An Analytical Approach for Guaranteeing Concurrency in Mobile Environments

Salman Abdul Moiz, and Lakshmi Rajamani

Abstract—**In a mobile database environment, multiple mobile hosts may access the shared data item at the same time. This may lead to inconsistency of data items. The traditional pessimistic protocols are not suitable in mobile environments because of disconnections of mobile hosts for invariant time. The timeout based protocols solves the problem of starvation of resources, but with an increase in number of rollback operations. The Analytical approach is a variation of Timeout based commit protocols where a transaction is executed only if the expected time for execution is within the current timer value. Experimental results show better throughput, and less waiting time for individual transactions**

Index Terms—**Execution Time, Fixed Host, Mobile Host, Timer.**

I. INTRODUCTION

The general characteristics of mobile environments like mobility and frequent disconnections makes the traditional locking mechanism unsuitable for achieving concurrency control. In a mobile computing environment, the characteristics of mobile environment make data accessibility a challenging issue

One of the key challenges in mobile database environment is the ability to simultaneously access the data items irrespective of the physical locations of mobile users. To handle the concurrency control issue, various concurrency control techniques have been proposed in literature which is usually based on three mechanisms viz., locking, timestamps and optimistic concurrency control. Though these techniques are well suited for the traditional database applications they may not work effectively in mobile database environments.

A Mobile Host may lock the data items needed for executing a transaction and may be disconnected for indefinite amount of time leading to starvation. To solve this problem various timeout mechanisms are proposed [1] [2]. However in these mechanisms the transaction is executed even when the time for execution of a transaction is higher. In this paper we propose a strategy which increases the throughput of the system at the same time the waiting time of a transaction may decrease. The idea behind this technique is that if the transaction is expected to take more time then the current timer value, it is rolled back by updating the current timer value. The rolled back transaction may be successfully executed in future.

The remaining part of this paper is organized as follows: Section 2 summarizes the survey of existing techniques, section 3 describes the architecture of mobile environment, section 4 specifies the proposed concurrency control strategy, section 5 specifies the performance metrics and section 6 concludes the paper

Salman Abdul Moiz is a Research Scientist at Centre for Development of Advanced Computing, Bangalore, India (phone: +091-080-28523300; e-mail: salman@cdacbangalore.in).
Dr. Lakshmi Rajamani is a Professor & Head, CSE Department, University College of Engineering, Osmania, University, Hyderabad, India (e-mail: drlakshmiraja@gmail.com).

II. RELATED WORK

Multiple mobile hosts may access the same data items leading to inconsistency. Several valuable attempts to efficiently implement the concurrency control mechanisms have been proposed. Concurrency control strategies proposed in the literature considers only a subset of performance issues.

The two phase locking protocol in not suitable for mobile environments as it requires clients to continuously communicate with server to obtain locks and detect the conflicts [3]. A Timeout based Mobile Transaction Commitment Protocol is a non-blocking protocol, however it faces the problem of time lag between local and global commit. The Mobile 2PC protocol preserves 2PC principle; however it assumes that all communicating partners are stationary with permanent available bandwidth [4]. In [5] Mobile speculative locking protocol is introduced to reduce the blocking of transaction if two phase locking is employed. This approach requires extra resources at the mobile host to carry out speculative execution. An optimistic concurrency control technique detects and resolves data conflicts in the phase of transaction validation. In a mobile environment if the transaction validation is done on the server, it may lead to delayed response causing overhead at the server [6]. An Optimistic Concurrency Control with Dynamic Time stamp Adjustment Protocol requires client side write operations. However because of the delay in execution of a transaction, it may never be executed [1]. In [7] [8], the authors propose an enhanced conventional optimistic concurrency control algorithm which terminates a particular transaction whenever a conflict is detected. However because of early termination a transaction need to be initiated again and again. This increases the uplink bandwidth.

In timer based strategies, if the timer value is small compared to the expected time for execution of a transaction, it will still continue the execution and later rolled back due to the expiry of timer value. This reduces the throughput of the system. In the proposed strategy a transaction is executed with an aim of increasing the throughput of the system. However the time for execution of a transaction is evaluated based on the history state i.e. the time taken for execution of a successful transaction.

III. MOBILE DATABASE ARCHITECTURE

The following figure specifies the reference model for mobile computing environments. It consists of two entities Fixed Host (FH) and Mobile Host (MH) respectively. Terminals, desktops, servers are the Fixed Host which are interconnected by means of a fixed network. Large databases can run on servers that guarantee efficient processing and reliable storage of database.

