Concurrency Control in Mobile Database Systems

Nitin Prabhu, Vijay Kumar
SICE, Computer Networking
University of Missouri Kansas City, MO 64110
npp21c(kumarv)@umkc.edu

Indrakshi Ray
Computer Science,
Colorado State University
Fort Collins CO 80523-1873
iray@cs.colostate.edu

Gi-Chul Yang
Division of Information Engineering,
Mokpo National University,
61Dorim-ri,Chungkye-myun, Muan-gun,Chonnam, 534-729,Korea

Abstract

In this paper, we propose an Concurrency Control Mechanism (CCM) for Mobile Database Systems (MDS) that ensures epsilon serializability and report its performance.

1. Introduction

The ability for users to perform transactional activities, such as, pay bills, transfer funds, etc., from anywhere and anytime is highly desirable. To meet these objectives, we need to develop transaction processing capabilities in MDS [1]. It consists of a number of mobile unit (MU) base station (BS), database servers (DBS), fixed hosts, etc.), which are connected through wireless and wired networks [1]. CCMs proposed earlier [8, 11, 12, 13] did not emphasize conservation of resources and some of them are message intensive. We argue in this paper that a weaker form of consistency is desirable and present a CCM which uses semantic properties [11] of broadcast data and report its performance.

2. Our Approach

In data processing discipline a weaker correctness criteria is acceptable in a number of situations and use ε-*serializability* (ESR) [9, 10], which tolerates a limited amount of inconsistency specified by ε to develop our CCM. It is based on a two-tier replication scheme [7] that produces ε serializable schedule.

The basic idea of two-tier replication is first to allow users to run transactions on the MU which makes data updates locally and when the MU connects to servers, then these transactions are re-executed at servers as *base transactions (BT)*. *BTs* are serialized on the master copy of the data and MUs are informed about the failed *BTs*. But the problem with this approach is that the MU executes transaction without the knowledge of what other transactions are doing, which can lead to a large number of rejected transactions [8]. Another problem is that the commit time of a transaction at MU is large because it knows its outcome only after the BT has been executed and the result is reported back to the MU. In this work we modify two-tier replication scheme to reduce the number of rejections and transactions commit time at MU.

3. The Concurrency Control Mechanism (CCM)

We assume that there is a central server that holds and manages the database. Let D be a set of data items such that $D = \{D_i\}$, where $i \in N$, set of natural numbers and $D_i \in S$ where S is a metric space. Let d_i be the current value of D_i. The data items are replicated at the MU's and let n_i be the number of replicas of D_i in MDS. We keep a limit Λ for the amount of change that can occur on the replica at each MU, thus Λ_i denotes the total maximum change allowed in a replica of D_i at a MU. If the transaction changes the data value by at most Λ_i in a MU, then they are free to commit; they do not have to wait for results of the BT at DBS. This reduces transactions' commit time and also helps to reduce the number of rejections, which could arise due to the BT not being able to commit. To control the validity of Λ_i, we define a timeout parameter whose value indicates a duration within which the value of Λ_i is valid. Timeout values of the data item should be some multiple (I) of broadcast cycle time (T). *I* depends on the frequency of incoming updates for the data item and also *I* should be sufficiently large so that the MU's can send their updates within duration $I \times T$. The server will not update the value of the data until $I \times T$ has elapsed. We assume that the MU's considers the uplink time and send their updates before the timeout expires at the server. The client can disconnect from MDS during the timeout period and can perform updates. If the client disconnects for a period longer than the timeout, then when it reconnects it should read the new values of Λ. If the updates are within the new limit set by Λ then the MU can send the updates to the server otherwise the MU will have to block some transactions so that total updates are within Λ. The blocked transactions will have to wait until the new values of Λ arrive at the MU. We describe the steps of the algorithm as follows:

At the DBS:

1. Λ_i is calculated for each data object D_i using the function $\Lambda_i = f_i(d_i, n_i)$. For each D_i there is a function f_i (d_i, n_i) associated with it and the function f_i (d_i, n_i) depends on the application semantics.
2. A timeout value τ is linked with Λ_i of the data item.
3. DBS broadcasts (d_i, Λ_i) for each data item and a τ for these values at the beginning of the broadcast cycle.
4. The DBS can receive either pre-committed transactions (transactions which have made updates to the replicas on the MU and committed) or it can receive request

Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA'04)
0-7695-2051-0/04 $ 20.00 © 2004 IEEE

transactions (transactions directly sent to the DBS by the MU). A request transaction (a transaction violating the limit is sent to the DBS as *request* transaction for execution on the master database) is not executed at an MU, as the transaction would have changed the value of replica D_i by more than Λ_i at the MU.

