Agent Based Energy Aware Real Time Mobile Transaction Management

J.L.Walter Jeyakumar
Associate Prof., Dept. of Computer Science,
St. Xavier's college,
Tirunelveli, Tamilnadu, INDIA
walterjeya@hotmail.com

***Abstract*— This paper presents a energy aware transaction scheme for mobile nodes. This model proposes an environment in which mobile users can have simultaneous access to data by using the cache stored in the nearest Fixed Agent. Data Access Manager module at the Fixed Agent controls the concurrency using invalidation technique. This approach supports disconnected transaction execution by allowing a disconnected mobile host to find out a nearest neighbor to receive invalidation report on behalf of it. This paper also addresses the issues of time and energy constraints of real time transaction processing. Four levels of priority are assigned to the requesting mobile transactions based on available energy and connectivity. The proposed transaction management framework has been simulated in J2ME and NS2 and performance of the scheme is compared with contemporary frameworks.**

Keywords– Transaction, Concurrency Control, Mobile Host, Fixed Agent, Cache invalidation

I. INTRODUCTION

Mobile computing is the process of computation on a mobile device. Mobile computing offers mobility with computing power. It provides distributed computations on a wide range of devices, systems and networks which are mobile, synchronized and interconnected via mobile communication standards and protocols. The features of mobile computing systems which make them different from the stationary systems are their wireless network connectivity, their small size, the mobile nature of their use, their power sources and their functionalities that are particularly suited to the mobile user. Frequent disconnections, mobility, limited battery power and resources pose new challenges to mobile computing environment. Frequent aborts due to disconnection should be minimized in mobile transactions. Correctness of transactions executed on both fixed and mobile hosts must be ensured by the operations on shared data. Blocking of mobile transactions due to long disconnection periods should be minimized to reduce communication cost and to increase concurrency. After disconnection, a mobile host should be able to process transactions and commit locally. Mobile computing provides the possibility of concurrent access of data by mobile hosts which may result in data inconsistency. Concurrency control methods have been used to control concurrency. Due to limitations and restrictions of wireless communication channels, it is difficult to ensure consistency of data.

In this paper, we present a priority based framework for real time transaction processing. Frequently accessed data are cached in the Fixed agents situated at different locations in the fixed wired network. An MH can connect to the nearest Fixed Agent to access data. But upon update request by a MH, updating is done at the local cache and invalidation report is sent to all the mobile hosts which have already accessed the same data. This will force the mobile hosts to refresh their data values. Data Access Manager (DAM) at the fixed agent is responsible for concurrency control and data invalidation. In order to give priority to the mobile nodes running on low power and with low connectivity, four levels of priority are used. This paper considers the issues of time and energy constraints of real time transaction processing. This framework also takes into account transaction update during disconnection.

The remaining part of this paper is organized as follows. Section 2 summarizes the related work. Section 3 focuses on the agent based architecture. Section 4 illustrates the proposed mobile transaction framework. Section 5 gives the performance analysis and in Section 6 the conclusion is presented.

II. RELATED WORK

Concurrency control techniques are used to avoid data inconsistency, when simultaneous access to data is made at the server. Conventional locking based concurrency control methods like centralized Two Phase locking and distributed Two Phase locking are not suitable for mobile environment. The system overhead that arises due to concurrency control mechanism can create a serious performance problem because of low capacity and limited resources in mobile environment [1]. Moreover, it makes mobile hosts to communicate with the server continuously to obtain and manage locks [2].

In Timestamp approach, the execution order of concurrent transactions is defined before they begin their execution. The execution order is established by associating a unique timestamp to every transaction. When two transactions conflict over a data item, their timestamps are used to enforce serialization by rolling back one of the conflicting transactions [3]. In optimistic concurrency control with dynamic time stamp adjustment protocol, client side write operations are required. But it may never be executed due to delay in

execution of a transaction [4]. In multi version transaction model [5], data is made available as soon as a transaction commits at a mobile host and another transaction can share this data. But data may be locked for a longer time at a mobile host before the lock is released at the database server.

