1-1-1998

A Transaction Model for Mobile Computing

Sanjay Kumar Madria
Missouri University of Science and Technology, madrias@mst.edu

Bharat Bhargava

Recommended Citation

S. K. Madria and B. Bhargava, "A Transaction Model for Mobile Computing," *Proceedings of the International Database Engineering and Applications Symposium, 1998*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1998.
The definitive version is available at https://doi.org/10.1109/IDEAS.1998.694363

A Transaction Model for Mobile Computing

Sanjay Kumar Madria
School of Computer Science
University Sains Malaysia
1800 Minden, Penang, Malaysia
skm@cs.usm.my

Bharat Bhargava
Department of Computer Sciences
Purdue University, West Lafayett
IN 47907, USA
bb@cs.purdue.edu

Abstract

In this paper, we introduce a prewrite operation before a write operation in a mobile transaction to improve data availability. A prewrite operation does not update the state of a data object but only makes visible the value that the data object will have after the commit of the transaction. Once the transaction has read all the values and declares all the prewrites, it can pre-commit at mobile host. The remaining transaction's execution is shifted to the stationary host. Writes on database consume both time and resources at stationary host and are therefore, delayed. A pre-committed transaction's prewrite values are made visible both at mobile and stationary hosts before the final commit of the transaction. This increases data availability during frequent disconnection common in mobile computing. Since the expensive part of the transaction execution is shifted to the stationary host, it reduces the computing expenses at mobile host.

1. Introduction

Wide area and wireless computing suggest that there will be more competition for shared data since it provide users with ability to access information and services through wireless connections that can be retained even while the user is moving. Further, mobile users will have to share their data with others. The task of ensuring consistency of shared data becomes more difficult in mobile computing because of limitations of wireless communication channels and restrictions imposed due to mobility and portability ([5], [15]). The access to the future information systems through mobile computers will be performed with the help of mobile transactions. However, a transaction in this environment is different than the transactions in the centralized or distributed databases in the following ways.

- The mobile transactions might have to split their computations into sets of operations, some of which execute on mobile host while others on stationary host. A mobile transaction share their states and partial results with other transactions due to disconnection and mobility.
- The mobile transactions require computations and communications to be supported by stationary hosts.
- As the mobile hosts move from one cell to another, the states of transaction, states of accessed data objects, and the location information also move.
- The mobile transactions are long-lived transactions due to the mobility of both the data and users, and due to the frequent disconnections.
- The mobile transactions should support and handle concurrency, recovery, disconnection and mutual consistency of the replicated data objects.

To support mobile computing, the transaction processing models (see section 2.1 for review of some existing model) should accommodate the limitations of mobile computing, such as unreliable communication, limited battery life, low band-width communication, and reduced storage capacity. Mobile computations should minimize aborts due to disconnection.

Operations on shared data must ensure correctness of transactions executed on both stationary and mobile hosts. The blocking of transactions execution on either the stationary or mobile hosts must be minimized to reduce communication cost and to increase concurrency. Proper support for mobile transactions must provide for local autonomy to allow transactions to be processed and committed on the mobile host despite temporary disconnection.

Our objective is to increase data availability at mobile and stationary hosts. The main features of our mobile transaction model are :

- Each mobile transaction has a prewrite operation before a write operation. A prewrite operation makes visible the value the data object will have after the commit of the transaction.
- Once all the prewrites have been processed, the mobile transaction pre-commits at mobile host. A pre-committed transaction's results are visible at mobile and stationary hosts before the final commit. This minimizes the blocking of other transactions and increases concurrency.
- The transaction continues its execution at mobile host by announcing prewrite values and then shifts the resource consuming part of the transaction's execution (updates of the database on disk) to the stationary host. This reduces the computing cost on mobile host.
- A pre-committed transaction is guaranteed to commit. This feature of our model avoids an undo or compensating transaction, which is costly in mobile computing.
- A pre-read returns a prewrite value whereas a read returns a write value.
- The transactions are serialized based on their pre-commit order. This feature of our model saves some computing cost as transactions which can not be serialized or deadlocked are aborted before pre-commit.
- Our model deals efficiently with the constrained resources in mobile computing environment.

