MRV}	0.5
α	0.1
$P_{DEADLINE}$	0.3

Table 1: AssumedValues of Parameters

Figures 6 and 7 show the execution time for a read transaction originated at a secondary site, a read transaction originated at the primary site, an update/write request (all for the GBRMAD technique), STE strategy and ETS strategy. Since the Hybrid strategy gives the same results as the STE strategy for read and the ETS strategy for update, this strategy is not considered for analysis. Figures 6 and 7 show that the execution time for a transaction is less for the STE and ETS strategy as compared to the GBRMAD technique. This increase in the execution cost is mainly due to the locking technique used. However, for read transactions originating at a secondary server, the execution time is close to the execution time for the STE and ETS strategy. The presence of up-to-date replicas at the secondary server reduces the execution cost of the transaction originating at that server. As the P_R value increases, the execution time of STE and ETS strategy decreases. Increasing the P_R value increases the probability that the remote server is within the reach of the server where the transaction originated. In the STE and ETS strategy, when a server does not respond, the server is added to the DQL list. Equations 5.1 and 5.2 show that increasing P_R reduces the probability of the adding a server to the DQL list thereby

decreasing the overall execution time by a very small amount. However, in the GBRMAD technique from figures 6 and 7 it is seen that increasing P_R, increases the probability of the success of a transaction, thereby increasing the probable execution time execution time. Decreasing the P_R value results in less queries being transferred to the remote servers and as a result the probable execution time of transaction decreases. The increase in the number of servers in a group increases the execution time by a very small amount. Since the GBRMAD technique results in an up-to-date query value the increase in the transaction execution time can be justified.

Figures 8, 9, and 10 show the energy consumption at the servers. It is seen that the energy consumption for the GBRMAD technique is significantly lower as compared to the STE and ETS strategy. This mainly due to the two modes of operations of the servers. By going into doze mode, the server can save lot energy. The energy consumption of the GBRMAD technique is calculated for all the three cases of time lag for η number of transactions as discussed in section 4.3. The total energy consumption according equation 4.8 is independent of the time lag. However, for equation 4.9 and 4.10 the total energy consumption increases with the increase in time lag. Figures 8, 9 and 10 show the graphs for case a, case b and case c respectively. Increasing the time lag between any two consecutive transactions increases the energy consumption by a very small amount (not distinctly visible on graphs). However, still the energy consumption of the techniques by [Karumanchi, 1999] is greater than the GBRMAD technique. This is because the servers can go into doze mode when no transactions are being processed thereby conserving lot of energy. For constructing the graphs in Figure 8, 9 and 10, the access times for each mode are calculated. For case b) and case c) in section 4.3, the time lag is

considered to be 0.03 and 0.15 seconds respectively such that it satisfies the requirements of Δ for case b and case c.