In [6], a transaction model for supporting mobile collaborative works was proposed. This model makes use of Export-Import repository which is a mobile sharing work space for sharing data states and data status. But in the Export-Import repository based model, locking is the main technique which has the following disadvantages. (i) More bandwidth is needed for request and reply since the locking and unlocking requests have to be sent to the server. (ii) Disconnection of mobile host or a transaction failure will result in blocking of other transactions for a long period. Transaction Management solution proposed in [7], is for reducing energy consumption at each MHs by allowing each MH to operate in three modes, Active, Doze, and Sleep thus providing a balance of energy consumption among MHs. In this paper, we propose a new mode called Energy Conservation mode (EC) in which MH alternates between sleep and doze mode during certain time periods, thereby minimizing number of transactions missing deadlines.

In [8] , AVI (Absolute Validity Interval) was introduced for enforcing concurrency control without locking. AVI is the valid life span of a data item. But it calculates AVI only based on previous update interval. In [9], a method based on PLP (Predicted Life Period), which takes care of the dynamicity of the life time of data was proposed. Here, life span of data is predicted based on the probability of updation of data item. This method makes PLP of data item very close to the actual valid life span of a data item. But this approach did not take into account the disconnection issue. The proposed approach is better than this framework since it allows the MHs to receive invalidation reports even if they become disconnected. Moreover, it takes into consideration, energy and time constraints of real time transaction processing. It also includes four levels of priority based on energy availability and connectivity in mobile nodes to reduce number of transactions that are aborted because of disconnections.

III. AGENT BASED ARCHITECTURE

The proposed Agent based architecture model is illustrated in Fig. 1. The model consists of Data Base Server (DBS), Fixed Agents (FAs) and Mobile Hosts (MHs). Every MH has a limited communication range. An MH can directly connect and communicate with other MHs which are within its communication range. Any MH can communicate with another MH in multiple hops using intermediate MHs in case they are not within each other's communication range. Fixed Agents located at different places are connected to the Data Base Server through wired network. In Fixed Agents, cache is used to store the frequently accessed data. An MH can find out the nearest Fixed Agent and connect to it to access data from the cache. Data Access Manager (DAM) module at the Fixed

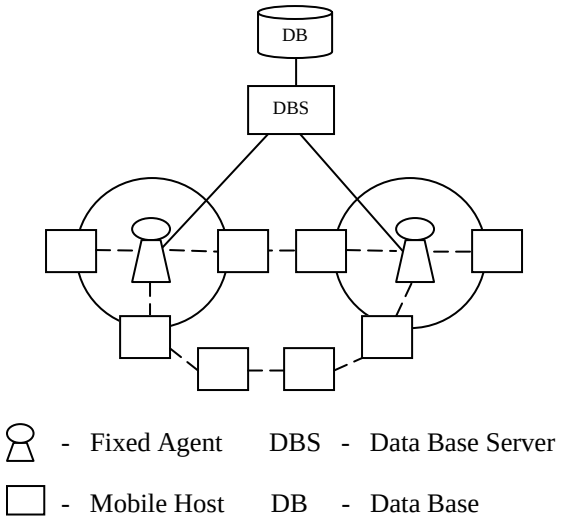

Figure 1. Agent Based Architecture model

Agent is responsible for coordinating the data access process from the cache. When data request is made for the first time, data is retrieved from the server and stored in the cache. Subsequent requests are handled by the Data Access Manager module itself. When a mobile host requests for data update, after local updation of the data item, invalidation report is sent to all the mobile hosts that have already accessed the same data. This makes all the mobile hosts to refresh their data values.

IV. PROPOSED MOBILE TRANSACTION FRAMEWORK

A mobile transaction framework that deals with real time transaction management is illustrated in this section. Mobile hosts can directly connect to the server. But simultaneous access to data at the server will increase the system overhead. To overcome this disadvantage, data are cached in the Fixed Agent. The Data Access Manager at the Fixed Agent is responsible for enforcing concurrency and cache invalidation.

A. Information available in the MH

In this proposed architecture, local database of an MH consists of following information.