2. Mobile Architecture

In mobile computing environment (see figure 1), the network consists of stationary and mobile hosts [5]. A mobile host (MH) changes its location and network connections while computations are being processed. While in motion, a mobile host retain its network connections through the support of stationary hosts with wireless connections. These stationary hosts are called mobile support stations (MSS) or base stations which perform the transaction and data management functions with the help of transaction managers (TMs) and data managers (DMs), respectively. Each MSS is responsible for all the mobile hosts within a given small geographical area, known as a cell. At any given instant, a MH communicates only with the MSS responsible for its cell. A MH may have some server capability to perform concurrency control and logging etc. Some MH has very slow CPU and very little memory and thus, acts as an I/O device only. Within this mobile computing environment, shared data are expected to be stored and controlled by a number of database servers executing on MSS.

When a MH leaves a cell serviced by a MSS, a hand-off protocol is used to transfer the responsibility for mobile transaction and data support to the MSS of the new cell. This hand-off may involve establishing a new communication link. It involves the migration of in progress transactions and database states from one MSS to another.

2.1 Review of Mobile Transaction Processing Models

The mobile mobile transaction processing is an active area of research. We outline some of the existing ideas as follows.

- Semantic based transaction processing models ([2],[17]) have been extended for mobile computing in [18] to increase concurrency by exploiting commutative operations. These techniques require caching large portion of the database or maintain multiple copies of many data items. In [18], fragmentability of data objects have been used to facilitate semantic based transaction processing in mobile databases. The scheme fragments data objects. Each fragmented data object

has to be cached independently and manipulated synchronously. This scheme works in the situations where the data objects can be fragmented like stacks and queues.

- In optimistic concurrency control based schemes [6], cached objects on mobile hosts can be updated without any co-ordination but the updates need to be propagated and validated at the database servers for the commitment of transactions. This scheme leads to aborts of mobile transactions unless the conflicts are rare. Since mobile transactions are expected to be long-lived due to disconnection and long network delays, the conflicts will be more in mobile computing environment.
- In pessimistic schemes, cached objects can be locked exclusively and mobile transactions can be committed locally. The pessimistic schemes lead to unnecessary transaction blocking since mobile hosts can not release any cached objects while it is disconnected. Existing caching methods attempt to cache the entire data objects or in some case the complete file. Caching of these potentially large objects over low bandwidth communication channels can result in wireless network congestion and high communication cost. The limited memory size of the MH allows only a small number of objects to be cached at any given time.

