Volume: 6 Issue: I Month of publication: January 2018

DOI: http://doi.org/10.22214/ijraset.2018.1060

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor :6.887
Volume 6 Issue I, January 2018- Available at www.ijraset.com

A Strategy to prioritize the waiting transactions to achieve Concurrency Control in Mobile Environments

Salman Abdul Moiz[1], Yeshwant Sai Kandubothu[2]
[1, 2] School of Computer & Information Sciences, University of Hyderabad

Abstract: The inherent features of mobile environments makes concurrency control in mobile environments a challenging problem. Pessimistic concurrency control mechanisms may not be suitable in mobile database environments as the wireless devices are prone to disconnections and mobility. The timeout based pessimistic concurrency control mechanisms results in higher number of rollbacks and starved transactions. The transactions waiting for shared data items in a job queue may be starved as it may not be able to complete its execution within the given time period. The transactions are usually scheduled on First Come First Serve Basis. This will result in either transaction being aborted or ending up waiting for longer period of time in job queue. In this paper a concurrency control mechanisms is proposed which predicts the expected time for execution of transaction and can proceed for transaction execution based on the desired timer value. Further it prioritizes the transactions waiting in a queue based on Transaction Response Ratio (TRR). The proposed algorithm is tested for concurrent access having 12 attributes and 900 tuples. The simulation results shows considerable decrease in waiting time of transaction and increase in commit rate.
***Keywords:** Mobile databases, Concurrency control, Transaction response ratio, waiting time, predicted execution time*

I. INTRODUCTION

The exponential increase in use of data services and applications on ubiquitous devices may lead to anomalies when several devices request for shared data item at the same time. This may make mobile database system inconsistent. When a portable device initiates a database transaction with an intention of updating the database, it has to lock the shared data items in pessimistic mode. These data items are locked on fixed host and the data needed for executing the transaction is read by the ubiquitous device locally. This will help the mobile device to execute the transaction locally. Even though there is a disconnection with the server, the transaction can continue to execute locally. Upon successful transaction execution and reconnection, the results will be updated on fixed host or server. However the devices are disconnected for longer time, the transactions requested my other devices may be starved as the data items needed to complete the transaction is locked. The timeout based mechanisms are used to deal with starvation of transactions in job queue. If the device executing the transaction is not able to return the result of transaction within certain time period it gets aborted. This reduces starvation of transaction but increases abort rate. Later the dynamic timer strategies helps in reducing the abort rate but the number of rollback operations increased. The frequent rollback anomaly was addressed by estimating the time for execution of transaction based on historical data of similar transactions executed by compatible devices. In this paper a concurrency control methodology suitable for mobile database environments is proposed. The proposed algorithm predicts the time for execution of a transaction. This helps in making a decision whether to continue the execution of the transaction of the expected time for execution is greater the desired timer value. The proposed algorithms is designed to reduce the waiting time of the transaction by scheduling it for execution. The transactions waiting in queue can be given preference based on their waiting time. The transaction response ratio (TRR) is computed to decide upon the next transaction to be schedule for execution among the waiting transactions. The transaction response ratio is calculated by predicting the execution time for each transaction. The results shows decrease in waiting time of transactions and also helps in scheduling the transactions which otherwise were waiting for longer period of time. The remaining part of this paper is organized as follows: Section –II specifies the optimistic concurrency control mechanisms available in literature. Section –III describes the mobile database architecture. Further concurrency control strategy by predicting the time for execution of a transaction is proposed. The mechanism to reduce waiting time of the transaction is also proposed. Section –IV specifies the results of the proposed strategy and section-V concludes the paper.