- ID is an unique id of an MH
- Position is used to get the coordinates of an MH from GPS
- RTR is for storing Radius of Transmission Range in an MH
- A_{ij} defines status of Energy Availability and Connectivity in an MH.
  [A_{00} – Low Energy & Low Connectivity,
  A_{01} – Low Energy & High Connectivity,
  A_{10} – High Energy & Low Connectivity,
  A_{11} – High Energy & High Connectivity]

Each Fixed Agent periodically broadcasts its ID and position and MHs will store this information in the FA_list in their local databases.

B. Two types of transactions

In real time transaction processing, transactions should complete before their deadlines. There are two types of transactions namely Firm and Soft [10]. A firm transaction is aborted when it misses a deadline i.e., its value becomes zero. Since a soft transaction has two deadlines, it can be executed even after its first deadline expires. The value of a soft transaction decreases after the first deadline and becomes zero after the second deadline [7].

C. The states of an MH

In order to reduce energy consumption at each MH and to provide a balance of energy consumption among MHs, each MH will operate in three states [7] as given below.

- Active state: The CPU is in a completely functional state and its communication device can transmit and receive signals.
- Doze state: The CPU is working at a lower rate. The communication device can receive signals.
- Sleep state: The CPU and communication device are not in a functional state

A mobile node will be in any of the three states at any time. For our scheme, we have taken the modes [7] with a slight modification. Instead of sleep mode, we have introduced a new mode called EC (Energy Conservation) mode in which MH alternates between sleep and doze states during certain time periods.

D. Energy and Connectivity evaluation

Mobile Hosts all the time maintains its energy availability and connectivity. Connectivity is evaluated based on signal strength. When signal strength goes below one fourth of total strength, connectivity is considered as Low. When available energy goes below 25% of total energy level, then energy availability is considered as Low. The status of of an MH based on Energy Availability and Connectivity (A_{ij}) can be A_{00} – Low Energy & Low Connectivity, A_{01} – Low Energy & High Connectivity, A_{10}–High Energy & Low Connectivity and A_{11} – High Energy & High Connectivity. When Data Access Manager receives a transaction request from a mobile host, it assigns a priority level using A_{ij}. A mobile host with low energy and low connectivity is assigned the highest priority. Other levels of priority are assigned according to the various possibilities as given in Table I.

TABLE I. LEVELS OF PRIORITY

Status of an MH(A_{ij})	Energy Availability(i)	Connectivity (j)	Priority
A_{00}	Low	Low	1
A_{01}	Low	High	2
A_{10}	High	Low	3
A_{11}	High	High	4

E. Concurrency Control Scheme

When more number of mobile hosts are accessing data simultaneously, the problem of data inconsistency arises. This problem can be solved if we use an efficient concurrency control mechanism. When data request is made for the first time, data is retrieved from the server and stored in the cache. Future requests for data are managed directly by the Data Access Manager.

Data Access Manager uses a suitable data item format to store data as quintuple [9] in the cache. It has (id, TLU, PLP, dataval, T_n) where id denotes unique Id of the data item, TLU indicates time of Last Update, PLP is Predicted Life Period, dataval is current value of the data item and T_n denotes number of transactions that concurrently access the data item.

When Data Access Manager fetches data for the first time from the server, it sets TLU to current time, PLP to optimal time based on the nature of data item and T_n to 1. T_n is incremented whenever a new data access request is made. Data in the cache becomes invalid, once it is updated in the server. Life span of a data item is predicted using PLP. It makes use of the probability of updating as a basis for setting valid life span of a data item. In PLP interval, data item is valid and all the mobile hosts can access same data item concurrently. When a MH makes update request or PLP expires, the data item is invalidated. Now PLP is modified and invalidation report is sent. The predicted life period of data item is computed using the following formula as given in [9]. PLP=PPLP ± (p*PPLP) Where PPLP is Previous Predicted Life Period and p is predicted probability of updation of data item. p = Total_updates / T_n. It is the ratio of data item update to data item access. Since predicted probability of updating is based on recent past history of updating rate, it is highly probable that PLP is very close to the actual validity interval of the data item.