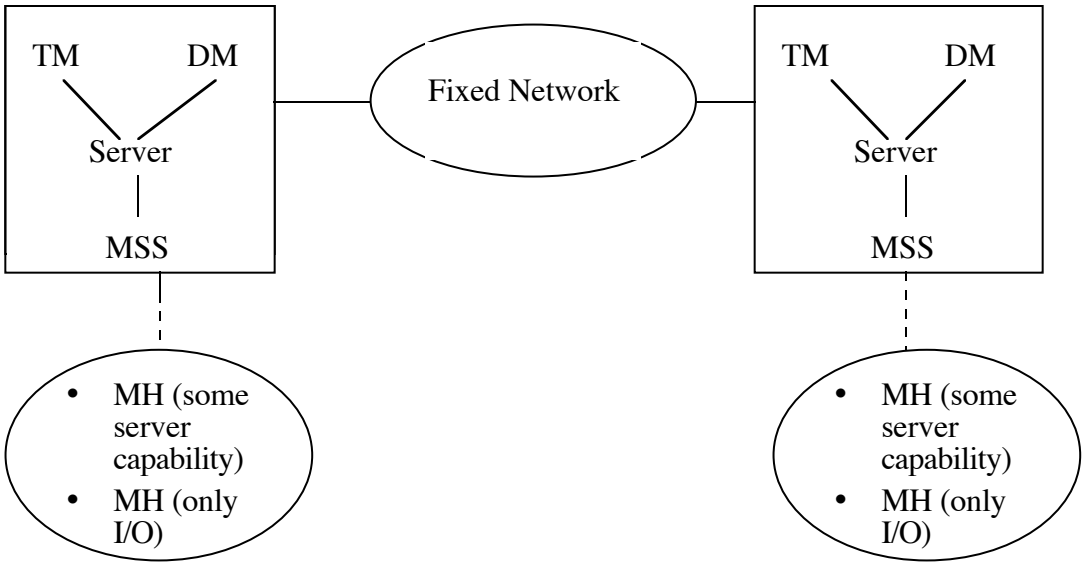

Figure 1. Mobile Architecture

- Dynamic object clustering has been proposed in mobile computing in ([13],[14]) using weak-read, weak-write, strict-read and strict-write. Strict-read and strict-write have the same semantics as normal read and write operations invoked by transactions satisfying ACID properties [1]. A weak-read returns the value of a locally cached object written by a strict-write or a weak-write. A weak-write operation only updates a locally cached object, which might become permanent on cluster merging if the weak-write does not conflict with any strict-read or strict-write operation. The weak transactions use local and global commits. The "local commit" is same as our "pre-commit" and "global commit"

is same as our "final commit" (see section 3). However, a weak transaction after local commit can abort and is compensated. In our model, a pre-committed transaction does not abort and hence, require no undo or compensation. A weak transaction's updates are visible to other weak transactions whereas prewrites are visible to all transactions.

- [7] presents a new transaction model using isolation-only transactions (IOT). IOTs do not provide failure atomicity and are similar to weak transactions of [13].
- An open nested transaction model has been proposed in [3] for modelling mobile transactions as a set of subtransactions. The model allows transactions to be executed on disconnection. It supports unilateral commitment of subtransactions and compensating transactions. However, not all the operations are compensated [3], and compensation is costly in mobile computing.
- A kangaroo transaction (KT) model was given in [4]. It incorporates the property that transactions in a mobile computing hop from a base station to another as the mobile unity moves. The mobility of the transaction model is captured by the use of split transaction [16]. A split transaction divides an ongoing transactions into serializable subtransactions. Earlier created subtransaction is committed and the second subtransaction continues execution. The mobile transaction is splitted when a hop occurs. The model captures the data behaviour of the mobile transaction using global and local transactions. The model also relies on compensating transaction in case a transaction aborts. Our model has the option of either using nested transactions or split transactions. However, the save point or split point of a transaction is explictly defined by the use of pre-commit. This feature of the model allows the split point to occur in any of the cell. Unlike KT model, the earlier subtransaction after pre-commit can still continue its execution with the new subtransaction since their commit orders in our model are based on pre-commit point. Unlike KT, our model does not need any compensatry transaction.
- Transaction models for mobile computing that perform updates at mobile computers have been developed in ([3],[13]). These efforts propose a new correctness criterion [3] that are weaker than the serializability. They can cope more efficiently with the restrictions of mobile and wireless communications.
- In ([9],[10],[11],[12]), prewrite operations have been used in nested transaction environment to increase concurrency and to avoid undo or compensated operations. The notion of a recovery point subtransaction has been introduced. In a nested transaction tree, if a recovery-point subtransaction executed successfully, its effects are not to be discarded. In this paper, we exploit some of these ideas in order to increase reliability and availability in mobile computing environment.