II. RELATED WORK

When multiple mobile hosts access the same date items simultaneously then it may lead to data inconsistency. One of the primary problems in mobile data base environment is Data Conflict anomaly [1]. Optimistic concurrency control techniques are proposed to resolve the data conflict in transaction validation phase. However it may lead to delay in responding to the request. Two phase locking protocol (2PL) [2] guarantees serializability in transaction processing but it requires mobile host to remain continuously connected with server. DHP-2PL [3] is a locking protocol based on high priority two phase locking to achieve concurrency control in mobile environments. However in DHP-2PL the lower priority transaction may restart again and again due to several restarts of lower priority transactions. Transaction Commit on Timeout Protocol (TCOT) [4] works on timeout principle. The basic idea of Single Phase Reliable Timeout Based Commit Protocol (SPRTBCP) [5] is to remove voting phase of 2PC. This is achieved by introducing the local databases. As the runtime of a transaction is not known, setting the exact timer within which that transaction is expected to complete its execution is a difficult task. Hence, in the proposed strategy an attempt is made to predict the run time of each transaction to accordingly set the timer and then prioritize the transactions.

III.PRIORITIZING THE MOBILE TRANSACTION

A. Mobile Database Environment

In mobile database environment, the request for a transaction for an application is initiated at any of the wireless or ubiquitous device. The transactions may be totally executed at server or fixed host or it may be executed on the client. The execution of transaction can also be shared between ubiquitous devices and the server respectively. The ubiquitous or mobile devices can vary from laptop to any smart device.

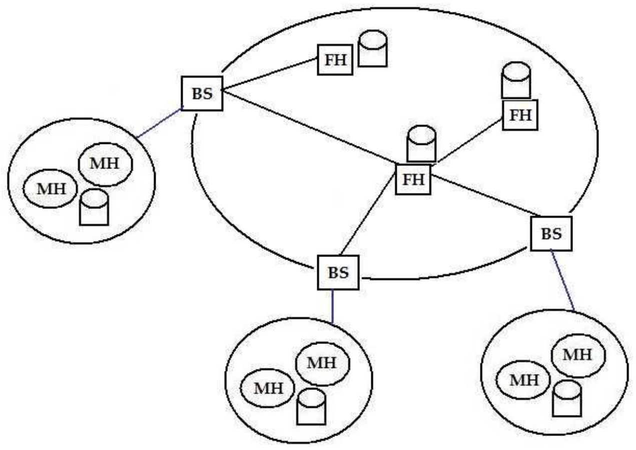

Fig. 1 Mobile Database Architecture [6] [7]

The Figure.1 describes the generic architecture of the mobile database system. It is based on the three tier architecture. The first element is the mobile client or mobile hosts which can move from one place to another. Fixed hosts are stationary servers which can store the database. The base station acts as an interface between mobile host and fixed host respectively. The base station functionality can be realized by mobile middleware. Mobile host is connected to base station through a wireless link and the base station is connected to a fixed host in a wired mode.

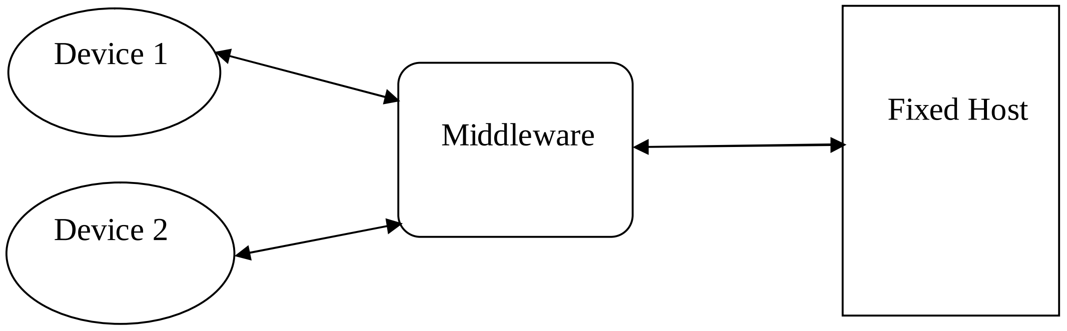

Fig. 2 Ubiquitous Middleware Architecture

Mobile or ubiquitous middleware acts as an interface between ubiquitous devices and fixed host respectively. There may be various databases for several applications on fixed host. Hence middleware is needed to provide support to heterogeneous databases and applications. The request for execution of transaction is initiated at ubiquitous devices. The transaction request is sent to the ubiquitous middleware. The middleware maintains the job queue as the concurrent transactions are executed based on pessimistic strategy. The requested data items are locked at server or fixed host and is read by devices with the help of message oriented middleware. The devices execute these transactions locally. Even if the device is disconnected, the transaction is executed locally. Once the transaction completes its execution successfully the results are sent back to middleware which updates the data items at fixed host and schedules any waiting transaction in the job queue.