F. Transaction Request by MH to Fixed agent

For submitting the transaction request, we use the technique proposed in [7] with some modifications. When an MH initiates a transaction, it will find out the nearest FA by searching its local database. Then it will submit the transaction request to this nearest FA after finding the route. MH will calculate the amount of time it must wait for the FA to return the transaction results using the run time estimate of the transaction and possible delay due to disconnection obtained from the transaction history as given below.

Wait_time = r + c1 where

r is the run time estimate of the transaction and
c1 is the estimated communication time for submitting the transaction to the FA and getting back the result.

If the MH could not receive the result before this Wait_time, it will assume that the nearest FA is disconnected. Hence, the MH will once again search its local database to find the next nearest FA and submit the transaction request after finding the route to this FA. If the MH moves away after

making a transaction request to an FA, it will intimate its current position to the FA. If the result is received by the requesting MH, it will search the data item in the latest invalidation report. If it exists, it will once again make a transaction request. Otherwise, if either energy availability or connectivity of the MH is low, Invalidation_Routing module is called. Here, the MH elects a nearest neighbor with high energy availability and high connectivity (A_{11}) to receive invalidation report if any, on behalf of the MH. This nearest neighbor will intimate this invalidation report to the MH, when it reconnects. This algorithm is shown in Fig. 2.

```
Trans_Req_to_FA(T,T_type,T_deadline,MH_pos,MH_ID)
// Transaction T with type T_type and deadline T_deadline is initiated by an
MH whose ID is MH_ID and position is MH_pos //
Begin
   Search the FA_List to find the nearest FA that is not yet visited (FA')
   Find a route to FA'
   Intimate the status of Energy availability and Connectivity (Aij) of the MH
     to DAM at FA'
   Start execution of the transaction T locally
   If Data Read
      Submit Read Request to DAM at FA'
   Else If Data Update
            Submit Update Request to DAM at FA'
       End If
  End if
  Set Wait_time = val
  Set timer = Wait_time
  While timer ≠ 0 do
     If the result of T is got
        Find a route to FA'
        Send an Ack to FA'
        Search data item in the latest invalidation report
        If data item exists
           Trans_Req_to_FA(T,T_type,T_deadline,
             MH_pos,MH_ID)
        Else if (Aij = A00 or A01 or A10)
             Invalidation_Routing()
           End if
        End if
        Commit
        Exit
     End if
     timer--
  End while
  T_deadline = T_deadline – Wait_time
  If T_deadline = 0   // T has missed deadline //
     Abort T
  Else
     Mark FA' as visited
     Trans_Req_to_FA(T,T_type,T_deadline,MH_pos,MH_ID)
  End if
End

Invalidation_Routing()
Begin
   The MH elects the nearest neighbor with A11 (MHp) and disconnect
   The MHp receives the invalidation report if any and intimates the MH
     when it reconnects
   Search data item in the invalidation report
   If data item exists
     Trans_Req_to_FA(T,T_type,T_deadline,MH_pos,MH_ID)
      End if
End
```

Figure 2. MH execution algorithm

G. Function of Data Access Manager

After receiving a transaction request from an MH, DAM module at FA will pass it to the Transaction Scheduler (TS). The TS will use priority based efficient scheduling algorithm based on energy availability and connectivity for scheduling the transactions. After scheduling, DAM will take the first transaction from the queue. If the data is in the cache of the FA after the MH makes a Read Request, it will update T_n. Otherwise it will fetch the data from the server and initialize the quintuple. Now the data is submitted to the MH after finding the route. If update request is made by the MH, DAM will update the data item locally and FA will broadcast the invalidation report to all the MHs that have already accessed the same data item. This forces all the transactions to refresh their data values. This update request is now forwarded to the server. The server updates the data and sends invalidation confirmation along with the updated value (See Fig. 5). Once Data Access Manager receives the confirmation, it updates the quintuple in the cache. The data in the cache is invalidated if updating is made in the server or PLP expires.

