Distributed and Parallel Databases, 11, 73–92, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

HiCoMo: High Commit Mobile Transactions

MINSOO LEE mslee@cise.ufl.edu
SUMI HELAL helal@cise.ufl.edu
Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL 32611-6120, USA

Recommended by: Jin Jing

Abstract. We introduce a new mobile transaction model applicable to decisionmaking applications over aggregate data warehoused on mobile hosts. The model allows the aggregate data to be updated in disconnection mode, while guaranteeing a very high rate of commitment on reconnection. We name such transactions High Commit Mobile Transactions, or HiCoMo. At reconnectiontime, HiCoMo's are analyzed and several base (fixed network) transactions are generated in order to bring the same effect upon the base tables from which the aggregates are derived. In this paper, we provide a formal definition for the concepts related to HiCoMo's, and a transformation algorithm that is used to analyze them and generate base transactions. We provide a simple example scenario to demonstrate the usefulness of this transaction model. Finally, we compare the commit behavior of HiCoMo's to that of the two-tier model using simulation and a mobile transaction benchmark drawn from an inventory application domain.

Keywords: mobile transactions, two-tier replication, disconnected operation, transaction models

1. Introduction

There is a significant body of research addressing issues of mobile transaction modeling and processing [2–4, 6, 7, 10]. Most of these models are optimistic in nature, allowing the mobile transaction to work in isolation (e.g., while disconnected) and rely on deferred conflict detection and resolution upon reconnection. Some models address the weak type of connection (no disconnect/connect scenarios) and enforce relaxed consistency during the mobile transaction execution. Working in disconnection mode for prolonged periods of time often lead to large accumulation of conflicts and eventually high abort rate. In addition, attempting to commit transactions over high BER slow wireless networks (in weakly connected mode) costs high atomicity-related abort rate. New transaction models need to address the poor commit behavior more directly. In this paper, we focus on a new breed of transaction models that relies on a relaxed definition of the basic notion of *conflict*. This is different from the approaches taken by most advanced transaction models, which rely on a relaxed notion of consistency (e.g., relaxation of serializability into quasi-serializability).

The new model, which we call High Commit Mobile Transactions (or HiCoMo) guarantees a high commit rate regardless of the disconnection behavior. Our approach is applicable to derived data such as aggregates and other summary and statistical data, and is used in an environment where there exists base tables in the fixed network and a data warehouse (aggregates of the data in the base tables) on the mobile hosts. The mobile host is assumed to be mostly disconnected from the fixed network.

Transactions initiated from the fixed network on the base tables are called base (source) transactions. Transactions initiated on the mobile hosts performing operations on the aggregate data are HiCoMo transactions. HiCoMo's consisting of queries and updates are issued on a warehouse of statistical data on the mobile host which could be disconnected from the fixed network. The updates performed on the data warehouse are intended to be reflected later on upon the base tables. When the mobile host reconnects to the fixed network, HiCoMo transactions are processed and transformed into updates on the base tables. It may not be possible to generate updates on the base tables that can accurately reflect the effects of the HiCoMo transactions on the data warehouse. In this case, a reasonable error margin is allowed. Therefore, it is almost always guaranteed that the HiCoMo's will succeed.

The key point in HiCoMo transactions is the processing carried out at the time when the mobile host reconnects to the fixed network and generates source transactions to create the effect that was done by a HiCoMo during disconnection. There are various ways that these source transactions can be generated. As these transactions will be performed on a large set of data, the operations can be spread upon several of the base tables and the operations can be generated so as to cooperate to bring about the overall effect of a HiCoMo transaction. If the overall effect cannot be realized by any of the configurations of the generated transactions, a more relaxed method of generating transactions is used, by allowing an error margin between the actual updates on the base tables and the updates on the data warehouse. In developing the HiCoMo model, the following questions and issues need to be addressed:

- How should the source transactions, which have the overall effect of a HiCoMo, be generated?
- How can error margins be specified and considered in generating these source transactions?
- What advanced transaction models may be applied, or may be modified to support and realize HiCoMo's?

In the rest of this paper we provide a formal definition of HiCoMo along with our assumptions about the data and its aggregations on the mobile host. Section 2 lists the assumptions and related concepts. Section 3 provides a formal definition. The HiCoMo to source transaction transformation algorithm is presented in Section 4. An example scenario using HiComo transactions is presented in Section 5. Section 6 presents our experimental results hat we obtained through simulation. Future work and conclusions are given in Sections 7 and 8.

2. HiCoMo concepts and assumptions

In this section we list our assumptions and discuss some of the prerequisite concepts including source transaction generation, error margin consideration in generating source transactions, the need for an extended nested transaction model, and convergence criteria that determine the success and completion of HiCoMo's.

2.1. Assumptions

We make the following assumptions regarding the applications and data sets over which HiCoMo's are intended to be used:

1. The aggregate data that will be stored in the data warehouse is one of the following: *average, summation, minimum*, and *maximum*,
2. Only commutative operations will be applied to the data warehouse such as: addition $(+)$, substraction $(-)$. The commutative property can be found in many applications. BY exploiting it, less conflicts will occur among different HiCoMo transactions applied on multiple copies of the data warehouse. If the operations are commutative, the HiCoMo transactions can be applied without a strong restriction on the order, whereas addition $(+)$ and multiplication $(\times)$ operations require that the operations be applied in the order they have occurred.
3. The error margin value will be specified at the time when the HiCoMo transaction is issued in terms of an absolute value. For example, if a HiCoMo transaction increases the average by 3000 and has an error margin of 20, the actual update effect on the base tables may be an increase within the range [2980, 3020].

2.2. *Generation of source transactions*

Source transactions will be generated based on the combinations of the aggregate type (i.e., average, summation, minimum, maximum), the operation type (i.e., addition, subtraction), and the base table configuration. A transformation algorithm is needed to analyze HiCoMo's and generate source transactions. Satisfiability (does there exist such a source transaction configuration?) is an important issue in designing this algorithm. Computability (can a satisfying source transaction configuration be computed within some time limits?) is also an important issue and should be addressed by the transformation algorithm.