5. Execution of transaction on master copy of the data object: (a) DBS serializes the pre-committed transactions on the master copy according to their order of arrival on the DBS and (b) After τ expires, DBS executes request transaction and reports to the MU whether the transaction was committed or aborted.

After τ expires the DBS repeats procedure from 1.

At MU:

1. MU has (d_i, Λ_i) for every D_i it has cached and their τs.
2. MU executes transaction t_i: Let $\Lambda_{i\text{-}ti}$ be the change made by t_i to D_i. Let $\Lambda_{i\text{-}c}$ be the current value of the total change in D_i since the last broadcast of value Λ_i.
3. If transaction t_i changes the value of D_i by an amount $\Lambda_{i\text{-}ti}$, then the amount $\Lambda_{i\text{-}ti}$ is added to $\Lambda_{i\text{-}c}$. The following cases are possible:
   - **a.** If $\Lambda_{i\text{-}ti} <= \Lambda_i$ and $\Lambda_{i\text{-}c} <= \Lambda_i$ then the transaction t_i is committed at MU and it is sent to the server for re-execution as a BT on the master copy.
   - **b.** If $\Lambda_{i\text{-}ti} <= \Lambda_i$ and $\Lambda_{i\text{-}c} > \Lambda_i$ then the transaction t_i is blocked at MU until new set of (d_i, Λ_i) is broadcasted by the server.
   - **c.** If $\Lambda_{i\text{-}ti} > \Lambda_i$ then the transaction t_i is blocked at MU and submitted to the server as a *request* transaction.

Relationship with ESR: Mechanism for maintaining ESR consists of *divergence control* (DC) and *consistency* restoration. A transaction imports inconsistency by reading uncommitted data of other transactions. A transaction exports inconsistency by allowing other transaction to read its uncommitted data. Transactions have import counter and export counter. The following example shows how these counters are maintained.

t_1: w1(x), w1(z);t_2: r2(x), r2(y);t_3: w3(y), r3(z);

Schedule: w1(x), r2(x), r2(y), w3(y), r3(z), w1(z);

In the above example t_2 reads from t_1. So it is counted as t_1 exporting one conflict to t_2 and t_2 importing one conflict from t_1. So export and import counter of t_1 are incremented by one. DC sets limit on conflicts by using import limit and export limit for each transaction. Thus, update transactions have export limit and read-only transactions have import limit, which specify the maximum number of conflicts they can be involved in. When import limit > 0 and export limit > 0, then successive transactions may introduce unbounded inconsistency. For example, t1 may change the value of data by large amount and t2 will read this value and operate on it as import and export counter are not violated and later if t1 aborts t2 will have operated on a value that deviated from consistent value by a large amount. This situation requires consistency restoration, which is achieved by consistency restoration algorithms. In our algorithm DC sets limits on the change allowed in each data item value at MU and does not allow transactions to violate this limit. If it does, then it sends a request transaction to DBS for execution. A transaction at MU will see inconsistent value of data item for a maximum period of τ after which the MU receives new consistent values. During τ, the value of data item d_i may diverge from the consistent value by a maximum amount $N_i*\Lambda_i$, where N_i is the number of replica of data item d_i. In this way transactions are allowed to execute on inconsistent data item but the inconsistency in data value is bounded by $N*\Lambda_i$. So DC includes $f_i(d_i, n_i)$, which calculates Λ_i for each d_i and also for the algorithm executed at MU to process transactions. Thus, our consistency restoration includes the execution of request and pre-committed transactions at DBS and broadcasting of the consistent value of the data item to the MU.

Example using CCM proposed: We illustrate how our scheme allows bounded inconsistency and restores consistency at intervals defined by τ. Let us consider a data object X representing total number of movie tickets. X belongs to the metric state space. Let N_x be the number of replicas of X. Initially X = 180 and $N_x = 3$. X is replicated at MU_1, MU_2 and MU_3. In this example the function $f_x(X, N_x)$ that calculates Λ_x is $\Lambda_x = f_x(X, N_x) = (X/2)/N_x = X/2N_x = 30$. Note that we divide X by 2 so that we keep some tickets for the request transaction, which cannot be executed at the MU. This function depends on the application semantics and the policy the application developer wants to follow. Each data item will have different function depending on its semantics. (Λ_x, X, τ), where τ is timeout within which the MU should send committed transaction for re-execution at the server, is broadcasted by the DBS to MU's.