If the MH that makes a transaction request is in active mode, it will receive the result. If the MH is in doze mode, and if the transaction is firm, then MH will wake up and receive the result so as to meet the deadline. For the soft transaction, the MH will decide whether it should come to the active mode to receive the result or to remain in the doze mode for reserving its energy for firm transactions. We use EC (Energy Conservation) mode in which MH alternates between sleep and doze mode during certain time periods. If the requesting MH is in EC mode at the time of receiving the result, it will not be able to receive the result until it comes to doze mode. This EC mode minimizes the number of transactions missing deadlines, since it conserves energy during sleep time period. If the transaction is firm, DAM will abort the transaction if it does not receive any acknowledgement till the deadline of the transaction. Otherwise, the slack time is computed using the second deadline of the transaction. It will abort the transaction if the slack time is zero. Otherwise, the slack time is divided into some time intervals and the result is submitted again to the requesting MH during those intervals. The algorithm is shown in Fig. 3.

H. Priority based real time transaction scheduling

Fixed Agent will schedule transactions by means of a scheduling algorithm that takes into account transaction types (firm and soft), deadlines of the transactions, energy availability and connectivity of the MHs. We use the technique proposed in [7], where the slack time s of a transaction is calculated based on its deadline d, runtime estimate c, probability of disconnection Pd, and average time loss due to disconnection Td using the formula given below.

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

The values of Pd and Td are obtained from the transaction history as follows. The FA can keep track of how many times

```
Data_Access_Manager(T,T_ID,T_R_type,T_type,T_d,T_e,MH_ID,MH_pos,
  A_ij)

// DAM module will be executed when FA receives a transaction request
T_R_type of transaction T with ID T_ID, transaction type T_type, deadline
T_d, run time estimate T_e, Requesting Mobile Host ID MH_ID, Requesting
MH position MH_pos, Status of Energy availability and Connectivity A_ij //
Begin
   Trans_Schedule(T_ID,T_type,T_d,T_e,A_ij) and take the first transaction
      T1 from the queue
   If T1_R_type is Read Request
      If data is in the cache of FA
         Update T_n
      Else
         Fetch data from server and initialize quintuple
      End if
      Send result of T to the MH after finding its route
   Else If T1_R_type is Update request
         Update locally and broadcast invalidation report
         Forward update request to server
         If Server Update / PLP expiration
            Update quintuple
            Send result of T to the MH after finding its route
         End if
      End if
   End if
   While FA has not received Ack from requesting MH and slack time for
     First deadline of T1 > 0 do
      If FA has received an Ack
         Remove T1 from the active transaction list
         Exit
      End if
   End While
   If FA has not received an Ack from the requesting MH
     // MH is in Energy Conserving (EC) mode //
     If T1_type is firm
        Abort the transaction
     Else // Transaction type is soft //
        Calculate the slack time using second deadline
        If slack time =0
           Abort the transaction
        Else
           Divide slack time into some time intervals and send the result
              again to the requesting MH in each time interval until FA
              receives an Ack
        End if
     End if
   End if
End
```

Figure 3. Data Access Manager Algorithm

each MH had been disconnected when the FA wanted to submit the result of a transaction to it and what was the duration of the disconnection and the average time loss due to disconnection for a particular MH. Here, transactions that have less slack time are scheduled before transactions with more slack time.

First (Highest) priority is given for all the transactions whose requesting MH has low energy and low connectivity (A_{00}). Second priority is given for all the transactions whose requesting MH has low energy and high connectivity (A_{01}). Third priority is given for all the transactions whose requesting MH has high energy and low connectivity (A_{10}). Fourth (Lowest) priority is given for all the transactions whose requesting MH has high energy and high connectivity (A_{11}). If the slack time of a soft transaction is negative, then its slack time and priority will be recalculated. The transaction is discarded, if the recalculated slack time is negative. The algorithm is given in Fig. 4.

```
Trans_Schedule( T_ID,T_type,T_d,T_e,A_ij)
Begin
   Calculate the slack time for all transactions using formula 1
   If the slack time of a transaction T is less than 0 and
      T_type is firm
         Remove T from the queue and discard it
   End if
   For all transactions whose requesting MH has low energy and
      low connectivity (A_00) sort the transactions that have slack
      time > 0 and assign First (Highest) priority
   For all transactions whose requesting MH has low energy and
      high connectivity (A_01) sort the transactions that have slack
      time > 0 and assign Second priority
   For all transactions whose requesting MH has high energy and
      low connectivity (A_10) sort the transactions that have slack
      time > 0 and assign Third priority
   For all transactions whose requesting MH has high energy and
      high connectivity (A_11) sort the transactions that have slack
      time > 0 and assign Fourth (Lowest) priority
   If the slack time of a soft transaction is negative
      Recalculate the slack time using its second deadline and
      Formula 1
      If the recalculated slack time of a soft transaction is negative
         Discard the soft transaction
      Else
         Recalculate its priority
      End if
   End if
End
```