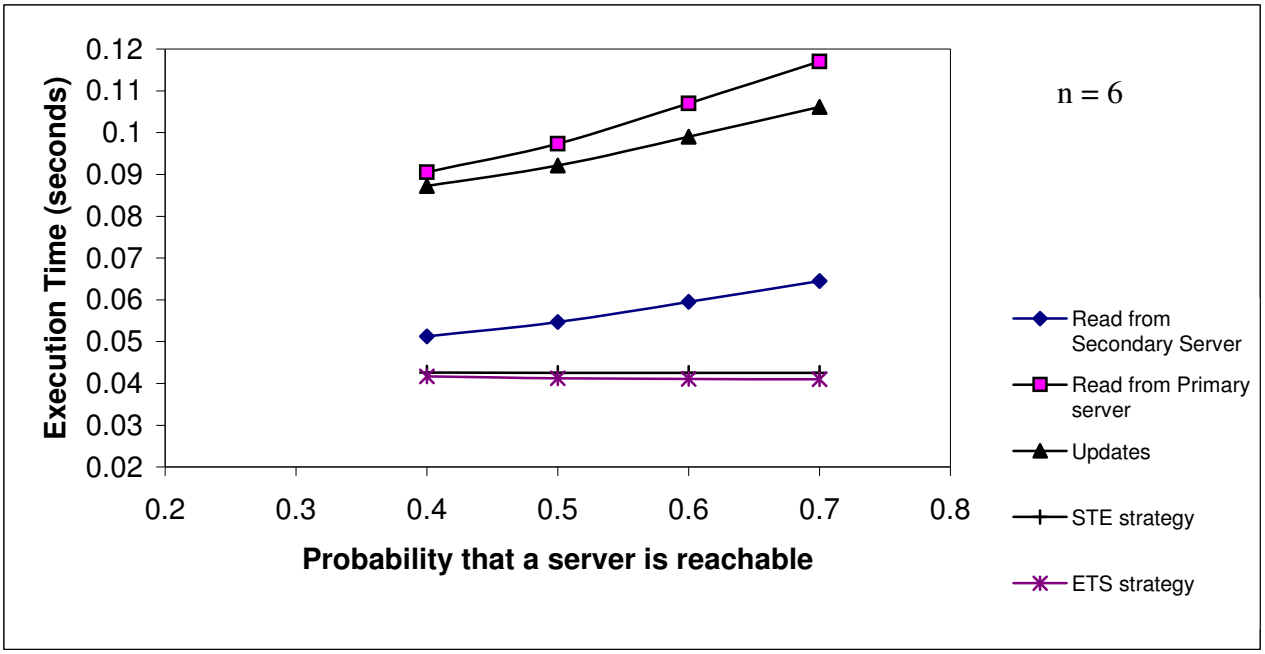

Figure 6: Execution Time Vs Probability that a server is reachable

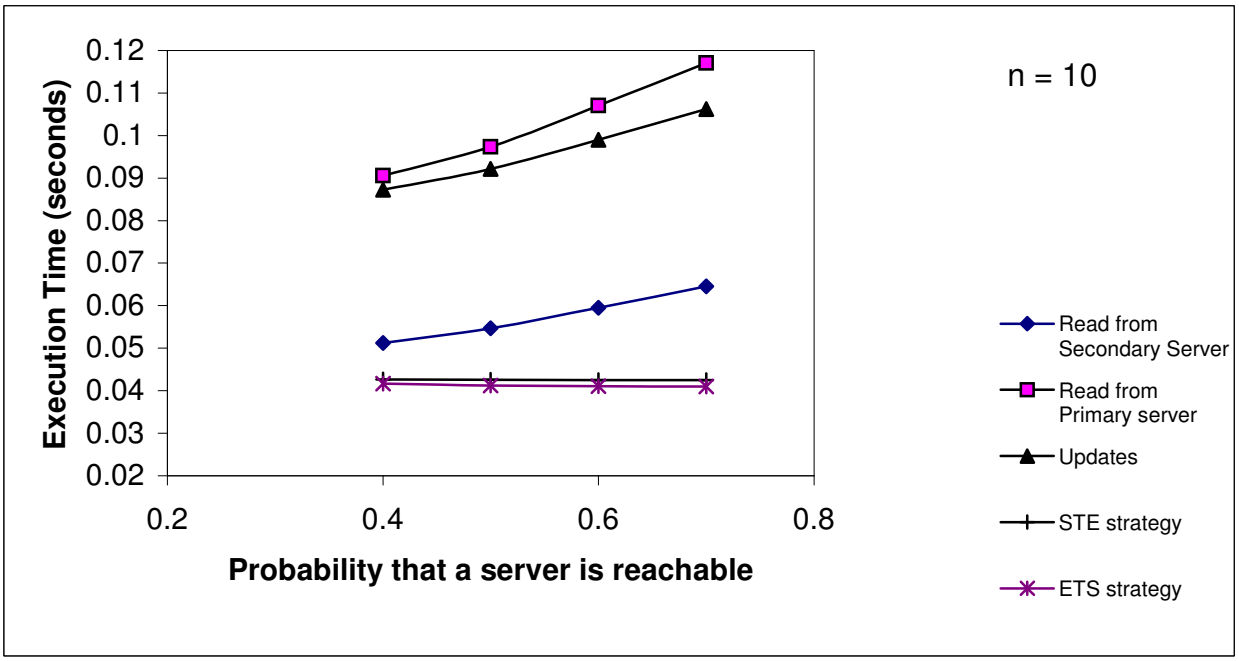