3. Prewrite Transaction Model

The prewrite transaction model presented in [11] has the following features :

- A prewrite operation announces the value that the data object will have after the commit of the corresponding write operation.
- A transaction is pre-committed if it has announced all the prewrites values and read all the required data objects, but the transaction has not been finally committed.
- A pre-read returns a prewrite value of the data object whereas a read returns the result of a write operation. A pre-read becomes a weak-read (in case the data objects involved are not simple) if it returns a prewrite value even though the transaction who announced the last prewrite has been finally committed. However, this weak-read should not to be aborted. This makes our weak-read different from the weak-read of [13].
- Each prewrite operation makes visible the value the transaction will eventually write. Prewrites may have different

semantics in different environments. For simple data objects, the prewrite and write values match exactly. For database files, the prewrites may only contain primary-key values and the new values of the fields of records. In case of design objects, prewrites may represent a model of the design. However, the final design released for manufacturing may differ from prewrite design of the model to some extent. For example, the final design may have a different colour shade than the prewrite design model. In case of a document object, the prewrite may represent an abstract of the detail document. For many read operations, these small differences make no difference to future computation.

- Once the required data objects are read or pre-read and prewrites are computed, the transaction pre-commits. A transaction is required to read or pre-read all the required data objects before pre-commit because after a transaction releases a lock for a prewrite operation, it can not get a lock for read operation due to the condition of two phase locking [1]. After a pre-commit, the prewrites are made visible to other transactions for processing. Initially, prewrites are kept in the transaction's private workspace. Once the transaction pre-commits, they are posted in the prewrite-buffer. The data objects are physically updated on the disk. The prewrites are handled at the transaction manager level whereas physical writes are handled at the data manager level.
- The transactions commit order in the serilaizable transactions execution history is decided at the order of pre-commit action.
- Our concurrency control algorithm is to be executed in two servers: For controlling pre-read (i.e., read of prewrite value) and prewrite operations at TM server and read (i.e., read of write value) and write operations at DM server. Since prewrite values are made publically visible after pre-commit, the lock-type held by the prewrite operation is converted to the lock-type for write operation after pre-commit provided no conflicting locks are held by other transactions. The lock acquired for a prewrite operation is not released after pre-commit because the two phase locking [1] does not allow a transaction to acquire a lock after the transaction has released some locks.
- A transaction is not allowed to abort after pre-commit. The prewrite provides non-strict execution without cascading aborts. In figure 2, T1 and T2 are two subtransactions where pw(x), w(x), pr(x), and r(x) are the prewrite value, write value, pre-read value and read-value respectively, for the data object x. The transaction T2 commits before T1. In case T1 aborts after T2 commits, there will not be a cascading abort. Since our model does not need "undo recovery" from transaction aborts [12], no compensating transaction is to be executed. In case the pre-committed transaction is forced to abort due to system dependent reasons such as system crash, the transaction restarts on system revival. In order to restart a failed pre-committed transaction from the last consistent state, prewrite and write logs are saved on stable storage [12].
- The model relaxes the isolation property as the prewrites are made visible to others after pre-commit but before the final commit of the transaction. Also, durability of prewrites is guaranteed at the pre-commit point.

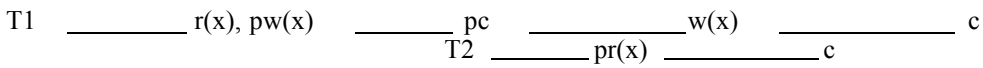

Figure 2. Two Concurrent Transactions

4. Prewrite Mobile Transaction Model

In this section, we see how prewrite model discussed in the previous section can be used in the mobile transaction applications. We discuss two cases.

Case 1: MH has limited server capability to do concurrency control, logging, and to execute pre-commit.

In our mobile transaction model, a transaction begins its execution at mobile host. When a transaction arrives at MH, the transaction's read requests are processed at MH in case it has the consistent cached data objects. Otherwise, the MH sends request for some of the reads (for which the MH has no consistent cached data objects) to the MSS. When a read access transaction arrives at MSS, it analyses the transaction and returns the prewrite value in response. If prewrite is not available, the write values are returned. The MSS tags to identify that the return value is a prewrite value. In case the transaction needs the final write, it has to initiate a read again. Once all the requested values are returned by MSS to MH, the transaction declares all the prewrite values for the data objects and pre-commits at MH. A pre-committed transaction's execution is then shifted from mobile host to the stationary host for the completion of its remaining execution. This moves the expensive part of the mobile transaction execution to the static network. At the stationary host, the transaction actually updates all the data objects for which the prewrite values have been declared earlier and commits thereafter (see figure 3). Thus, the prewrite value of the required data object is made visible by MSS to other mobile hosts in that cell before the data object is updated at the stationary host. This increases availability for concurrently running transactions. Prewrites are stored in the transaction's private workspace at MH and once the transaction pre-commits, they are moved to the main memory of MSS.