Another alternative approach is to reinforce the HiCoMo transaction effects on the base tables without reapplying the entire transaction logic. This is based on the idea of just bringing the effect up to date on the base tables.

2.3. *Error margin consideration*

The error margin could be initially disregarded to maintain consistent data among the base tables and the data warehouse. After the consistent source transaction combinations are considered and none of them are possible candidates due to repeated conflicts with other HiCoMo transactions or base transactions performed during disconnection, the error margin is allowed into the source transaction generation process. The margin may be varied from a lower error to an upper error bound. Increasing the error margin may be guided through try and errors of previous attempts to generate source transactions.

2.4. *Extended nested transaction model*

The generated source transactions will be applied as a whole to the base tables. If some of the source transactions abort, another transaction configuration should be tried. A transaction model which can support a concurrent and yet individual abort scheme should be adopted. The nested transaction model can allow for this kind of behavior with concurrent and independent subtransactions. Each of the source transactions can be modeled as a subtransaction. Although this model is sufficient to apply to a single source transaction configuration on the base tables, it does not capture the semantics of retrying another source transaction configuration when some subtransactions abort. Thus, the nested transaction semantics need to be slightly modified by capturing the semantics of retrying a subtransaction that has aborted using a new subtransaction replacement. This will modify the commit semantics of the nested transaction model.

2.5. Convergence as an acceptance criteria

Convergence is a simple yet very widely recognized acceptance criteria. It basically states that if eventually the base tables state is equal to what is reflected on the data warehouse, convergence is satisfied. If the operations are commutative, this acceptance criteria is satisfied most of the time. But, it is still possible that the operations are commutative and the acceptance criteria may not be satisfied due to interactions with other HiCoMo transactions. Some other HiCoMo transaction may have updated the base tables, and although the source transactions that were generated are applied to the base tables, it may not bring them to a state different from the data warehouse state. This simple convergence criteria needs to be investigated with respect to the HiCoMo transactions.

3. Formal definition of HiCoMo transactions

In order to clarify the semantics of a HiCoMo transaction, a formal definition is provided in this section. The problem to be tackled in supporting HiCoMo transactions is also identified with a formal representation through this process of formalization.

Several preliminary definitions are needed.

Definition 1. An **Aggregate Function (AF)** is a function that is applied to several Base Tables to obtain a table of aggregate values that represent statistics of the Base Tables. It is defined as a tuple of the following form:

$$\mathsf{AF} = (\mathsf{AGG\_OP},\ \mathsf{AGG\_FIELD},\ \mathsf{GROUP\_FIELD},\ \mathsf{n},\ \{\mathsf{SP}_1,\ \mathsf{SP}_2,\ \mathsf{SP}_3, \ldots, \mathsf{SP}_\mathsf{n}\})$$

- **AGG_OP** is an aggregate operation such as average(AVG), summation(SUM), minimum(MIN), maximum(MAX) etc.
- **AGG_FIELD** is the field(column) name of the final table that the AGG_OP should be applied to.
- **GROUP_FIELD** is the field(column) name of the final table that a group-by operation should be applied when performing the AGG_OP on AGG_FIELD. **n** is the number of Base Tables that are expected as input to the AF function. (i.e, 10, 25 ...).
- $\{\mathbf{SP_1}, \mathbf{SP_2}, \mathbf{SP_3}, \ldots, \mathbf{SP_n}\}$ is the set of select-project operations that are to be performed on each of the Base Tables. Each $\mathrm{SP_i}$ is the select-project operation that is to be performed on the Base Table $\mathbf{B_i}$ (given as the input to AF) where $1 \leq \mathrm{i} \leq \mathrm{n}$.

When the AF is applied to a set of Base Tables, it yields a single table containing aggregate values.

$$\begin{aligned}
&\mathsf{AF}(\{\mathsf{B}_1, \mathsf{B}_2, \mathsf{B}_3, \ldots, \mathsf{B}_n\})\\
&\quad = \mathsf{AGG\_OP}\left(\mathsf{AGG\_FIELD}, \mathsf{GROUP\_FIELD}, \left(\bigcup_{i=1,\ldots n} \mathsf{SP}_i(\mathsf{B}_i)\right)\right)\\
&\quad = \mathsf{AGG\_OP}(\mathsf{AGG\_FIELD}, \mathsf{GROUP\_FIELD}, \mathsf{T1})\\
&\quad = \mathsf{AT}
\end{aligned}$$

T1 is the table that is the result of calculating $\bigcup_{i=1,\ldots n} \mathsf{SP}_i(\mathsf{B}_i)$. AT is the resulting Aggregate Table. Note that these input Base Tables can have a different schema, but after the select-project (SP_i) operations are performed on each of the tables, the resulting tables should be in the same form that commonly allows an aggregation operation on a specific field along with a group-by operation.

Definition 2. A tuple (**AF, B_SET**) is called the **Evironment Tuple**, and represents the current environment that a HiCoMo transaction is used. **AF** is the Aggregate Function that maps a set of n Base Tables to a single Aggregate Table (i.e., the domain of the function is: $T^n \rightarrow T$, where T is a table). **B_SET** is a set $\{B_1, B_2, B_3, \ldots, B_n\}$, where each B_i is a Base Table existing in the fixed network. Actually, the set of Base Tables can also be viewed as one table which is horizontally partitioned into several tables.

Ideally, we would desire that the following condition always hold, but in the mobile environment need to relax this condition due to disconnection:

$$\mathsf{AF}\ (\{\mathsf{B}_1, \mathsf{B}_2, \mathsf{B}_3, \ldots, \mathsf{B}_n\}) = \mathsf{AT}$$

where AT is the actual Aggregate Table residing on the mobile host, and should be equal to the table derived by applying the Aggregate Function to the Base Tables stored on the fixed network, in the optimal case (i.e., when no disconnection occurs).