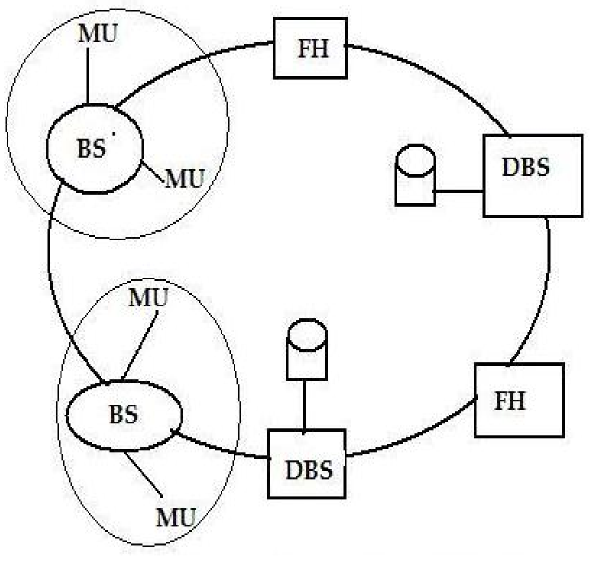

Fig. 1 Mobile Database Architecture

Mobile Hosts (MH) like Palmtops, Laptops, PDA's or Cellular phones is not always connected to the fixed network. In the offline transactions, the transactions are executed on the mobile hosts. The request for a transaction is initiated at the mobile host then the required data items needed to execute the transaction are read by the mobile host. After reading the data items, the mobile host may be disconnected to save battery consumption. When the transaction is successfully executed at the mobile host, the results are integrated with DBS at fixed host.

A Mobile unit connects to a fixed host through a wireless link A Base station connects to a mobile unit and is equipped with a wireless interface. It is also known as a Mobile Support Station. During execution of a transaction, a Mobile Hosts (MH) may move from once cell to another (handoff). It might also be disconnected intentionally to save the power consumption or bandwidth.

Mobile users are more likely to face with more disconnection because of the properties of the mobile environment. The interface between the mobile clients and fixed hosts is realized by the base stations. Base stations act as an interface between the mobile computers and fixed hosts. The base station acts as the coordinator, the mobile host and the Database Systems (DBS) acts as the participants. The coordinator will be responsible to make the final commit decision.

IV. ANALYTICAL APPROACH FOR CONCURRENCY CONTROL

In the timeout based mechanisms the rollback or abort decision is made when the timer expires. In the proposed strategy the decision regarding the state of the transaction is made at the beginning. This is possible if the expected time for execution is known.

A. *Execution Time or Deadline*

Transactions are associated with timing constraints in form of deadlines, independent of whether they originate from the mobile clients or the static hosts over wired or wireless networks. For instance, it may be a financial or opportunity loss if a stock-trading transaction cannot be completed with a certain timing constraint, disregarding whether the stock trader is submitting the transaction (purchasing or read-only) in his office (wired) or on a ride to somewhere (wireless)[9].

In addition, the temporal validity of some data objects such as stock prices or sensor data poses another type of timing constraints to the database systems. Transaction correctness is then defined as meeting its timing constraints and using data that is absolutely and relatively timing consistent [10][11]. As such every transaction has to be associated with Execution time or deadline. In [12], the problems of disconnection and fault recovery are tackled using time-out management strategy

The time needed to complete a task is give by

$$\mathbf{T}_{\text{Execution}} = \mathbf{T}_{\text{Processing}} + \mathbf{T}_{\text{Transfer}}$$

$\mathbf{T}_{\text{Processing}}$ is the actual time taken to execute the transaction at mobile host

$\mathbf{T}_{\text{Transfer}}$ represents time taken to send lock/unlock request to the fixed host, which varies depending on terminal connectivity.

B. *Concurrency Control Strategy*

When multiple mobile hosts requests for the same shared data item for execution of a transaction, the base station which acts as the coordinator locks the data items requested by one of the mobile host. As the mobile hosts are prone to disconnections, the other mobile host which requires the same shared data item has to wait indefinitely. To overcome this problem each and every transaction is set a timer within which it is expected to complete its execution. Otherwise it may be rolled back.