Figure 7: Execution Time Vs Probability that a server is reachable

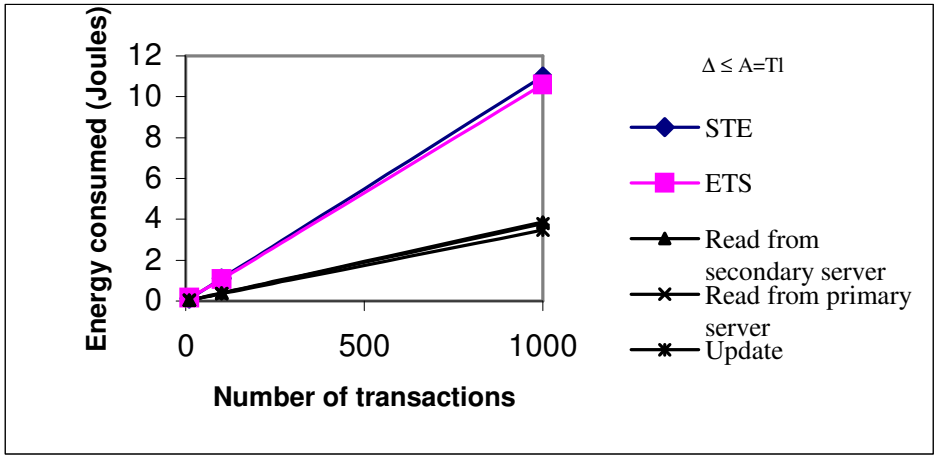

Figure 8: Total Energy consumption at a Server for $P_R = 0.5$.