B. Prioritizing Transactions

When multiple transaction request from devices request to access same shared data items to the middleware, it may lead to conflict. The pessimistic concurrency approach allows one device to access the data items needed to execute the transaction. The other devices which requested for same shared data item has to wait till the later transaction completes.. The waiting transactions can be prioritized based on Transaction Response Ratio (TRR) which is calculated using waiting time and estimated execution time of the transactions. The proposed strategy is implemented using the three tier model depicted in the figure 2. Whenever a data item is requested by a transaction then that data item is locked by the middleware and the transaction is executed locally. As the data item is locked, other transactions which are trying to access the same data item are kept waiting. By calculating the transaction response ratio for each transaction in each iteration the highest TRR gained transaction is executed next. When the current transaction completes its execution the data item is unlocked and given access to the other transaction (if any) having the highest transaction response ratio (TRR).

C. Waiting Time and Execution Time

Given a transaction set with arrival times (A_t)(msec) of each transaction and the times at which the transaction starts its execution (S_t)(msec), waiting times can be calculated. The Schedule time (S_t)(msec) of a transaction is the time at which the transaction starts it's execution. The waiting times of the transaction can be calculated as:

$$\text{Waiting Time (Wt)} = \text{Scheduled Time } (S_t) - \text{Arrival Time } (A_t)$$

Mobile hosts may have different configuration such as processing speeds, cores etc… The similar transaction request from mobiles with similar configurations form a cluster because the time taken to execute the transaction locally varies from one device to another. After execution of the transaction, the actual time taken to execute the transaction is included to the cluster which makes values more accurate for further iterations. The transaction execution time is predicted based on the following expression [8]:

$$Tn+1 = a * Tn + (1-a) * tn$$

Where,

T_n+1 = Run time of the next transaction
T_n = Average of all run times within a chosen cluster
a = Step factor ($0 < a < 1$)
t_n =Most recent successfully executed transaction run time

The Step factor (a) is the percentage of weightage that is given to average of all run times within the chosen cluster (T_n). The value of 'a' should be $0 < a < 1$. t_n is the most recent successfully executed transaction run time within the chosen cluster. Considering 'a' to be 50% for T_n also makes 50% to tn. Basic idea is to have preference to cluster's average (T_n). But problem occurs if the cluster is not been used for a longer periods of time. If a transaction initiated from a mobile belonging to a cluster which is not used for long period then computation of predicted run time of that transaction may not yield accurate results. As the step factor (a) is varied (decreasing) then the predicted runtime of the transaction is deviating more from the cluster's average (T_n) which is not effective. Now the step factor (a) is varied in increasing manner. The predicted runtimes are moving closer to T_n as the value of 'a' is increased. These results might be accurate as the values of predicted execution times move closer to clusters average. But the case when a cluster is not been used for longer period of time then preferring only cluster value is not effective. Hence, most recent successfully executed transaction (t_n) is considered. Likewise, it is not effective to give more weightage to 't_n' than 'T_n'. Hence, the optimal value of step factor(a) is considered as 0.5 giving equal weightage to both 'T_n' and 't_n'.