A transaction which was not executed within certain time period "t" by a particular mobile host may not be executed within the stipulated time in future due to the connectivity or computing power of a mobile host as such for each rolled back transaction the timer t is to be increased by a small factor "∂". The step factor may vary from one mobile application to another. This may be maintained at the fixed host. If the timer "t" reaches a threshold value "T" due to consecutive roll back operations of a particular transaction, then the transaction may be aborted.

In the proposed strategy for each transaction its time for execution is maintained at the fixed host. The coordinator evaluates the time for execution with the current timer value, to determine whether the transaction can proceed for execution. Fig. 2 describes the behavior of the proposed concurrency control strategy.

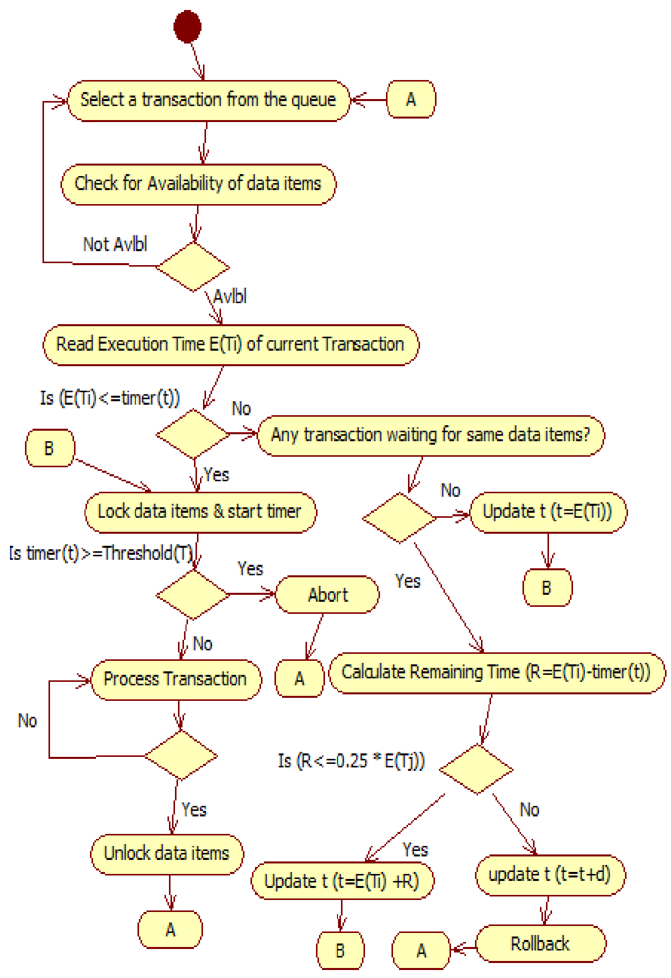
Select a transaction from the queue
A
Check for Availability of data items
Not Avlbl
Avlbl
Read Execution Time E(Ti) of current Transaction
Is (E(Ti)<=timer(t))
No
Any transaction waiting for same data items?
B
Yes
Lock data items & start timer
No
Update t (t=E(Ti))
Is timer(t)>=Threshold(T)
Yes
Abort
Yes
B
No
A
Calculate Remaining Time (R=E(Ti)-timer(t))
Process Transaction
No
Is (R<=0.25 * E(Tj))
Yes
No
Unlock data items
Yes
Update t (t=E(Ti) +R)
update t (t=t+d)
A
B
A
Rollback

Fig.2 Behavior of Analytical approach for concurrency control in mobile environments

Let $E(T_i)$ represents expected time for execution of transaction T_i. The time for execution is set based on the type of transaction or it might be evaluated based on the History state of successful transactions. Each time a transaction request is submitted to the coordinator, it evaluates the expected time for execution based on varied parameters such as history state, processing speed or the time set by the service provider.

"t' represents the time period within which the transaction is expected to complete its execution. "∂" represents the step factor with which the timer is increased for every rollback operation. "T" represents threshold value of timer within which a transaction has to complete its execution, otherwise it may be aborted.