Definition 3. A **Transaction Transformation Function (TTF)** is a function that transforms a HiCoMo Transaction (HTR) applied on the Aggregate Table of the mobile host into a set of Base Transactions that are to be applied on the Base Tables. The TTF is what need to be identified to support HiCoMo transactions. The TTF can be applied on a HiCoMo Transaction in the following form:

$$\begin{aligned}
&\mathsf{TTF}\ (\mathbf{HTR}, \mathsf{AF}, \{\mathsf{B}_1,\mathsf{B}_2, \ldots, \mathsf{B}_n\}, \mathsf{AT}, \{\mathsf{IC}_1, \mathsf{IC}_2, \ldots, \mathsf{IC}_j\},\\
&\qquad \{\mathsf{HTR}_1, \mathsf{HTR}_2, \ldots, \mathsf{HTR}_m\}, \{\mathsf{BTR}_1, \mathsf{BTR}_2, \ldots, \mathsf{BTR}_p\})\\
&\quad = \{\{\ \mathbf{BTR_{p+1}, BTR_{p+2}, \ldots, BTR_k}\ \}, \{\mathsf{B}'_1, \mathsf{B}'_2, \ldots, \mathsf{B}'_n\}, \mathsf{AT}',\\
&\qquad \{\mathsf{HTR}'_1, \mathsf{HTR}'_2, \ldots, \mathsf{HTR}'_m\}\}
\end{aligned}$$

- **HTR** (={**SEL, GROUP_FIELD, OP, AGG_FIELD, VAL, ERR, AT**}) is the **HiCoMo Transaction** to be transformed. It is composed of a selection operation (SEL) on a group-by field(GROUP_FIELD) of the aggregate table (AT), an operation (OP) such as increase(+) or decrease(−) on one of the aggregate fields (AGG_FIELD) of the aggregate table, a value (VAL) that indicates the amount of increase or decrease, an error margin (ERR) allowable, and an Aggregate Table (AT) that the HiCoMo Transaction HTR is applied upon. **AF** is the Aggregate Function defined above.

- $\{\mathbf{B_1, B_2, \ldots, B_n}\}$ is the set of Base Tables reflecting the state *before* the HiCoMo Transaction HTR is transformed and applied to the base tables.
- **AT** is the Aggregate Table *before* the HiCoMo Transaction HTR is applied.
- $\{\mathbf{IC_1, IC_2, \ldots, IC_j}\}$ is the set of Integrity Constraints on the Base Tables.
- $\{\mathbf{HTR_1, HTR_2, \ldots, HTR_m}\}$ is the set of *possibly* conflicting HiCoMo Transactions with HTR *before* HTR is transformed and applied. In our case, we only consider commutative operations for HiCoMo Transactions. Thus, HiCoMo Transactions do not conflict with each other.
- $\{\mathbf{BTR_1, BTR_2, \ldots, BTR_p}\}$ is the set of conflicting Base Transactions with HTR.
- $\{\mathbf{BTR_{p+1}, BTR_{p+2}, \ldots, BTR_k}\}$ is the set of Base Transactions that were generated by the HiCoMo Transaction HTR. This may also be a set containing only ABORT, which means that HTR cannot be transformed and must be aborted.
- $\{\mathbf{B'_1, B'_2, \ldots, B'_n}\}$ is the set of Base Tables reflecting the state *after* the HiCoMo Transaction HTR is transformed and applied to the base tables.
- $\mathbf{AT'}$ is the Aggregate Table *after* the HiCoMo Transaction is applied.
- $\{\mathbf{HTR'_1, HTR'_2, \ldots, HTR'_m}\}$ is the set of *possibly* conflicting HiCoMo Transactions with HTR *after* HTR is transformed and applied. In our case, as there are no conflicts among HiCoMo Transactions, it should be the same as *before* the HiCoMo Transaction HTR is applied.

The following sections focus on how to provide this HiCoMo Transformation Function. An algorithm for performing the transformation and an extended nested transaction model are developed to support the HiCoMo Transaction concept.

4. Transformation algorithm

The Transaction Transformation Function, which was previously described, is the key focus of this section. Recall that the TTF needs as input the HiCoMo Transaction, the Aggregate Function, the Base Tables, the Aggregate Table, the Integrity Constraints, the conflicting HiCoMo Transactions, and the conflicting Base Transactions. All of these factors are considered when transforming a HiCoMo Transaction into a set of Base Transactions. A general description of the algorithm for performing the transformation is as follows.

Step 1. Conflict detection: Detect a conflict between the given HiCoMo Transaction and the other HiCoMo Transactions that are not yet transformed into Base Transactions, and also the conflicting Base Transactions. In our case, actually no conflicts are detected among HiCoMo Transactions. If a conflict is detected, abort the given HiCoMo Transaction that is being considered for transformation. This simple abort strategy is due to the fact that we want to provide durability for Base Transactions, and more significantly, there is no way to control what happened within other Base Transactions.

Step 2. Initial base transaction generation: If there is no conflict, decide what kind of initial Base Transactions should be created. This is based on the information provided by the HiCoMo Transaction and other factors such as the Aggregate Function and Base Tables. Once these are decided, run these transactions as sub-transactions of an extended nested transaction.

Step 3. Sub-transaction abort and alternate base transaction generation: Some of the subtransactions may abort due to integrity constraints that restrict certain updates from being performed. This case is difficult to predict prior to generating the Base Transaction. Therefore, the aborted transactions need to be dealt with afterwards. The aborted subtransactions will affect the overall result of the HiCoMo Transaction, and need to be somehow compensated for. If the difference that the aborted sub-transactions create is within an error margin, it is allowed to just simply finish with several aborted subtransactions. Otherwise, the update amount for the aborted sub-transactions may be redistributed among themselves and retried. If after a certain amount of redistribution is done and the abort of the sub-transactions are still occurring, the error margin may also be considered in the redistribution. If the sub-transactions can succeed at some point, the transformation is complete. Otherwise, the HiCoMo Transaction need to be aborted.