Using the waiting time and the expected time for execution of a transaction, transaction response ratio (TRR) is computed. If there more than one mobile transactions waiting in the queue. Then transaction response ratio (TRR) is computed for the transactions present in the queue and the transaction with highest TRR has to be scheduled next. The transaction response ratio is computed as follows [9]:

$$\text{Transaction Response Ratio (TRR)} = (W_t + T_e) / T_e$$

Where,
W_t = Waiting time of a transaction in waiting queue
T_e = Expected time of execution

D. Concurrency Control Algorithms

Fig. 3 specifies the workflow of the concurrency control strategy. The ubiquitous devices can send the request for execution of a transaction to the middleware. The middleware will check the availability of data item. If the data item is available it will be locked at fixed host and the data items needed to execute the transactions are sent to the device for local execution. If the data items are not available then the transaction is logged in the job queue at middleware.

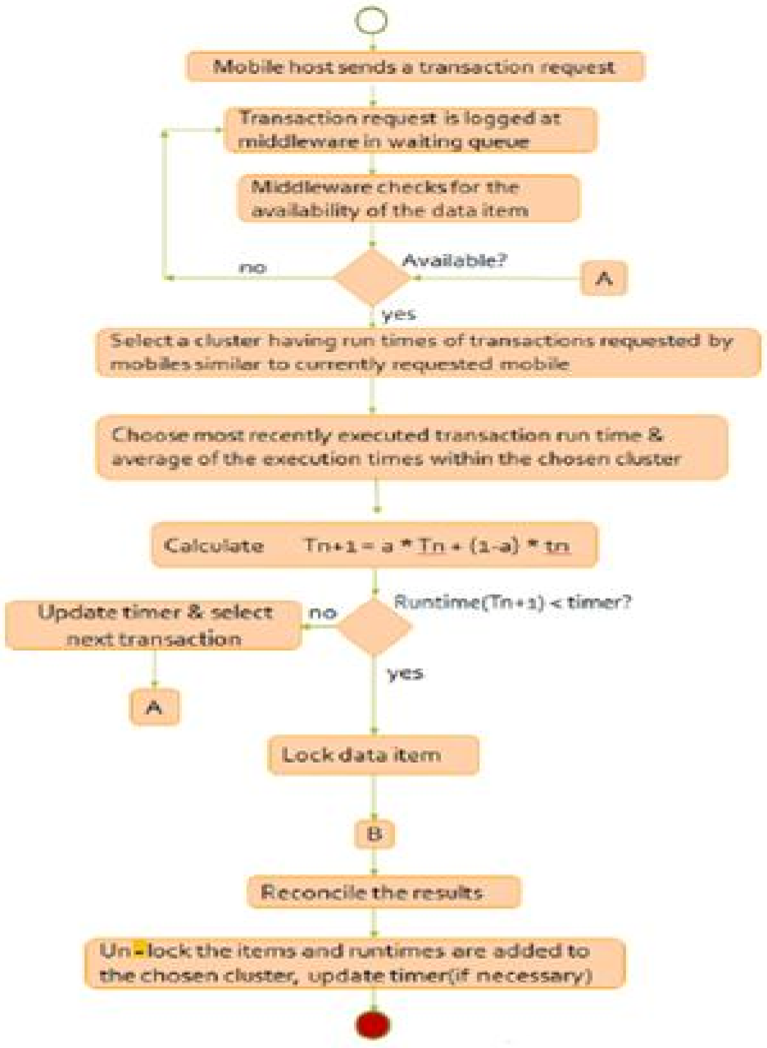
Mobile host sends a transaction request
Transaction request is logged at middleware in waiting queue
Middleware checks for the availability of the data item
Available?
no
A
yes
Select a cluster having run times of transactions requested by mobiles similar to currently requested mobile
Choose most recently executed transaction run time & average of the execution times within the chosen cluster
Calculate Tn+1 = a * Tn + (1-a) * tn
Runtime(Tn+1) < timer?
Update timer & select next transaction
no
A
yes
Lock data item
B
Reconcile the results
Un-lock the items and runtimes are added to the chosen cluster, update timer(if necessary)