The following scenarios are realized for the implementation of proposed analytical approach for concurrency control:

Case (i): Execution time of a transaction [E (T_i)] less than or equal to timer "t"

A transaction request initiated by mobile host is submitted to the coordinator (base station). The coordinator checks the availability of data items needed for execution of a transaction. If the required data items are not locked by any other mobile host, the expected time for execution of transaction T_i is compared with current timer "t". If $E(T_i) \leq t$, the data needed for execution of transaction is read by mobile host and the execution continues offline. After successful completion of the transaction, the results are returned to the coordinator which makes the final decision by informing about the final commit decision to DBS and mobile host respectively and the data items are unlocked. If the timer value is increased as a result of successive rollback operations, it must not cross the threshold value "T". If timer "t" crosses the threshold T, it will be aborted. A transaction proceeds for execution only if it can be executed within time period "t". This helps in increase of throughput of the system.

Case (ii): [E (T_i)] >"t" and no other transaction is requesting for same data items

If the execution time of transaction T_i is greater than timer "t", but there doesn't exist any transaction in job queue waiting for the same shared data item as requested by T_i, the transaction may continue execution after updating the timer value with required time for execution i.e. t = E (T_i), provided the timer t is less than the threshold "T". However if t>T for transactions initiated by several mobile hosts respectively, then the value of t & T has to be updated. This is done as a process of resetting the timer after analyzing the success rate of the transactions.

Case (iii): [E (T_i)] >"t" and other transaction say T_j is waiting for the data items requested by T_i.

If (E (T_i)] >"t") and another transaction say T_j is waiting for the shared data items as requested by T_i, the transaction T_i may be rolled back. However if E (T_i) is slightly more than t, then it is better that T_i may be allowed to continue its execution. In this scenario the remaining time of execution of T_i is evaluated (R= E (T_i) - t). If R is less than 25% of execution time of T_j, T_i is allowed to continue its execution by updating the timer value (t= E (T_i)). Otherwise T_i may be rolled back by increasing the timer value by "∂" factor and T_j is allowed to continue its execution. Increasing timer value t by a small factor may help the subsequent waiting transaction whose time for execution is slightly greater than t, to get executed directly. This is because the timer t is increased by "∂" factor as a result of rollback operation of previous transaction.

The advantage with this strategy is if the time for execution is slightly more than "t", it may be allowed to continue its execution; this reduces the waiting time of the transaction. Further the uplink bandwidth is also reduced because the re-execution requests for a transaction by mobile host are minimized.

C. State Transition Representation

A transaction is said to be in *Ready* state if the request for its execution is initiated and it is placed in the queue. The decision regarding the execution of transaction is done in this state. A transaction enters into an *Active* state if the time for execution is less than or equal to the timer value or remaining time for execution of current transaction is not more than 25% of the execution time of waiting transaction. Otherwise it enters into a *Pending* state as a result of rollback operation. It may enter into *Abort* state if the timer value exceeds the threshold limit. It is assumed that the transaction is executed successfully after entering into Active state. From the Active state a transaction can enter into commit state upon its successful completion.

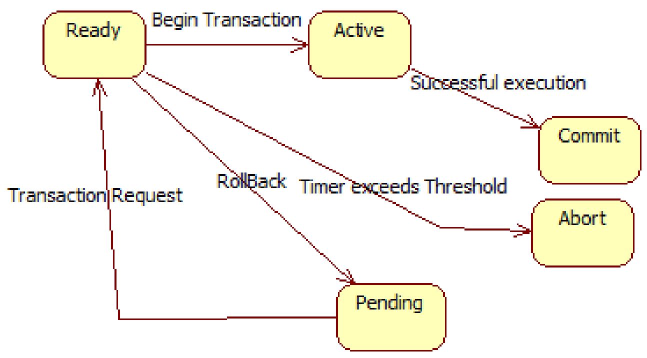

Fig.3 Transaction State diagram of Analytical approach for Concurrency Control in Mobile Environments

The state machine M for Analytical approach is represented as a pentuple

$$M = (Q, \sum, \delta, q_0, F)$$

Where Q represents set of states
$\sum$ represents the set of inputs needed for transition
δ represents the transition function
q_0 represents the initial state and
F represents the final state.