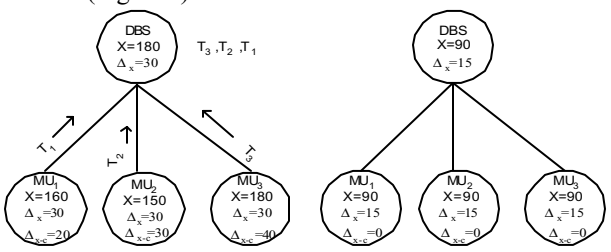
Case 1: t_1, t_2, t_3 arrive at MU_1, MU_2 and MU_3 respectively. Suppose t_1 books 20 tickets, t_2 books 30 tickets and t_3 books 40 tickets (Figure 1a). Let $\Lambda_{x\text{-}c}$ represent change in value of data object X. Each MU that has replica of data object X will maintain value of $\Lambda_{x\text{-}c}$.

At MU_1: Initially $\Lambda_{x\text{-}c} = 0$. t_1 books 20 tickets, so $\Lambda_{x\text{-}t1} = 20$ and $\Lambda_{x\text{-}c} = \Lambda_{x\text{-}c} + \Lambda_{x\text{-}t1} = 20$. As $\Lambda_{x\text{-}c} < \Lambda_x$, t_1 is committed at MU_1, so X is updated to 160 and t_1 is send to DBS for re-execution on the master copy.

At MU_2: Initially $\Lambda_{x\text{-}c} = 0$. t_2 books 30 tickets, so $\Lambda_{x\text{-}t2} = 30$ and $\Lambda_{x\text{-}c} = \Lambda_{x\text{-}c} + \Lambda_{x\text{-}t2} = 30$. As $\Lambda_{x\text{-}c} < \Lambda_x$, t_2 is committed at MU_2, so X is updated to 150 and t_2 is send to DBS for re-execution on the master copy.

At MU_3: Initially $\Lambda_{x\text{-}c} = 0$. t_3 books 40 tickets, so $\Lambda_{x\text{-}t3} = 30$ and $\Lambda_{x\text{-}c} = \Lambda_{x\text{-}c} + \Lambda_{x\text{-}t3} = 40$. As $\Lambda_{x\text{-}c} > \Lambda_x$, t_3 is not executed at MU_3 but sent to DBS as *request* transaction.

DBS receives t_3, t_2 and t_1 in this order. Since t_3 is a request transaction it is executed after τ has expired and after the execution of t_2 and t_1 on the master copy. So the execution at DBS is: X=180, t_2, X=150, t_1, X=130, t_3, X=90 and after the execution, Λ is recomputed using the f_x (X, N_x). Thus, $\Lambda_x = f_x$ (X, N_x) = X/2N_x = 15. DBS broadcasts (X=90, Λ_x=15, τ) and each MU now can update the value of X by not more that 15 and sends the transaction for re-execution within τ (Figure 1).



a. Ti are sent to server **b.** New value of (Λ, X) are broadcasted

Fig 1 Intermediate Stages in the CCM scheme

Thus transactions on MU see inconsistent value of the number of tickets for τ after that DBS sends the consistent value of number of tickets. The transactions that want to know the number of tickets available will get approximate value of number of tickets. Inconsistency in value of data is bounded by refreshing the data value at a regular interval of τ and setting a limit on Λ on the maximum update that can be made during that period.

Case 2: MU_3 receives the values (X=90, Λ_x=15, τ) from the DBS. For every new τ, $\Lambda_{x\text{-}c}$ is reset to zero. t_4 and t_5 arrive at MU_3. t_4 books 10 and t_5 books 8 tickets. Let us assume they execute in the order t_4, t_5. After execution of t_4, $\Lambda_{x\text{-}t4}$ = 10 and $\Lambda_{x\text{-}c}$= $\Lambda_{x\text{-}c}$+ $\Lambda_{x\text{-}t4}$= 10. As $\Lambda_{x\text{-}c} < \Lambda_x$, t_4 is executed at MU_3 and t_4 is sent to the DBS for re-execution. Before τ expires t_5 arrives at MU_3 and $\Lambda_{x\text{-}t5}$ = 10. $\Lambda_{x\text{-}c}$ = 8. So $\Lambda_{x\text{-}c}$= $\Lambda_{x\text{-}c}$+$\Lambda_{x\text{-}t5}$ = 18. As $\Lambda_{x\text{-}c} > \Lambda_x$, t_5 is not executed at MU_3 and t_5 is send as request transaction to the DBS.