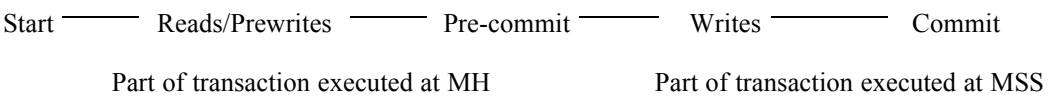

Figure 3. The Transaction Processing in Mobile Computing

Example 1: Real-Time Application

- Consider a newspaper reporter who is travelling in his van equipped with a mobile computer. If an accident site he likes to report immediately, he initiates a reporting transaction that consists of some "headline-report" about the accident and sends it to the newspaper office (MSS). The newspaper office immediately displays this on the web page as well as on electronic bulletin board. Since the mobile computer can run out of battery power or due to weak connections or network congestion, the reporter does not prepare and send the full-report. The "headline-report" corresponds to prewrite value of our model. It reduces the blocking of other transactions at stationary host as it may suffice the requirement of many transactions. For example, based on headline-report, some transactions like reservation of beds in hospitals for victims and transactions for seat reservations in air-lines for the relatives to travel to the accident site etc. can be initiated. Also, once the headline reports arrived at MSS, the reporting transaction's remaining execution is shifted to the MSS for the completion of report. There are two choices for the completion of full-report. First, the newspaper office (MSS) contacts some other reporter in that area to go to the accident site physically and file the complete report. Second, the reporter at MH can complete the full story at the time of disconnection (i.e., during lunch-time) or weak connections. Thus, MSS can deal effectively with the situations like failure of MH or in case the mobile host has crossed the cell or is in doze

mode. At the time of reconnection, the full-report would be incorporated into the database. The "headline-report" can also to be transmitted to other base stations in the fixed network so that transactions there can also be executed without delay.

Case 2: MH has very slow CPU and small memory, thus acts as an I/O device only.

In case the mobile host can not execute the transaction at MH (i.e., MH only acts as I/O), it can submit the transaction to the MSS. The MSS server returns all the required values and declares all the prewrite values corresponding to the write operations. After the transaction pre-commits, the prewrite values are send to the mobile host and at the same time, the rest of the transaction starts executing at the MSS.

Example 2 : Stock Buying-Selling Application

- Once a stock-selling transaction is accepted for sell at the given limit-price, the prewrites will have information about the amount of money available before the stock-selling transaction is committed at MSS. Once this information is received at MH, the user can initiate another stock-buying transaction and sends its "buying-order" to the corresponding MSS.

4.1 Mobile Transaction Model and Partially Replicated System

If the data objects are replicated partially, the mobile transaction makes visible all the prewrites of the required data objects available in its current cell and pre-commits in the current cell. If it does not do this and waits to announce the prewrite values of all other required data objects by moving into different cells, it will block other transactions until it visits those cells. Our approach of pre-committing in the current cell in such situations also creates some problems. Once a transaction is pre-committed, it can not again announce prewrite values for the objects in other cells. This is due to the fact that once the prewrite values are made visible at MSS by releasing some locks, it can not again acquire locks due to the two phase locking. This problem can be resolved in two ways. Either the transaction requests all the locks for all the required data objects and wait until all the locks are granted. This strategy will delay the execution severely in case some links are down. The other approach to solve the problem is by using either the nested transaction [8] or split transaction [16].