Q= {Ready, Active, Pending, Commit, Abort}
$\sum$ = {Begin Transaction (BT), Successful Execution (SE), Timer Exceeds Threshold (tET), Rollback (RB), Transaction Request (TR)}
q_0= {Ready}
F= {Commit}
The transitions of the state machines are defined as

a. δ (Ready, BT) = Active
b. δ (Active, SE) = Commit
c. δ (Ready, RB) = Pending
d. δ (Ready, tET) = Abort
e. δ (Pending, TR) = Ready

Let T_i represents the transaction initiated by the mobile host. Let "t" represents the current timer value and E (T_i) represents the expected time for execution of transaction T_i. If E(T_i) is less than t or no other transaction is requesting for data items requested by T_i or $(E(T_i) - t) \leq 0.25 * T_j$ (i.e. remaining time of execution of T_i is less than 25% of the expected time for execution of T_j), then the transaction may enter into an active state and thereby commit successfully (irrespective of failures). If E(T_i)> T (i.e. the expected time for execution of a transaction is greater than threshold value), then the transaction is aborted. Otherwise the transaction is entered into Pending state as a result of rollback operation by updating timer t to $t + \partial$. From the Pending state it moves into a Ready state.

The following cases describe the correctness of the protocol.

(i) Expected time for execution of the transaction (t_e) is less than timer t or marginally more than the timer t
δ (Ready, BT) = Active
δ (Active, SE) = Commit $\in$ F, Hence accepted

The transaction T_i changes its state from Ready to Active (Begin Transaction). If the transaction is successfully executed (SE), it enters into a commit state which is the accepted final state in the state machine M.

(ii) The expected time of execution is larger than t
[i.e. $(E(T_i) - t) > 0.25 * T_j$]. However $E(T_i) \leq T$
δ (Ready, RB) = Pending
δ (Pending, TR) = Ready

If the expected time of execution of current transaction is quiet higher than the timer t, it moves into Pending state as a result of rollback operation. The timer value is increased by $t + \partial$ (where $t + \partial \leq T$). Later it moves into ready state and may commit or abort or rollback.

(iii) The expected time for execution of the transaction is greater than the threshold value.
δ (Ready, tET) = Abort

If the time for execution of the current transaction is greater than the threshold value then the transaction enters into an Aborted state.

D. Mobility Management

A mobile host may move from once cell (C1) to another (C2) after initiating the transaction in C1. The coordinator of the transaction remains the base station in C1 where the mobile host was registered before starting the execution.

Assume that the mobile host is registered with Base station B1, which acts as the coordinator. The mobile hosts requests for a transaction T1 and starts the execution. When mobile host moves from cell C1 to cell C2, it is registered with a base station (say B2) in cell C2. The information regarding the coordinator and the transaction (Mobile host, Transaction, Coordinator) is registered with base station B2. When the transaction is successfully completed, the results are returned to the base station in cell C2. The base station B2 returns the results of the transaction to Base station B1 in cell C1, which performs the final update. Similarly when the mobile host is moved to another cell and the offline transaction is to be rolled back, the coordinator sends a message to the base station to which a mobile host is currently registered in foreign cell to roll back the transaction.

V. PERFORMANCE METRICS

The Analytical approach for achieving concurrency control in mobile environments is simulated using, postgress as the database. The front-end differs from application to application. A front-end for mobile banking with basic transactions is designed. The simulator follows MVC (Model-View- Controller) architecture. The functionality of the coordinator i.e the Base station is implemented using J2EE at fixed host. The system is finally tested using 10 different types of transactions involving concurrent requests for execution of a transaction.

The proposed strategy is suitable for M-Banking applications when the time for execution of a transaction by a particular mobile host is known in advance. The expected time for execution can be evaluated either by using the history state or based on the type of transaction. If the time for execution and the timer value is known, the decision regarding executing a transaction can be made. This helps in increasing the throughput and reducing the waiting time of a transaction in a job queue.

The relations that are maintained at the base station (coordinator) are Transaction_Info, Timer, Current_Transaction relation & Base station relation. The Transaction_Info relation describes the list of banking transactions that can be executed on a mobile.

The Timer relation specifies the time within which a mobile host is expected to return the result to the coordinator after completing the operation. Current_Transaction relation describes the list of transactions which are active and non-conflicting. The time for execution can depend on the mobile host or the type of transaction. If the time for execution is dependent on a transaction, it is maintained in Transaction_Info relation.

Table I. List of Banking Transactions

Transaction Id	Name	Relation	Data Item(s)
T1	Deposit	Account	Amount
T2	Withdrawal	Account	Amount
T3	Transfer	Account	Accountno , Amount

Table I lists the possible bank transactions. Two mobile hosts may execute the same transaction T_i if their account numbers are different. A row locking mechanism is simulated to implement the M-Banking scenario. The fixed host also maintains a Timer relation (Table II) that specifies the time(t) within which the transaction is to expected to be completed and maximum threshold value (T).