The Extended Nested Transaction model is a simple extension to the traditional nested transaction model. The main feature is that is has a modified commit semantics. The failure of the sub-transactions are localized, but the failures are not just left as it is. When a sub-transaction fails, the top-level transaction is notified of the failure. When all of the sub-transaction fates are decided, the top-level transaction will go on to the next step which is a modified step added by our model. The top-level transaction will readjust the sub-transaction parameters and rerun the sub-transaction. If the sub-transaction aborts again, it may be rerun again with another parameter settings. The criteria for stopping this retrying semantics is decided by the algorithm discussed above. An aggregate update amount should be reflected within a pre-specified error margin given to the HiCoMo Transaction. Therefore the condition for committing the top-level transaction is as follows.

```
If fate of all sub-transactions are decided
{
        Gather all results of sub-transaction.
        If results satisfy error margin of HiCoMo Transaction, commit.
        If not,
        {
            If error margin has been fully exploited, abort.
            If not, use regeneration algorithm of Base Transactions to
            retry sub-transactions.
        }
}
```

A detail description of each of the steps for the transformation algorithm are provided in the following sections.

4.1. Step 1: Conflict detection

One of the assumptions we made earlier is that we only consider commutative operations for HiCoMo Transactions. In other words, the OP part of the HiCoMo Transaction will only be allowed to be addition (+) or subtraction (−), and not division nor multiplication. This assumption makes it possible to reorder the HiCoMo Transactions, and eliminates the conflicts among HiCoMo Transactions.

Therefore, we only need to detect conflicts between the given HiCoMo Transaction and the Base Transactions. The Base Transactions can be composed of any type of operations including not only addition or subtraction but also division or multiplication. Thus, Base Transactions are not commutative with HiCoMo Transactions. So, the *order between the HiCoMo Transaction and Base Transactions are important*. We also want to guarantee durability for Base Transactions, as they are always performed in the fixed network without disconnection problems. This means that when there is a conflict between a HiCoMo Transaction and some Base Transactions, *the HiCoMo Transaction should be aborted* and not the Base Transactions.

The conflict detection can be implemented by a *time-stamp based optimistic concurrency control strategy* [1, 8]. In this strategy, a transaction goes through three phases: execution, validation, and update. The execution phase in our case is done when the HiCoMo Transaction performs updates to the aggregate table on the mobile host. This phase is actually considered as starting from the disconnection point (i.e., all read operations are done at this point) and ends when the mobile host is reconnected. The validation phase checks for any conflicts of these updates with any other updates. If the updates performed by the HiCoMo Transaction are younger (i.e., more recent) than any other conflicting Base Transaction updates, then the HiCoMo Transaction updates are allowed to be actually applied on the Base Tables. Otherwise, the HiCoMo Transaction is aborted. The validation phase starts when the mobile host has reconnected to the fixed network. The update phase is done by transforming the HiCoMo Transaction into Base Transactions and actually applying them to the Base Tables. Using this strategy, the transactions will be serialized in time-stamp order.

Note that the HiCoMo Transaction will eventually need to update a subset of the tuples in the Base Tables that are aggregated in the table of the mobile host. This subset may be large and may possibly result in many conflicts, but the basic problem is that there is no control over what amount of updates are performed by the Base Transactions. Thus, it is difficult to enforce a desired update effect by the HiCoMo Transaction. Advanced issues and solutions related to this problem are discussed later on.

4.2. Step 2: Initial base transaction generation

Recall that a HiCoMo Transaction is composed of {SEL, GROUP_FIELD, OP, AGG_FIELD, VAL, ERR, AT}. A simple example of a HiCoMo Transaction is as follows:

SEL: consultant
GROUP_FIELD: Level
OP: increase (+)
AGG_FIELD: Salary
VAL: 2000
ERR: 100
AT: aggregate table generated by the following SQL query

```
SELECT     Level, AVG(Salary)
FROM       Employee
WHERE      Employed_date < 01/01/1995
GROUPBY    Level
```

This HiCoMo Transaction will increase the average Salary of the manager Level employees who have started working prior to the date 01/01/1995 by $2000 with an error margin of $100.

By carefully looking into this example, the relationship between the HiCoMo Transaction and the desired generated Base Transactions can be observed.

First, the *target tuples of the generated Base Transactions* are decided based on: (1) which Base Table tuples have been aggregated into the aggregate table (i.e., WHERE clause of SQL query that produces AT), (2) and also which aggregates are selected by the HiCoMo Transaction within the aggregate table (i.e. SEL, GROUP_FIELD). If we denote the selection condition of (1) as S, then we obtain the following query that results in the target tuples of the generated Base Transaction.

```
SELECT   *
FROM     base tables
WHERE    S AND GROUP_FIELD = SEL
```

In the case of the above example, it would result in:

```
SELECT   *
FROM     Employee
WHERE    Employed_date < 01/01/1995 AND Level = manager
```

Second, the *different types of aggregate operations (i.e., AGG_OP in the aggregate function: AVG, SUM, MIN, MAX) used when creating the aggregate table* must be considered to generate the correct Base Transactions. The following rules are applied to perform the transformation. They use the AGG_OP, and the OP, AGG_FIELD, VAL information of the HiCoMo Transaction.

- If AGG_OP = AVG, generate Base Transaction(s) that will update each of the target tuples by applying the OP on the AGG_FIELD with the amount of VAL. In the above example, this means that the Base Transaction(s) will increase each of the Salary fields of the target tuples by $2000. This should bring about the effect of increasing the average by $2000.