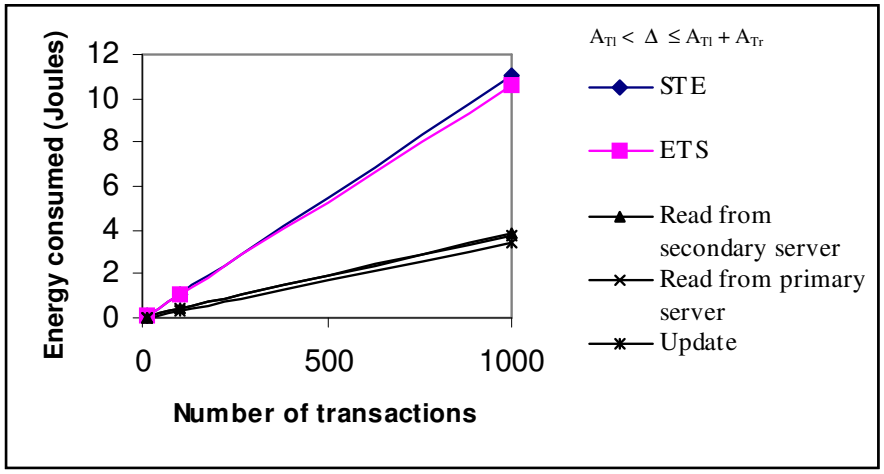

Figure 9: Energy Consumption at the server for Pr = 0.5

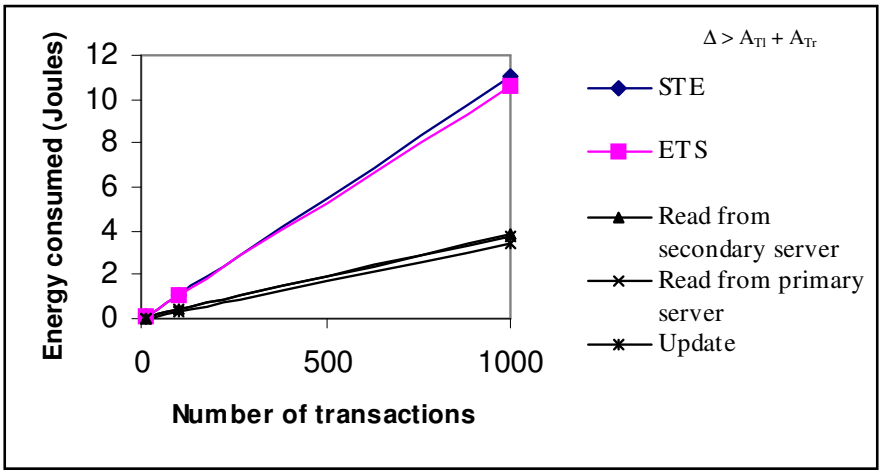

Figure 10: Energy Consumption at the server for Pr = 0.5

6 Conclusions and Future Work

In a mobile ad hoc environment, the available battery power and cost of communication play a vital role in allocating replicas. Hence, any replication scheme should take into consideration the available energy levels of each server and aim at reducing the communication cost. Frequent formation and joining of partitions being common phenomena, the replication scheme should aim at maintaining consistency and high availability of data at low cost. The GBRMAD data replication scheme uses the variation of the majority voting strategy for updates/reads. The GBRMAD data replication scheme selects servers having the best energy levels for soft transactions thereby ensuring that servers having lower energy levels do not get depleted of its energy very fast. This scheme thus takes into account the available energy levels at the servers for a soft transaction. Since for firm transactions the deadline has to be met, firm transactions access the nearest servers irrespective of the available energy levels. Transactions that do not require the most recent value have to access a single server thereby reducing the access time and

cost of communication as compared to the transactions requiring the most recent value. The execution time for a transaction is almost twice for the GBRMAD technique as compared to the STE and ETS strategy for $P_R = 0.4$ and increases with increase in P_R. However, this increase in execution time can be justified with more accuracy of the results obtained for the GBRMAD technique. The introduction of different modes of operations can significantly reduce the energy consumption at the server where a transaction is originated.

The GBRMAD scheme assumes the same initial storage capacity on all servers. In future this assumption can be relaxed and a method to distribute servers among group has to be studied. Simulation of the GBRMAD technique is to be performed to compare the efficiency of the technique as compared to the one proposed by [Karumanchi, 1999].

7 References:

[Ahuja, 1993] Ahuja R., Magnanti T., Orlin J., "Network Flows- Theory, Algorithms and Applications", Prentice Hall, pages 166-197, 1993

[Draffan, 1980] Draffan I. W., Poole F., "Distributed Databases", Cambridge University Press, Cambridge, pages 226-228, 1980.

[Dirckze, 1999] Dirckze R. A., "*Transaction Management in Mobile Multidatabases"*, Ph. D. Dissertation, Department of Computer Science, University of Oklahoma, Norman, 1999.

[Gruenwald, 2000] Gruenwald L., *"An Energy-Efficient Transaction Management Technique For Real-Time Mobile Database Systems in Ad Hoc Network Environments",* NSF Proposal, 2000.

[Huang, 1994] Huang, Y., Sistla, P., Wolfson, O., "*Data Replication for Mobile Computers."* Vol 5, pages 13-24, SIGMOD 1994.

[Imielinski, 1994] Imielinski T., Viswanathan S., "Adaptive Wireless Information Systems", in Proceedings of SIGDBS (Special Interest Group in Database Systems) Conference, October 1994

[Karumanchi, 1999] Karumanchi G., Muralidharan S., Prakash R., "*Information Dissemination in Partitionable Mobile Ad Hoc Networks"* Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems, 18-21 October, 1999, Lausanne, Switzerland.

[Mohan 1994] Mohan S., Jain R., "*Two user location strategies for personal communications services"*, IEEE Personal Communications, pages 42-50, First Quarter 1994.

[Wu, 1997] Wu, S., Chang, Y., "*An Active Replication Scheme for Mobile Data Management*." Published by National Science Council, project no. 87-2213-E-259-004, 1997.

[Shivakumar, 1997] Shivakumar N., Jannink J., Widom J., "*Per-user Profile Replication in Mobile Environments: Algorithms, Analysis, and Simulation Results*" Mobile Networks and Applications, pages 129-140, vol. 2, 1997.