Table II. List of Transactions along with the Timer values

TransactionId	Timer Value(t)	Threshold Value(T)
T1	3msec	6msec
T2	4msec	6msec
T3	3msec	5msec

Table III lists the transaction requests by mobile hosts in order of arrival. It contains the Accountno and expected execution time which is known in advance. If the time for execution is slightly more than the timer value and no other transaction is requesting for same data items or remaining time of execution of current transaction is less than 25% of execution time of the transaction requesting for similar data items and waiting in a queue, then the transaction may proceed for execution, otherwise it is rolled back.

Table III. Job queue (M-Banking)

SiteId	Transaction request	Accountno	Execution Time
M1	T1	101	3
M2	T2	102	6
M3	T1	103	2
M4	T1	101	4
M5	T2	102	5

Transactions initiated by mobile host M1, M2 and M3 can be executed in parallel. Though M1 and M3 have requested for same transaction request but their account numbers are different. However M4 has to wait till the completion of transaction by M1. Similarly M5 has to wait for the completion of transaction M2. M2, M5 and M1, M4 might represent the scenario of joint account holders. The simulation of Analytical approach is depicted in the Table IV. The current transactions relation is maintained at the base station (coordinator) which is responsible for scheduling of the transactions.

Table IV. Current Transactions relation

Mobile Host	t	RT	Decision	$t=t+\delta$	CT*	Status
M1	3	Nil	√	3	3	Commit
M2	4	2	×	5	-	Pending
M3	3	Nil	√	3	5	Commit
M4	3	1	√	4	9	Commit
M5	5	Nil	√	5	5	Commit
M2	5	1	√	5	11	Commit
RT: Remaining Time CT: Completion time						

M1 needs 3 msec for execution of transaction T1 and the timer t is also set to 3 msecs as such the transaction moves into active state and executes successfully. M2 requires 6 msec for execution of transaction T2. However the timer value set (Table II) for execution of transaction T2 is 4 msec. The remaining time for execution is 2msec. M5 has also requested for transaction T2 with execution time 5 msec. Since 25% of 5 is 1.25 which is less than remaining time (2) of transaction initiated by M2, M2 has to be rolled back and it enters into pending state by updating the timer t.

M3 requires 2 msec for execution of transaction T1 and the timer value is 3 msec. It moves into an active state and commits. M4 has to execute the transaction T1 only after M1 completes the transaction. M4 needs 4 msec to execute the transaction T1. The timer value is 3msec, however there is no other mobile host requesting for transaction T1.

Hence M4 moves into active state, and the completion time is 9 msec because M4 may start its execution only after completion of M1 (at 5 msec). Similarly M5 commits then again M2 gets a chance and there is no waiting transaction for T2. Hence M2 also executes successfully. The advantage with this approach is the system throughput is high because the transaction enters into an active state only if sufficient time for execution exists. Otherwise the waiting transaction is executed. Further the average waiting time in the transaction job queue also decreases as compared to the strategies proposed in the literature.

The Analytical approach for guaranteeing concurrency control in mobile environments is compared with the traditional timeout mechanism and TCOT [2].

Fig. 4 specifies the comparison of commit rate of proposed strategy with the static timeout based protocols.

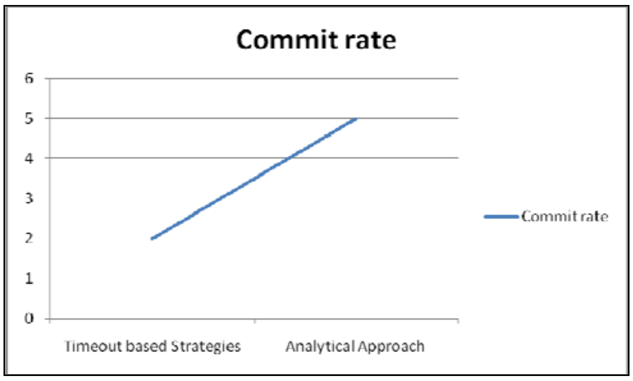

Fig.4 Comparison of Static timeout protocols with proposed strategy with respect to the commit rate.

Though the commit rate of the proposed strategy is similar to the dynamic timer adjustment strategies [2], however in the dynamic timer adjustment strategies this commit rate is achieved after a series of rollback operations.