- If AGG_OP = SUM, generate Base Transaction(s) that will update the first tuple among the target tuples by applying the OP on the AGG_FIELD with the amount of VAL. If we assume that the aggregate table in the above example was generated by a SUM, this means that the Base Transaction(s) will increase the Salary field of the first target tuple by \$2000. This should bring about the effect of increasing the summation by \$2000.
- If AGG_OP = MIN and OP = subtraction($-$), generate Base Transaction(s) that will update the target tuple with the minimum value on the AGG_FIELD by applying the OP on the AGG_FIELD with the amount of VAL. If we assume that the aggregate table in the above example was generated by a MIN, this means that the Base Transaction(s) will decrease the Salary field of the minimum target tuple by \$2000. This should bring about the effect of decreasing the minimum by \$2000.
- If AGG_OP = MIN and OP = addition($+$), generate Base Transaction(s) that will update the target tuple with the minimum value on the AGG_FIELD by applying the OP on the AGG_FIELD with the amount of VAL. Also, other Base Transaction(s) that update target tuples falling below this new minimum are generated. If we assume that the aggregate table in the above example was generated by a MIN, this means that the Base Transaction(s) will increase the Salary field of the minimum target tuple by \$2000. Additional Base Transaction(s) are generated to update other target tuples that have a Salary field below this new minimum to exactly the minimum value. This should bring about the effect of increasing the minimum by \$2000.
- If AGG_OP = MAX and OP = addition($+$), generate Base Transaction(s) that will update the target tuple with the maximum value on the AGG_FIELD by applying the OP on the AGG_FIELD with the amount of VAL. If we assume that the aggregate table in the above example was generated by a MAX, this means that the Base Transaction(s) will increase the Salary field of the maximum target tuple by \$2000. This should bring about the effect of increasing the maximum by \$2000.
- If AGG_OP = MAX and OP = subtraction($-$), generate Base Transaction(s) that will update the target tuple with the maximum value on the AGG_FIELD by applying the OP on the AGG_FIELD with the amount of VAL. Also, other Base Transaction(s) that update target tuples exceeding this new maximum are generated. If we assume that the aggregate table in the above example was generated by a MAX, this means that the Base Transaction(s) will decrease the Salary field of the maximum target tuple by \$2000. Additional Base Transaction(s) are generated to update other target tuples that have a Salary field exceeding this new maximum to exactly the maximum value. This should bring about the effect of decreasing the maximum by \$2000.

We assume that one Base Transaction is generated per Base Table. Therefore, if multiple Base Tables are aggregated, multiple Base Transactions may be generated by the HiCoMo Transaction. There are transaction granularity issues related to this decision of a per Base Table transaction generation. If the Base Transaction granularity is too small such as being generated for each target tuple rather than each Base Table, the problem is that the processing overhead of these large number of generated transactions may be tremendous. But the good thing is that each failure of a Base Transaction is localized. On the other hand, if the Base Transaction granularity is too large and a single target tuple update fails, then the whole Base

Transaction will need to be reconsidered. This decreases the transaction processing overhead due to the small number of Base Transactions generated, but can create unnecessary aborting of even succeeded large portions of the transaction.

There are also alternate ways of generating the Base Transaction(s) by using a different distribution of the update values among the target tuples of the Base Table(s). A simple example for the SUM discussed above, is to distribute parts of the total update amount among all of the target tuples instead of just the first one. The goal of the initial generation of the Base Transaction(s) is to not only *allow as uniform updates among the tuples as possible* but also to minimize the generated Base Transaction life time by *updating as less target tuples as possible*.

4.3. Step 3: Sub-transaction abort and alternate base transaction generation

The initial Base Transaction(s) generated are executed as sub-transactions of an extended nested transaction that will be explained in the next section.

Although we are guaranteed that the Base Transactions generated will not conilict with other Base Transactions, it is still possible for the generated Base Transactions to be aborted. One of the main reasons would be the existence of *integrity constaints* on the fields that are being updated. An example of such an integrity constraint would be that a field should have a nonnegative value where the Base Transaction attempts to drive the value below zero. The sub-transaction that this Base Transaction is being executed in will be aborted.

The extended nested transaction localizes failure of sub-transactions. Therefore, individual aborts of sub-transactions do not directly affect the execution of other sub-transactions, but will definitely affect the overall aggregate update desired by the HiCoMo Transaction. The strategy proposed for this situation is:

- If the aborting of the sub-transactions still leave the overall update effect within a pre-specified error margin of the HiCoMo Transaction, it may be alright to leave the aborted sub-transactions as they are.
- If a large number of sub-transactions abort and the overall update effect is not within the pre-specified error margin of the HiCoMo Transaction, a redistribution or alternate strategy of generating Base Transactions for the aborted sub-transactions should be activated.

The error of the overall update effect can be calculated using the total number of target tuples, the number of aborted sub-transactions, the type of aggregate, the HiCoMo Transaction operation and value, the value of the target tuple before the sub-transaction aborts, and the value of the target tuple after the sub-transaction succeeds.

When aborted sub-transactions need to be rerun, they will be generated according to the following rules:

- If AGG_OP = AVG, regenerate Base Transaction(s) for the aborted sub-transactions with consideration of error margins. The error margin that should be applied depends on the total abort ratio that had occurred. Assuming 10% of the sub-transactions had aborted, we apply 10% of the error margin to the update values on each of the aborted subtranactions. Thus, the VAL will be decreased by 10% of the error margin and applied to each target tuple of the aborted sub-transactions. In the above example, this means that the Base Transaction(s) will increase each of the Salary fields of the target tuples by only \$1990(=\$2000 − \$100*10%). This should bring about the effect of increasing the average by \$2000 with an error margin within \$10(=\$100*10%). The error margin would then be applied in the next round with double (20%), triple (30%) the initial ratio as this process goes on until the error margin is no longer adjustable. Each time before the regeneration process of the Base Transactions, the total error is calculated including the aborted sub-transactions. If the result is not satisfactory, another round of regeneration starts. Otherwise, it will stop successfully.

After going through all of the regeneration, it may be possible that the total error margin cannot be satisfied. In this case, the HiCoMo Transaction is aborted. A simple reason why redistributing values was not used before trying to use the error margin was that the aborted sub-transactions were *all* already making the resulting values to exceed or go below some integrity constraint. Trying to even out the values among the individual target tuples of the aborted sub-transactions will still make them *all* to abort. The only way to save some of the sub-transactions is to increase (or decrease) the potentially good ones by just a small value and put all of the left over big burdens on the potentially bad ones. This will create a hazard for the error margin and may become uncontrollable. Thus, we avoid this approach.