Case 3: MUs receive values (X=80, Λ_x = 0, τ) from DBS. T_6 arrives at MU_2 and books 2 tickets.

At MU_2: Initially $\Lambda_{x\text{-}c}$ = 0 t_6 books 2 tickets, so $\Lambda_{x\text{-}t6}$= 2. So $\Lambda_{x\text{-}c}$= $\Lambda_{x\text{-}c}$+ $\Lambda_{x\text{-}t6}$= 2 . As $\Lambda_{x\text{-}c} > \Lambda_x$, t_6 is not executed at MU_2 and is sent as a request transaction to the DBS.

All update transactions that arrive at MU are sent to DBS as request transaction since Λ_x is zero. Only read transaction can be performed on data item X at the MU.

4. Performance study

We have used simulation model to measure the performance of the concurrency control mechanism. Due to space limitation we do not include the queuing diagram. The server uses a flat broadcast program [4] in which all data items are broadcasted exactly once every broadcast cycle. MU continuously monitors the broadcast channel to read and caches the data items it needs. When MU is disconnected then a) it misses the broadcast and hence cannot read data items, which were broadcasted during that period which may be in the cache and b) it cannot send both request and precommitted transactions to the server for execution. After MU reconnects, the precommitted transactions have to be re-executed because the Λ values based on which the transaction had executed may not be valid after the disconnection period. Also transactions that could not be executed due to the crossing of Λ limits for MU, are blocked until the new Λ values arrive in the next broadcast. The delta values for the data items were classified in to (100,70,50,30,20). During a broadcast cycle equal number of data items have these five Λ values. We used this Λ value distribution to simulate the cases of large and small allowable changes in data at MU. The transaction updating a data item would change value of the data by amount with mean 0.2*Λ_x.

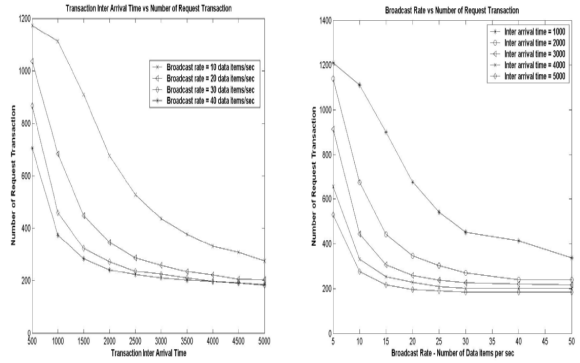
Effect of Transaction inter-arrival time and Broadcast rate: Fig 5a shows the graph of request transactions vs. inter arrival time. We observe the number of request transactions decrease increase with inter arrival time. The change in Λ of the data item will be less as inter arrival rate increases. As a result more transactions will be pre committed, as the Λ values of data item will not cross the quota Λ assigned to the MU. Fig 5b shows the graph of request transactions vs. broadcast rate. The period T of the broadcast cycle decreases with broadcast rate. Λ for each data item are refreshed after time T.

Effect of MPL: Figure 6b shows the effect of MPL on number of request transaction. MPL value differs for every transaction inter arrival rate and it depends on the period (T) of the broadcast cycle.

Effect of disconnections: Figure 6a shows the effect of No. of disconnections on No. of request transactions. As number of disconnection increases the MU will have more number of stale data items in the cache as MU cannot read from the broadcast and the τ of Λ of the data items will be expired. As a result transactions arriving at the MU cannot update those data items whose τs have been expired and have to wait till the next broadcast cycle. Increase in the disconnection duration leads to more number of data items whose timeouts are expired. We observe that number of request transactions increases with increase in number of disconnections and also are higher for larger disconnection duration.

Commit time of request and pre commit transactions: Fig 7a,b shows the effect of inter arrival time of transaction on commit time of precommit and request transaction. The precommit transactions are committed on the MU so they commit instantaneously as compared to the request transaction which have to be sent to the server. The commit time of request transaction is between 1.5T to 2T

(T is the period of the broadcast cycle). This is because the request transactions submitted during the broadcast cycle *I* get their result at the end of broadcast cycle *I+1*. We can send the commit results of request transactions instantaneously after they are executed at the server but broadcast will become aperiodic which is not a desirable because for optimal performance periodic broadcast is required [4]. Commit time for request transaction can be further reduced if reports are send at periodic intervals within a broadcast cycle. In fig 7a and 7b we observe that commit time of both precommit and request transaction decreases with increase in interarrival time because at lower inter arrival time request transactions are more (fig. 5a). At lower inter arrival times, Λ of data items expire faster and the transactions have to wait till new Λ are allocated in the next broadcast cycle. So commit time for precommit transaction is more at lower inter arrival time. At lower inter arrival time as the number of request transactions are more, server load increases and hence commit time of request transaction is also more as compared to at higher inter arrival time when number of request transaction are less (fig 5a).