Fig. 3 Concurrency control strategy in mobile database environment

Once the data items required for the transaction execution is locked, the middleware computes the expected time for execution of the transaction. The expected time for execution of transaction initially considers the average execution time(Tn) taken by same transaction or similar types of devices. Since the transaction may not have been initiated by particular type of device, it is assumed that the expected time for execution may not be accurate. Hence the time taken by recently executed transaction belonging to same cluster is selected (tn). The expected time for execution of current transaction (Tn+1) is computed by selecting the value of ‘a’

which varies from one service to another. The value of ‘a’ determines as to how much weight is to be given to tn and Tn respectively.
To avoid starvation of transaction, each transaction is expected to complete its execution within certain period of time (timer). The timer value is decided based on type of services and will be decided by the service provider. If the expected time for execution of transaction is less than the timer value, the execution of transaction begins. Once the transaction is successfully completed, the results are updated on the fixed host and the actual time taken to execute the transaction is added to the cluster to evaluate the new expected time for execution of the transaction.
If the expected time for execution of transaction is greater than the timer value, the transaction still proceeds its execution if there are no other jobs in waiting queue. However if there are any transactions waiting in the job queue then it is assumed that the transaction may not complete the execution even in future. Hence the timer value is increased by a small factor (∂) thereby the timer value is updated and the next waiting transaction is selected.
The next hypothesis is to be reduce the waiting time of transaction in job queue. The transaction response ratio is computed to identify the highest priority transaction.

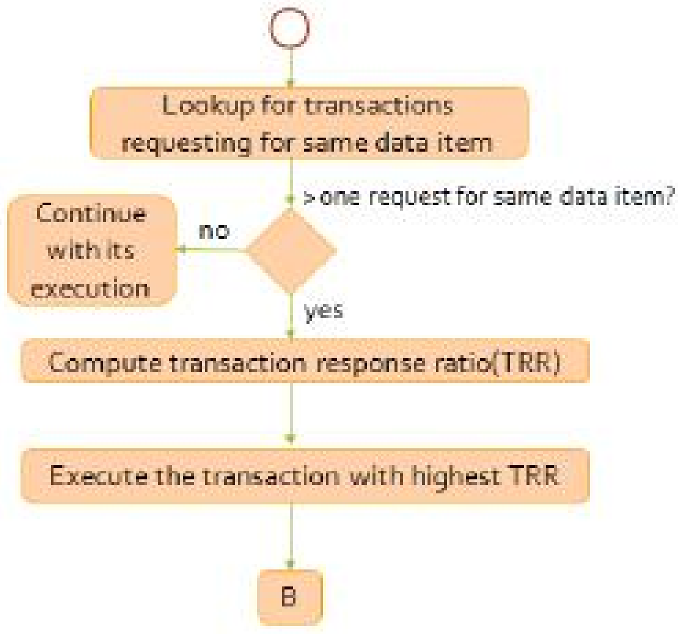
Lookup for transactions requesting for same data item
> one request for same data item?
no
Continue with its execution
yes
Compute transaction response ratio(TRR)
Execute the transaction with highest TRR
B

Fig. 4 Management of waiting transactions

Once a transaction is successfully executed or it is rolledback, the a lookup operation is performed on waiting queue to schedule the next transaction. If there are transactions requesting for same shared data item, then the transaction response ration is computed for each transaction. The transaction with highest transaction response ration is scheduled next for execution. After successful execution of transaction, the state of shared data item is updated on the fixed host.

IV. RESULTS AND ANALYSIS

The proposed strategy is simulated by realizing a middleware which is connected to fixed host. The application to initiate transactions were uploaded on several ubiquitous devices to simulate concurrent execution of transaction.

A. Results of a scenario

In this section the algorithm is simulated with a sample scenario. The next part of this section shows the simulated results on few datasets. Consider the scenario in Table I, where six hosts are requesting for same shared data item(s) for execution of transaction initiated by them. It is also assumed that each of these devices are of different configurations. Hence the expected time of execution for same transaction varies from one device to the other. Since transactions are scheduled one after the other, their waiting time can be computed accordingly.