Nested Transaction Approach: To deal with the problem, we use the nested transaction model [8]. Once a transaction has announced all the prewrite values for the data objects available in its current cell and pre-committed, a new subtransaction is created. The earlier subtransaction is serialized before this new subtransaction based on pre-commit point. The earlier subtransaction's execution can be continued at the old MSS. The new subtransaction's responsibility is shifted to the MSS of the new cell with the help of hand-off protocol. Thus, both the transactions can be executed independently and concurrently in their respective cells. The pre-commit points are the save points of the nested transaction execution, thus they provide failure-tolerant execution. In case the earlier transaction aborts in the previous cell, its own effects and the new subtransaction are not undone. The aborted transaction can be restarted from the last pre-commit point. Another way of handling partially replicated data objects is to split [16] the transaction as soon as the MH moves to a new cell. The splitted transaction acts as a new transaction and therefore, can continue its processing in the new cell.

Prewrites can help in partial replication of the data objects in some of the cells it moves. That is, it can make some new servers to support its files. For example, suppose a transaction at MH has send a "headline-report" about the accident and moved to another cell. In the new cell, it sends the "headline-report" to its new MSS also. The MSS can therefore, get the headline-report before the full-report is processed at earlier MSS. Thus, the new MSS can serve those mobile transactions, which require headline-report for further processing like reservation of beds in hospitals in the current cell etc. Thus, transactions at the new MSS are not blocked until the completion of earlier transaction at the previous MSS.

5. Concurrent Operations and Locking

The operation-compatibility matrix of the various operations is given in table 1. In mobile transaction model, a pre-read operation can be executed at both MH and MSS at the same time. A pre-read can be executed at MH (MSS) while a write can take place at MSS (MH) concurrently. Similarly, a read can be executed concurrently with a prewrite. A prewrite operation can also be executed concurrently with another write operation since prewrites are managed at the transaction manager level whereas the writes are performed at the data manager level. However, there are some interesting cases as follows:

Case 1: Suppose a pre-read is currently being executed at MH and at the same time, another transaction which has announced the prewrite values finally commits at MSS (final updates are performed) (see figure 4(a)).

In this case, the pre-read will return a prewrite value which might be different than the last write value. For example, if the data object x is the simple data object, the read transaction T2 can commit as the prewrite and write values will be same. In case x is a design object, the system designate the read T2 as a weak- read since the final design may differ from the prewrite model. The transaction T2 can resubmit its read request later to MSS if it needs the latest complete model of the design.

Case 2: In table 1, observe that a read operation is compatible with a prewrite. Consider a case where a read transaction commits at MH after the transaction who announced the prewrite operation has been pre-committed.

The read in this situation will return an old value. However, this is not a significant problem because the transaction can still be serialized. For example, transaction T2 returns a write value, however it commits after T1 has pre-committed. The transaction T2 can be serialized before T1.

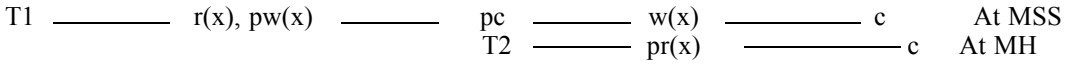

Figure 4 (a). The Situation in Case 1

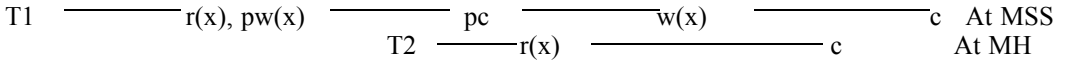

Figure 4(b). The Situation in Case 2

	Pre-read	Read	Pre-write	Write
Pre-read	Yes	Yes	No	Yes
Read	Yes	Yes	No	No
Pre-write	No	Yes	No	Yes
Write	Yes	No	Yes	No

Table 1. Operation Compatibility Matrix

Locking: We develop a concurrency control algorithm to control the conflicting operations in our mobile transaction model. The concurrency control algorithm is executed in two phases may be at two places. In the first phase, the concurrency control for controlling prewrite and pre-read operations is performed at the transaction manager level at MH or

MSS. In the second phase, the concurrency control for controlling write and read operations (to access write values) is performed at the data manager (DM) level at MSS mainly since the data managers are accessed only while performing updates on the databases.