(a) (b)

Figure 5

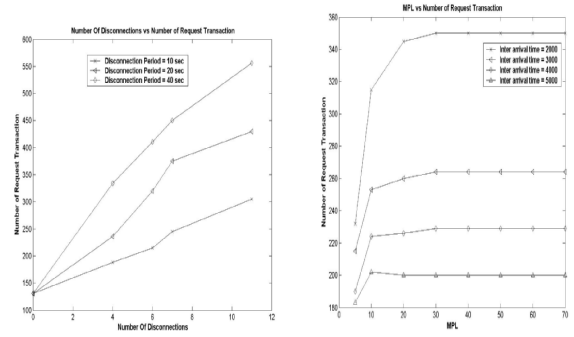
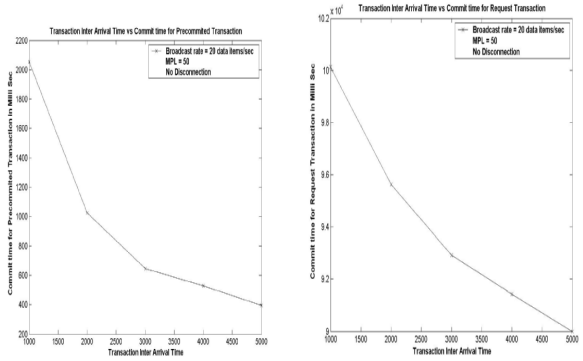


Figure 6 (b).

5. Conclusion

CCMs for MDS must take into account its inherent resource limitations in managing transactions execution. Keeping these in mind, we proposed a new concurrency control mechanism that uses ε serializability as the correctness criterion. We studied the effects of factors unique to the broadcast environment on the performance.



(a) (b)

Figure 7

10. References

[1] Daniel Barbara. "Mobile Computing and Databases – A survey". IEEE TKDE, 11(1), Jan'99

[2] M. H. Dunham, A. Helal, "Mobile Computing and Databases: Anything New?" SIGMOD Rec., 24(4), Dec. 95

[3] Evaggelia Pitroura and Bharat Bhargava, "Revising transaction concepts for Mobile Computing". Proc. Workshop on Mobile Computing Sy s and application, '94.

[4] S. Acharya, M. Franklin, S. Zdonik, "Dessemination-based Data Delivery Using Broadcast Disks". IEEE Personal Communications, 2(6), Dec 1995.

[5] Evaggelia Pitroura and Bharat Bhargava, "Maintaining Consistency of Data in Mobile Distributed Environment". In Proceedings of 15th ICDCS, Vancouver, Canada, 1995.

[6] Chrysanthis, P.K., "Transaction processing in Mobile Computing Environment". In IEEE workshop on Advances in Parallel and Distributed Sys.

[7] J. Gray, P. Helland, P. E. O'Neil, D. Shasha "The Dangers of Replication and a Solution ". In SIGMOD Conf. 1996

[8] Daniel Barbará "Certification Reports: Supporting Transactions in Wireless Systems". In ICDCS 1997.

[9] Kun-Lung Wu, Philip S. Yu, Calton Pu "Divergence Control for Epsilon-Serializability" ICDE 1992: 506-515

[10] Calton Pu. "Generalized transaction processing with epsilon-serializability" In Procs. of 4rt Int. Workshop on High Performance Transaction Sys., Asilomar, CA, Sep. 91.

[11] Gary D. Walborn, Panos K. Chrysanthis "Supporting Semantics-Based Transaction Processing in Mobile Database Applications". Proceedings of 14th IEEE Symposium on Reliable Distributed Systems, 1995.

[12] Jin Jing, Omran Bukhres, Ahmed Elmagarmid "Distributed Lock management for Mobile transaction". In the proceedings of ICDCS, 1995.

[13] S. Madria, Bharat Bhargava "A transaction model for Mobile computing". Proc. of Int. Database Eng. and Appn. Symp. 98.