TABLE I
TRANSACTIONS INITIATED BY SEVERAL DEVICES

Device	Arrival Time (AT)	Expected Execution Time (Te)	Waiting Time (WT)
D1	0	3	0
D2	2	4	1
D3	1	3	8
D4	7	11	3
D5	9	3	12
D6	9	4	15

D1 request for execution of transaction and enters in execution state. When D1 is under execution D2 and D3 arrives. D2 has to wait till D1 completes its execution and D3 need to wait till D1 and D2 successfully completes its execution. The expected execution time (Te) is evaluated as the average time taken to execute the given transaction by similar configuration devices. In Table I D1, D3 and D5 forms a cluster as they have similar configuration. When D1 is executed, it takes 4 Secs to execute (say), the expected time for D3 can be executed using the expression Tn+1 = a * Tn + (1-a) * tn
Assuming that the value of a=0.5, the time for execution of D3 can be computed as 3.5 msecs. (0.5*3+0.5*4). Similarly the time for each execution is computed to check whether it is less than the desired timer value. In the above scenario it is expected that the device has to return the value to fixed host in 4msec. Hence the transaction initiated by D1 gets executed.
Once the transaction complete its execution, the next transaction is to be scheduled. The decision about the next transaction depends on the transaction response ratio (TRR).
Consider the scenario in Table 1. When D1 arrives, there are no other transaction waiting in job queue. D1 continues the execution of transaction for 3msec. By the time D1 commits, two transactions initiated by D2 and D3 are waiting in job queue. The decision has to be taken as to which job is to be scheduled. The transaction response ration of D2 and D3 is then computed. Transaction Response Ratio (TRR) = $(W_t + T_e) / T_e$
TRR of D2 is 1.25 (1+4/4) and that of D3 is 1.67(2+3/3). Since the TRR of D3 is higher it is scheduled for execution. D3 completes its execution in 7th msec, the job queue now contains D2 and D4. The TRR is computed for these two devices to make a decision as to which job is to be scheduled.

B. Performance Characteristics

The proposed strategy is simulated using a data sets [10] each having twelve attributes and tuples above nine hundred. The results shows that average waiting time of the transaction set is reduced compared to FCFS. The commit rate is increased and the abort rate reduced compared to 2 Phase Locking (2PL) protocol. In pessimistic concurrency control mechanism, the transaction requesting shared data items are usually placed in job queue and are scheduled on first cum first serve basis (FCFS). In the proposed approach the transaction is scheduled based on the Transaction Response Ratio (TRR) which is calculated based expected time for execution an waiting time of the transactions. Figure 5 shows the graph plotted with average waiting time of the transaction sets that are tested on proposed algorithm and the scheduling of transaction is compared with the first cum first serve (FCFS) approach. Several group of transaction sets requesting for share data items are simulated for average waiting times. The average waiting time in each of the scenarios are compared. It is observed that in the proposed strategy the waiting time of the transaction has reduced. The proposed strategy not only reduces the waiting time but also increases the throughput of the system.

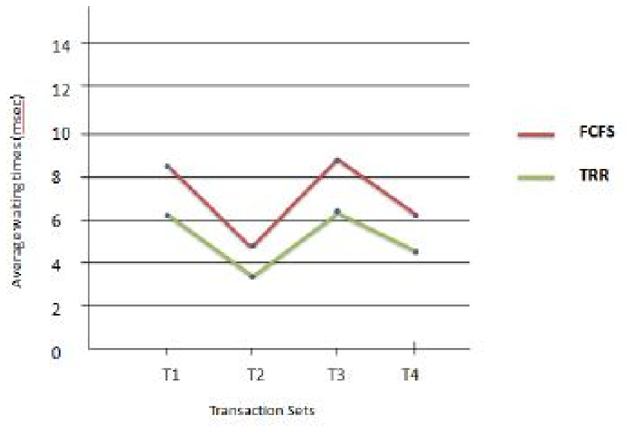

Fig. 5 FCFS vs Proposed strategy(Average waiting times)

In the proposed strategy, if the transaction timeouts then transaction is rolled back and added back to the waiting queue. The timer is then increased by a value of step factor (a) (which is equal to 0.5msec). For each transaction, if it timeouts then the timer is increased by the step factor (a) only twice. Even if the transaction timeouts after incrementing the timer twice, then the transaction will be aborted and has to be re-initiated. Whereas, in 2PL if the transactions timeouts then it is aborted and it has to be re-initiated the very first time.