We use read-lock for read, pre-read-lock for pre-read, prewrite-lock for prewrite, and write-lock for write operations, respectively. A prewrite-lock conflicts with other pre-read- and prewrite-locks, however, it does not conflict with read- and write-locks. A prewrite-lock can not be released after pre-commit as the transaction has to still get a write-lock for final updates. A prewrite-lock acquired by a pre-committed transaction is converted to a write-lock provided no other transaction holds the conflicting locks. Once a prewrite-lock is updated to a write-lock, the same transaction can not acquire any other lock [11]. However, pre-read-locks can be acquired by others to access prewrite values. The locking protocols are given in figure 5. The formal mobile transaction-processing algorithm is given in figure 6.

There will be no a deadlock involving the transactions which are pre-committed. This is due to the fact that prewrite- and write-locks are acquired in an ordered fashion so deadlocks will occur only at the time of acquiring prewrite- or read-locks. Thus, a pre-committed transaction will not be aborted due to the deadlocks.

Locks on the data are managed and provided by the MSS. In case of replicated objects, the number of locks to be acquired in our algorithm depends on the particular replication algorithm used. The two main replication algorithms used are majority consensus and read-one-write all (ROWA) [1]. In majority consensus algorithm, locks are acquired on majority of sites whereas read-one-write-all (ROWA) requires lock on all the sites. In case read/write ratio is less, ROWA is preferred otherwise majority consensus will be preferred.

Pre-Read-Lock(X): Grant the requested pre-read-lock to a transaction T on X if no other transaction holds a prewrite-lock on X.

Read-Lock(X): Grant the requested read-lock to a transaction T on X if no other transaction holds a write-lock on X.

Prewrite-Lock(X): Grant the prewrite-lock to a transaction T on X if no other transaction holds a Prewrite- or pre-read-lock on X.

Write-Lock(X): A request to update a prewrite-lock on X held by a transaction T to write-lock on X is granted as follows:

```
Begin
  If the write-lock-wait queue for X is empty then
    Begin
     If the transaction T is pre-committed and no other transaction holds a read- or write-lock on X then convert prewrite-lock to write-lock;
    End;
  else
    Begin
     put the transaction T in a write-lock-wait queue for X;
    End;
End.
```

Figure 5. The Locking Protocol

---

1. A transaction T is submitted to the mobile host.
2. Mobile host analyzes the transaction T to find out about its read and write requests (we assume here that MH has some server capability).

(/*pre-commit part of the algorithm executed at MH*/)

3. If the transaction T has read and write operations then

```
Begin
  For all reads and writes ∈ T
    MH sends a request to MSS for all the required read values and request for prewrite-locks;
  After MSS acquirs the necessary read-locks, it returns read values to MH (return values will be prewrite value. If no prewrite value is found, the write values are returned
  For all writes ∈ T
    Begin
      Announce all the prewrite values at MH (without waiting for prewrite-locks);
      Store the prewrite logs at MH;
```

Write pre-commit log record and destination move log record;
Send prewrite values, prewrite logs and other log records to MSS;
If MSS acquired all prewrite-locks for all the data objects for which prewrite values have been announced then
Begin
MSS accepts the prewrite values and logs;
MSS update the prewrite-lock to write-lock provided no other transaction holds conflicting locks (see figure 5);
End;
else the prewrite values and corresponding logs are discarded at MH;
End;
End;
4. For each prewrite announced (/*post pre-commit algorithm executed at MSS*/)
Begin
Update those data objects in the database for which prewrites have been announced;
Store necessary log records;
End;

Figure 6. Mobile Transaction Processing Algorithm

6. Conclusion

In this paper, we have presented a mobile transaction model using prewrites to increase availability. The model allows a transactionís execution to shift from the MH to MSS for database updates, thus minimizes the computing expenses at MH. Prewrite values help in increasing availability as the transactions can be executed during disconnections both at MH and MSS without blocking. For future work, we would like to discuss a detail crash recovery algorithm for mobile transaction model as well as some implementation issues and formal correctness proof.