Figure 4. Priority based real time scheduling algorithm

```
Server_Execution()
Begin
      Wait for connection
      If connection request
         Connect authorized Mobile Hosts
         If Data Read
            Send data item
         Else if Data Update
               Update data and send invalidation confirmation
                  with updated data
            End if
         End if
      End if
End
```

Figure 5. Server execution algorithm

V. PERFORMANCE ANALYSIS

Simulation for the framework is done by considering 25 mobile nodes and 10 Fixed Agents in Pentium Dual Core System @ 2.4 GHz with 3 GB RAM using J2ME and NS2. The results of the analysis are shown in Fig. 6, Fig. 7 and Fig. 8. Response time is calculated as the time taken to service the request made by the mobile host.

Compared to the approach that uses MANET in [7], our approach gives interesting results. It is found from Fig. 6 that, as the number of transactions increases, the number of transactions missing deadlines increases gradually in our approach (Cache in FA) whereas, in the approach using MANET, there is a steep increase. For example, for 20 transactions, the number of transactions missing deadlines is almost the same for both approaches. But for 50 transactions, the proposed approach provides an improvement of 50%. This is due to the fact that the cache in Fixed agent allows the transactions to finish quickly. Moreover, the use of EC mode enhances the chances of completion of transactions before deadline without abortion.

Similarly, we get better results in our approach due to the cache in the Fixed Agent, when we consider the number of Firm transactions that are completed before their deadlines. For example in Fig. 7, as the number of firm transactions increases, there is a gradual increase in the number of completed firm transactions for both approaches. But the proposed approach shows an improvement of over 60% for 50 transactions when compared with the MANET based approach. Also, in our approach, we get less average response time compared to the approach using MANET. For example in Fig. 8, for 10 transactions, the average response time taken in both the approaches is almost the same. But for 50 transactions, the proposed model provides an improvement of 40%. Once again, this is due to the cache in FA which gives faster access.