- If AGG_OP = SUM, regenerate Base Transaction(s) that will update the *next* tuple among the target tuples by applying the OP on the AGG_FIELD with the amount of VAL. If we assume that the aggregate table in the above example was generated by a SUM, this means that the Base Transaction(s) will increase the Salary field of the *next* target tuple by \$2000. This should bring about the effect of increasing the summation by \$2000. Retry this for 10% (a fixed number) of the total tuples. If it doesn't work for this number of retries, redistribute the SUM amount in an evenly fashion over all of the target tuples. In other words, apply (VAL/number of target tuples) on each of the tuples. If an abort of a sub-transaction occurs, proceed as in the above AGG_OP = AVG case where error margins are considered.
- If AGG_OP = MIN and OP = subtraction(−), regenerate Base Transaction(s) that will update the target tuple with the minimum value on the AGG_FIELD by applying the OP on the AGG_FIELD with the amount of VAL adjusted by multiples of 10% of the error margin. Each time, the sub-transaction is aborted, the error margin increases (i.e., 20%, 30%, ...). If the error margin cannot be satisfied after all retries, then abort the HiCoMo Transaction. If we assume that the aggregate table in the above example was generated by a MIN, this means that the Base Transaction(s) will decrease the Salary field of the minimum target tuple by \$1990(\$2000 − \$100*10%). This should bring about the effect of decreasing the minimum by \$2000 with an error margin of 10%.
- If AGG_OP = MIN and OP = addition(+), regenerate Base Transaction(s) that will update the target tuple with the minimum value on the AGG_FIELD by applying the OP on the AGG_FIELD with the amount of VAL adjusted by multiples of 10% of the error margin. Also, other Base Transaction(s) that update target tuples falling below

this new minimum are generated. Each time, any sub-transaction is aborted, the error margin increases (i.e., 20%, 30%, ...). If the error margin cannot be satisfied after all retries, then abort the HiCoMo Transaction. If we assume that the aggregate table in the above example was generated by a MIN, this means that the Base Transaction(s) will increase the Salary field of the minimum target tuple by $1990(\$2000 - \$100 * 10\%)$. Additional Base Transaction(s) are generated to update other target tuples that have a Salary field below this new minimum to exactly the minimum value. This should bring about the effect of increasing the minimum by $2000 with an error margin of 10%.
- If AGG_OP = MAX and OP = addition(+), regenerate Base Transaction(s) that will update the target tuple with the maximum value on the AGG_FIELD by applying the OP on the AGG_FIELD with the amount of VAL adjusted by multiples of 10% of the error margin. Each time, the sub-transaction is aborted, the error margin increases(i.e., 20%, 30%, ...). If the error margin cannot be satisfied after all retries, then abort the HiCoMo Transaction. If we assume that the aggregate table in the above example was generated by a MAX, this means that the Base Transaction(s) will increase the Salary field of the maximum target tuple by $1990(\$2000 - \$100 * 10\%)$. This should bring about the effect of increasing the maximum by $2000 with an error margin of 10%.
- If AGG_OP = MAX and OP = subtraction(−), regenerate Base Transaction(s) that will update the target tuple with the maximum value on the AGG_FIELD by applying the OP on the AGG_FIELD with the amount of VAL adjusted by multiples of 10% of the error margin. Also, other Base Transaction(s) that update target tuples exceeding this new maximum are generated. Each time, any sub-transaction is aborted, the error margin increases (i.e., 20%, 30%, ...). If the error margin cannot be satisfied after all retries, then abort the HiCoMo Transaction. If we assume that the aggregate table in the above example was generated by a MAX, this means that the Base Transaction(s) will decrease the Salary field of the maximum target tuple by $1990(\$2000 - \$100 * 10\%)$. Additional Base Transaction(s) are generated to update other target tuples that have a Salary field exceeding this new maximum to exactly the maximum value. This should bring about the effect of decreasing the maximum by $2000 with an error margin of 10%.

5. Example scenario using HiCoMo transactions

HiCoMo Transactions are based on new type of applications that may be very common in the mobile computing environment. A CEO of a company may want to carry around only HiCoMo data of his company rather than the *whole* database which may be huge. He is the person who can make company-wise decisions and can directly update the HiCOMo data that he carries around and will be later on reflected onto the real database of his company.

A simple example scenario is provided in this section to demonstrate how the concept works. Suppose there are the following two tables Employee1 and Employee2 in the company. These tables are separately managed on different sites on the fixed network, but have the same schema (for simplicity purpose).

Employee1 on site A

Emp_no	Name	Level	Travel_allowance	Dept
104467	Jack Crane	Partner	79000	Finance
350933	Chris White	Consultant	68000	Finance
230988	Smith Gordon	Partner	69000	Marketing

Employee2 on site B

Emp_no	Name	Level	Travel_allowance	Dept
308867	Janette Sanders	Consultant	66000	Marketing
111436	Sue Hill	Consultant	74000	Manufacture
566217	Bart Simpson	Partner	82000	Management

The CEO takes the aggregate table which consists of the average travel allowance of his 6 (this is just a simplified version of the company) employees grouped by the category of Level on his mobile host. The table should look like this.

Level	AVG (Travel allowance)
Consultant	69333
Partner	76666

The CEO travels around and finds out that his consultant's travel allowance is too little to do the travel that is required considering the increasing amount of travel that is needed as the company is rapidly growing. Therefore, he wants to increase the average travel allowance of the consultants so that these important human resources would be able to travel in comfort and perform all the tasks that are required.

The CEO wants to increase the average by $6000 with an error margin of $100.

But there is an integrity constraint on the Travel allowance field such that the travel allowance of an employee who is not at the Partner Level should not exceed $80000.

When the CEO updates the value on his mobile computer, the travel allowance for the Consultant should change to 75333 [$\pm$100].

The CEO later reconnects to the fixed network and downloads his transactions on the company database. The base transactions will be generated from the HiCoMo transaction according to the algorithms discussed in earlier sections.

Assuming that there were no conflicting base transactions with the HiCoMo transactions, we go to the next step where the HiCoMo transaction is transformed into base transactions.