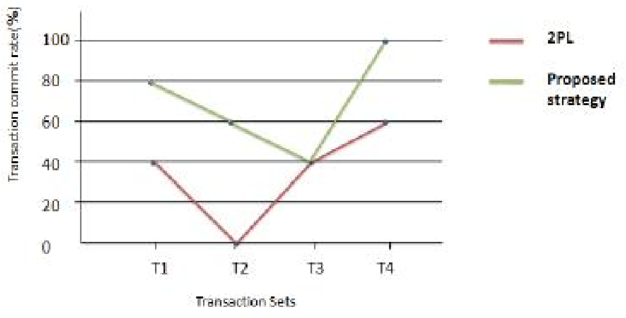

Fig. 6 2PL vs Proposed strategy(Commit rate)

Figure 6 is the graph plotted for the comparison between 2PL protocol and proposed strategy. X-axis consists of transaction sets and Y-axis consists of transaction commit rate in percentages (%). It is also observed that the commit rate in the proposed strategy is higher as compared to the traditional two phase locking protocol.

V. CONCLUSIONS

The proposed concurrency controls strategy is efficient as it can take a decision whether the transaction can proceed for execution in the initial stage. This is possible as the expected time for execution of transaction is evaluated before initiating the transaction execution. Further in order to reduce the waiting time of transactions in job queue, the transaction response ration is considered which helps in deciding which transaction to be executed next. This has considerably increased the throughput of the system.
In future more and more clusters can be formed, the challenge is to maintain the clusters and update the repository of transaction logs regularly. Further if a transaction of a particular was not scheduled for a longer period of time, its expected time for execution may not be accurate. Further this may change dynamically as there is a revolution in the hardware configuration of these devices. The proposed mechanism may also be extended to Internet of things with the ability to process transactions locally on each device.

REFERENCES

[1] Victor C.s Lee et al., Efficient Validation of Mobile Transactions in Wireless Environments, Journal of System Software 69(1-2):183-193, 2004.
[2] Agil A Alivey, “ Two Phase Locking Protocol in Distributed Database”, Proceedings of IMM Of NAS of Azerbaijan, pp.251-254, 2003
[3] Kam Yiu Lam et al, “Concurrency Control in Real Time Database Systems”, Information Systems, vol. 25, No. 4, pp. 261-286, 2000.
[4] Vijay Kumar et. Al, “A Timeout based Mobile Transaction Commitment Protocol”, IEEE Transaction of Computers, Vol.51, No.10, 2002.
[5] Bharati Harsoor and S Ramachandram. “Single Phase Reliable Timeout Based Commit Protocol”. GSTF Journal on Computing (JoC), 1(4), 2014.

[6] Patrica Serranao Alvarado et al. "A Survey of Mobile Transactions", Distributed & Parallel Database, 16, 193-230, Kluwer Academic Publishers, 2004.

[7] Salman Abdul Moiz et. Al, "Concurrency Control in Mobile Environments: Issues & Challenges", International Journal of Database Management System, Vol.3, No.4, pp.147-159, 2011.

[8] Abraham Silberschatz, Peter B Galvin, Greg Gagne, and A Silberschatz. Operating system concepts, 4. Addison-Wesley Reading, 1998.

[9] Rakesh Kumar Sanodiya, Sanjeev Sharma, and Varsha Sharma. "A Highest Response Ratio Next (HRRN) Algorithm Based Load Balancing Policy For Cloud Computing".

[10] Rebeccabilbro.Titanic.http://github.com/rebeccabilbro/titanic/tree/master/data, 2015.