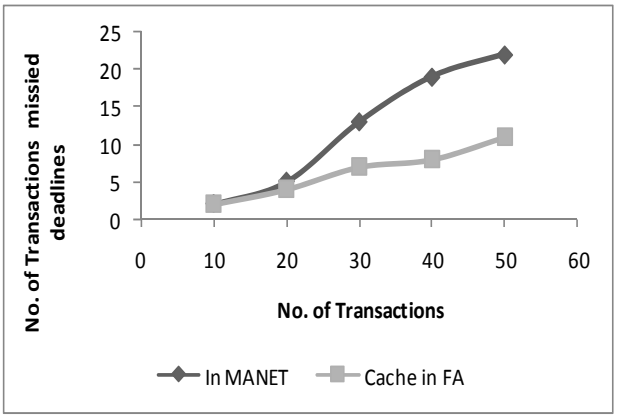

Figure 6. Analysis of no. of Transactions missing deadlines

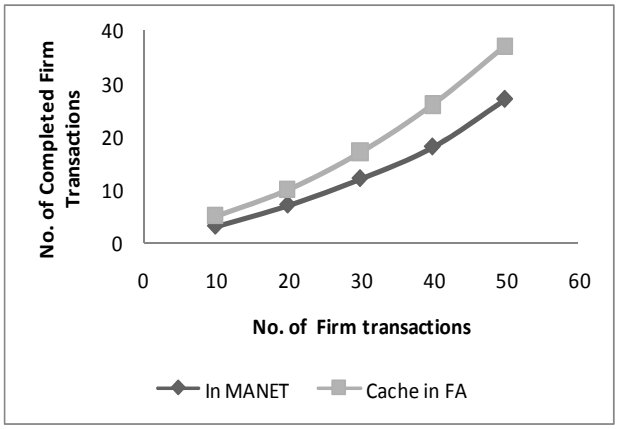

Figure 7. Analysis of no. of completed Firm Transactions

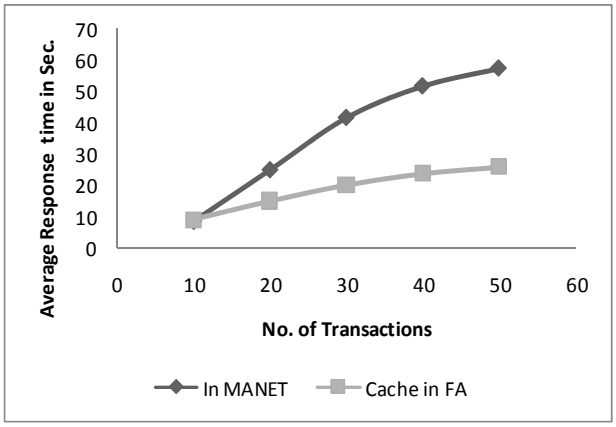

Figure 8. Analysis of average Response time

VI. CONCLUSION

In this paper, we have proposed a priority based transaction frame work for real time transaction processing. In order to avoid abortion of transactions due to low energy availability and low connectivity, we have used four levels of priority. We make use of Fixed Agents in the wired network to store the cache. This cached data can be accessed by the mobile hosts when they get connected. By using a Fixed Agent and concurrency control without locking for accessing data, we claim that message communication costs and database update costs are minimized to a larger extent. Also the use of cache in the Fixed Agent minimizes the number of transactions missing deadlines. This framework supports disconnected operations by choosing a nearest neighbor to receive invalidation report on behalf of the MH.

REFERENCES

[1] Vijay Kumar, *"Mobile Database Systems",* Wiley-Interscience, 2006.

[2] Victor C.S., Kwok wa Lam and Son, S.H., "Concurrency Control Using Timestamp Ordering in Broadcast Environments", *The Computer Journal, Vol.45* No.4, pp. 410-422, 2002.

[3] P.A Bernstein, V. Hadzilacos and N. Goodman, "*Concurrency control and Recovery in Database Systems",* Addison Wesley, 1987.

[4] Ho-Jin Choi, Byeong-Soo Jeong, "A Timestamp Based Optimistic Concurrency Control for Handling Mobile Transactions", *ICCSA 2006, LNCS 3981*, pp. 796-805, 2006.

[5] Madria, S. K., M. Baseer, and S. S. Bhowmick, "A Multiversion Transaction Model to Improve Data Availability in Mobile Computing,", *CoopIS/DOA/ODBASE*, pp. 322-338, 2002.

[6] Le, H. N., and M. Nygård, "A transaction model for Supporting mobile Collaborative Works", *IEEE*, pp. 347-355, 2007.

[7] Le Gruenwald, Shankar M. Banik, Chuo N. Lau, "Managing real time database transactions in mobile ad-hoc networks", Springer, pp. 27-54, 2007.

[8] Salman Abdul Moiz, Mohammed Khaja Nizamuddin, "Concurrency Control without Locking in Mobile Environments", *IEEE*, pp. 1336-1339, 2008.

[9] Miraclin Joyce Pamila J.C and Thanuskodi K., "Framework for transaction management in mobile computing environment", *ICGST-CNIR Journal*, pp. 19-24, 2009.

[10] Ramamirtham, Real time databases, "Distributed and Parallel Databases", vol 1,no. 2, pp. 199-226, 1993.

Dr J.L. Walter Jeyakumar has obtained his M.C.A from Bharathidasan University in 1988 and completed M.Phil and Ph.D from Manonmaniam Sundaranar University in 2007 and 2012 respectively. He is currently working as Associate Professor at St. Xavier's college, Palayamkottai, Tirunelveli. He has 25 years of teaching experience. His areas of interests are networks and Mobile Computing. He has published many research articles in the International / National Conferences and Journals.