Two base transactions are identified, one for each Employee table. The two base transactions will attempt to increase the Travel allowance of the Consultants by $6000. These two base transactions are executed as sub-transactions of an extended nested transaction. The updates on Employee1 is successful, but the sub-transaction executed on Employee2 is aborted because of the integrity constraint enforced on Sue Hill (her travel allowance will exceed $8000). The sub-transaction on Employee2 is readjusted to increase only by $5990(=$6000 − $100*10%). This time the sub-transaction succeeds. Thus, the actual error margin is within $10(=$100*10%). The resulting Employee tables will look like the following:

Employee1 on site A

Emp_no	Name	Level	Travel_allowance	Dept
104467	Jack Crane	Partner	79000	Finance
350933	Chris White	Consultant	**74000**	Finance
230988	Smith Gordon	Partner	69000	Marketing

Employee2 on site B

Emp_no	Name	Level	Travel_allowance	Dept
308867	Janette Sanders	Consultant	**71990**	Marketing
111436	Sue Hill	Consultant	**79990**	Manufacture
566217	Bart Simpson	Partner	82000	Management

The resulting CEO table should also be updated to capture the correct average values. It should look like the following:

Level	AVG (Travel_allowance)
Consultant	**75326**
Partner	76666

An error margin of $7(=$75333 − $75326)! This is definitely within the error margin of $100.

6. Experimental results

The goal of the experiment is to investigate how the commit ratio is affected when using the HiCoMo transactions to process updates originating on a mobile computer. The experiment focuses on the time the integration of the HiCoMo is performed on the base nodes.

6.1. The simulator

The simulator is shown in the following figure 1. The simulator consists of two processing modules which are the HiCoMo Transaction Manager (HTM) and the Base Transaction Manager (BTM). The HTM is the main module which performs HiCoMo transaction

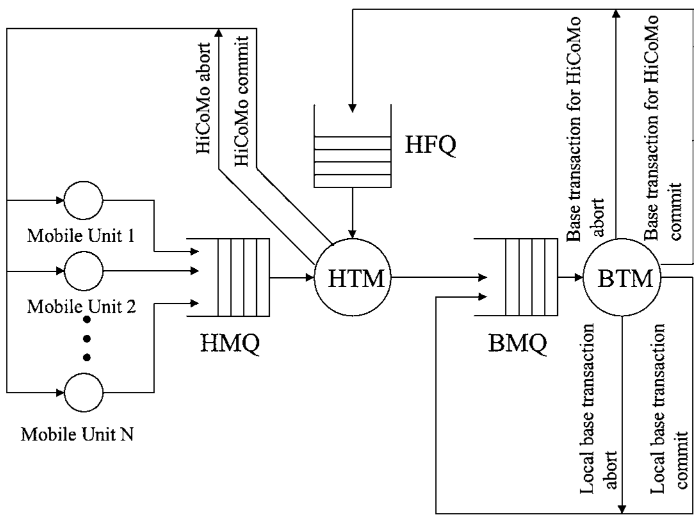

HiCoMo abort
HiCoMo commit
HFQ
Base transaction for HiCoMo abort
Base transaction for HiCoMo commit
Mobile Unit 1
Mobile Unit 2
Mobile Unit N
HTM
BTM
HMQ
BMQ
Local base transaction abort
Local base transaction commit

Figure 1. The simulator for HiCoMo transaction processing.

relevant tasks such as transforming the HiCoMo transactions into base transactions and also issuing them to the BTM for execution. The HTM keeps track of the HiCoMo transaction execution status and performs adjustments such as error margin consideration into the process of regenerating base transactions. The BTM is an ordinary transaction manager that resides on a fixed node maintaining base data. The BTM will accept a transaction execution request and return the results of the transaction execution.

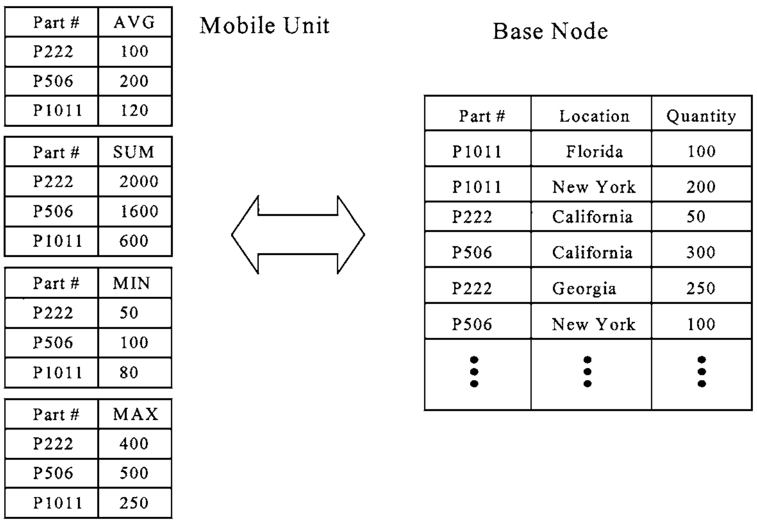
This setup in our simulation represents the approach where an existing local transaction manager (i.e., a BTM) can be used in conjunction with the HTM to process HiCoMo transactions. The HTM has two input queues: the HiCoMo Manager Queue (HMQ) and the HiCoMo Feedback Queue (HFQ). The HMQ contains the HiCoMo transactions that are issued by a mobile node and are to be reintegrated into the base nodes. The HFQ contains the feedback from the BTM regarding the execution results of base transactions issued to the BTM from the HTM. The BTM has a single input queue which is the Base transaction Manager Queue (BMQ). The BMQ includes the base transactions issued from the HTM as well as local base transactions generated on the local node.

The results of processing in the HTM are one of the following: HiCoMo commit, HiCoMo abort, base transaction generation. The commit and abort of HiCoMo transactions will initiate the generation of new HiCoMo transactions to be introduced into the simulation and are put back into the HMQ. The base transactions that are generated by the HTM are put into the BTM. The result of the processing performed in the BTM are: base transaction from HiCoMo commit, base transaction from HiCoMo abort, local base transaction commit, and local base transaction abort. The former two cases are fed back into the HMQ. The latter two cases result in the generation of new base transactions and are put into the BMQ.

6.2. Methodology

The evaluation of the HiCoMo transaction processing is performed based on a benchmark that represents an inventory management scenario. The base node contains a large table of inventory information while the mobile nodes contain aggregate information about the data existing on the base node. Some constraints exist within the system such as the quantity of a part should not become a negative value, or the maximum and minimum quantities for a part is specified. These constraints will cause the abort of transactions. An example of such tables containing the inventory information are shown in figure 2.