Fig.5 depicts the comparison rate of rollback operations of proposed strategy with Dynamic timer adjustment strategy. The rollback operation not only increases the waiting time for each transaction but it also decreases the performance of the system.

Further the expected time for execution of a transaction may be updated from time to time. For example when most of the transaction requests are not able to complete the transaction within the specified time, it can be increased so that the subsequent request may result in higher commit rate. This also reduces the time taken for computation.

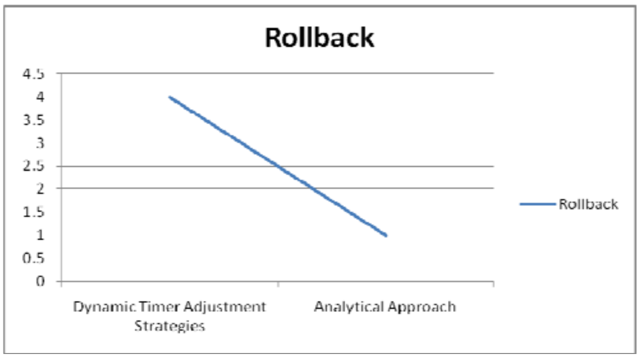

Fig.5 Comparison of dynamic timer adjustment strategies with the proposed strategies

VI. Conclusion

A transaction throughput would increase if the possible time of its execution is compared as against the current timer value. If the current timer value is very small as compared to the time of execution of a transaction it may be executed later. This increases the throughput and increase the success rate of the transaction because the transaction can proceed for execution only if sufficient time is available. The average waiting time for each transaction decreases because even if the expected time for execution is slightly greater than the timer value, it gets executed. In the Analytical approach for most of the time the transaction is not aborted after executing it for certain period of time. This forms a major factor for mobile applications because a mobile user must know instantly whether its request can be forwarded or the transaction has to be initiated later.

References

[1] Ho Chin Choi, Byeong-Soo Jeong, "*A Time stamp Based optimistic Concurrency Control for handling Mobile Transactions*", ICCSA 2006, LNCS Vol. 3981, PP. 796-805, Springer Heidelberg, 2006
[2] Kumar V., Nitin Prabhu, Maggie Dunham, Ayse Yasemin Seydim, "TCOT, A Timeout based Mobile Transaction Commitment Protocol", In:IIS9979453(2004).
[3] Nadia Nouali, Anne Doucet, Habiba Drias, "A Two Phase Commit Protocol for Mobile Wirelss Environment" 2005 , 16th Australasian Database Conference Vol.39 (ADC 2005)
[4] Patrica Serran-Alvarado et Al, "A Survey of Mobile Transactions" Distributed & Parallel Databases, Kluwer publishers, 16, 193-230, 2004.
[5] P. Krishna Reddy, Masaru Kitsuregawa, "*Speculative Lock Management to Increase Concurrency in Mobile Environments*", MDA'99, LNCS 1748, pp. 82- 96, 1999.
[6] Victor C.S., Kwok wa Lam and Son, S.H., "*Concurrency Control Using Time-stamp Ordering in Broadcast Environments*", theComputer Journal, Vol.45, No.4 PP.410-422, 2002
[7] Minsoo Lee, Sumi Helal, "*HiCoMo: High Commit Mobile Transactions*", Distributed and Parallel Databases, 11, 73-92, 2002, Kluwer Academic Publishers
[8] Anand Yendluri, Wen-Chi Hou, and Chih-Fang Wang, "*Improving Concurrency Control in Mobile Databases*", Springer Verlag LNCS 2973, PP. 642-655, 2004.
[9] Victor C.S. Lee, Kwok Wa Lam, Tei-wei Kuo,"*Efficient Validation of Mobile Transactions in Wireless Environments*", The Journal of Systems and Software 69(2004), 183-193.
[10] Stankovic, J.A; Zhao, W, "*On real time transactions*", SIGMOD Record 17(1), pp. 14-18, 1988
[11] Stankovic, J.A; Son, S.H; Hansson, J; "*Misconceptions about real time Databases*", Computer 32(6), pp. 29-37, 1999.
[12] Marco Ballette, Antonio Liotta, Samir M. Ramzy, "*Execution Time Time Prediction in DSM based Mobile Grids*", IEEE International Symposium on Cluster Computing & the Grid, pp. 881-888, 2005.