References

[1] Bernstein P, Hadzilacos, and Goodman, N., Concurrency Control and Recovery in Database Systems, Addison-wesley Publishing Co.,1987.

[2] Barghouti, N., and Kaiser G., Concurrency Control in Advanced Database Applications, ACM Computing Surveys, 23(3):269-317,1991.

[3] Chrysanthis, P.K., Transaction Processing in a Mobile Computing Environment, Proceedings of IEEE workshop on Advances in Parallel and Distributed Systems, pp.77-82, Oct.1993.

[4] Eich, M.H.and Helal, A., A Mobile Transaction Model That Captures Both Data and Movement Behaviour, ACM/Baltzer Journal on Special Topics on Mobile Networks and Applications, 1997.

[5] Imielinksi T. and Badrinath B. R., Wireless Mobile Computing:Challenges in Data Management, Communications of ACM, 37(10), October 1994.

[6] Kisler J. and M. Satyanarayanan, Disconnected Operation in the Coda File System, ACM Transactions on Computer Systems, 10(1), 1992.

[7] Lu Q. and Satyanaraynan, M., Improving Data Consistency in Mobile Computing Using Isolation-Only Transactions, in proceedings of the fifth workshop on Hot Topics in Operating Systems, Orcas Island, Washington, May 1995.

[8] Moss, J.E.B., Nested Transactions: An Approach to Reliable Distributed Computing, Ph.D. Thesis. Also, Technical Report MIT/LCS/TR-260 MIT Laboratory for Computer Science, Cambridge, MA., April, 1981.

[9] Madria, S.K., Concurrency Control and Recovery Algorithms in Nested Transaction Environment and Their Proofs of Correctness, Ph.D. Thesis, Department of Mathematics, Indian Institute of Technology, Delhi, 1995.

[10] Madria, S.K., A Prewrite Transaction Model, accepted for 3rd International Baltic Workshop on Database and Information System to be held in Riga, Latvia, April 15-17.

[11] Madria, S.K. and Bhargava, B., System Defined Prewrites to Increase Concurrency in Databases, accepted for First East-Europian Symposium on Advances in Databases and Information Systems (sponsored by ACM-SIGMOD), St.-Petersburg (Russia), Sept.97.

[12] Madria, S.K., Maheshwari, S.N, Chandra, B., Bhargava, B., Crash Recovery Algorithm in an Open and Safe Nested Transaction Model, 8th International Conference on Database and Expert System Applications (DEXA), France, Sept.97. Appeared in Lecture Notes in Computer Science, Vol. 1308, Springer Verlag.

[13] Pitoura E. and B. Bhargava, Building Information Systems for Mobile Environments, Proceedings of 3rd International Conference on

Information and Knowledge Management, pp.371-378, 1994.
[14] Pitoura E. and B. Bhargava, Maintaining Consistency of Data in Mobile Computing Environments, in proceedings of 15th International Conference on Distributed Computing Systems, June,1995.
[15] Pitoura E. and B. Bhargava, Dealing with Mobility: Issues and Research Challenges, Technical Report TR-93-070, Department of Computer Sciences, Purdue University, 1993.
[16] Pu C., Kaiser G., and Hutchinson, Split-transactions for Open-ended Activities, in proceedings of the 14th VLDB Conference, 1988.
[17] Ramamritham K. and Chrysanthis. P.K., A Taxonomy of Correctness Criterion in Database Applications, Journal of Very Large Databases, 4(1), Jan.1996.
[18] Walborn, G. D., Chrysanthis, P.K., Supporting Semantics-Based Transaction Processing in Mobile Database Applications, in proceedings of 14th IEEE Symposium on Reliable Distributed Systems, pp.31-40, Sept.1995.