During our experiments we compare the commit ratios of the HiCoMo transactions with the well-known two-tier transactions. The pure two-tier transaction model is suitable for processing replicated data on the mobile node. We have modified the assumptions of the mobile environment to accomodate a new set of applications targeted at processing aggregate data maintained on the mobile unit. In order to compare the commit ratio with the two-tier transaction model which was designed without the consideration of this application, we need to transform the HiCoMo transaction into a equivalent flat transaction representing the two-tier transaction and reintegrate into the base data. This method uses a pure flat transaction processing mechanism, while our HiCoMo transaction processing uses nested transactions. The HiCoMo transactions generated within the simulator are saved while running the HiCoMo transaction processing based on our model, and are again executed using the equivalent transformations for the two-tier transaction model. The commit ratios obtained during these two runs are compared. The mobile transactions are composed of 80% updating the average, 10% updating the sum, and 5% each updating the minimum and maximum values.



Part #	AVG
P222	100
P506	200
P1011	120

Part #	SUM
P222	2000
P506	1600
P1011	600

Part #	MIN
P222	50
P506	100
P1011	80

Part #	MAX
P222	400
P506	500
P1011	250

Part #	Location	Quantity
P1011	Florida	100
P1011	New York	200
P222	California	50
P506	California	300
P222	Georgia	250
P506	New York	100
⋮	⋮	⋮

Figure 2. Example of the inventory management benchmark.

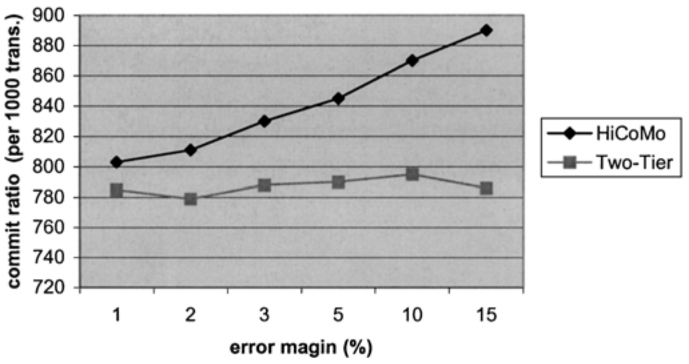


Figure 3. Comparison of the commit ratio while varying the error margin.

6.3. *Experiment 1: Varying the error margin*

The error margins that are provided when issuing the HiCoMo transactions were varied and the commit ratios were compared against the two-tier transaction. As the HiCoMo includes the retry semantics using error margin considerations, the error margin allowed in the HiCoMo transaction affects the commit ratio. On the other hand, the two-tier transaction is a flat transaction which does not consider error margins nor retry semantics. Therefore, the error margin gives the HiCoMo transaction a better possibility of committing. The error margins were varied from 1% to 15% of the update value specified. The results are shown in figure 3, and indicate that as the error margin increases the commit ratio increases. The figure shows the commit ratio against 1000 HiCoMo transactions issued on the mobile node. The base table size was fixed as 1000 tuples.

6.4. *Experiment 2: Varying the base table size*

The size of the base table was varied to compare how the table size will affect the processing of transaction commit ratios on aggregate data when using HiCoMo transaction processing in comparison with a simple extension to the two-tier transactions. As the table size grows, the aggregate data on the mobile unit represents a large amount of data. An update on the aggregate data transforms into an update on a large table. The two-tier transformation to a flat transaction increases the possibility of the constraint violation. The HiCoMo transaction also will result in a larger conflict possibility, but has more room than a flat transaction due to the fact that it uses a transformation mechanism that reduces the conflict with error margin consideration and individual subtransaction aborts and regeneration. Figure 4 shows the result of varying the base table size from 100 tuples to 100000 tuples. Although both show that the commit ratio declines as the base table size increases, the HiCoMo transaction processing is not as drastic as the extended two-tier transaction mechanism due to its flexibility. This was experimented with the error margin fixed at 10%.

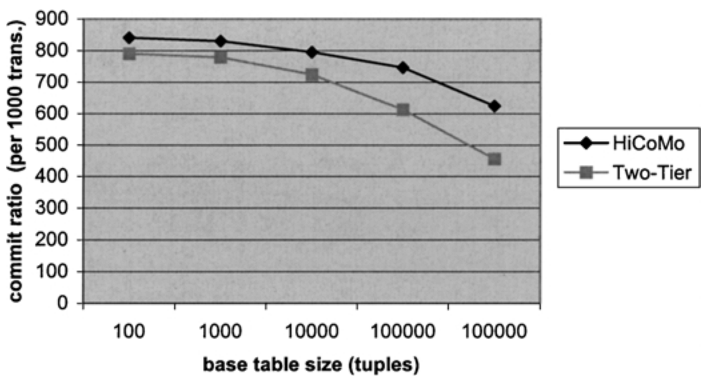


Figure 4. Comparison of the commit ratio while varying the base table size.

7. Further improvements and issues

There are several issues that can further improve the proposed approach.

First, the *partitioning granularities of the base table* will affect the performance of the algorithm. If the base tables are not partitioned, then in our case only a single base transaction will be generated from the HiCoMo transaction. This will create the danger of a high abort rate due to the large granularity of the sub-transaction. Therefore, tuple level sub-transaction generation schemes may also be investigated. But it should be well understood that the finer the granularity, the more sub-transactions are generated. The large number of generated sub-transactions will create again a large transaction processing overhead. The granularity tradeoff is an important factor in the performance of the algorithm.

Second, conflicts between base transactions and HiCoMo transactions may be reduced by employing some advanced mechanisms which have not yet been fully explored in our work. Adding *vertical partitioning* may help resolve conflicts between base transactions and HiCoMo transactions if they modify different fields. Another way is that although the conflicts may exist, the HiCoMo transaction may not just abort but take into consideration its own updates together with the updates performed by the base transactions and incorporate it into the aggregate update value. This may although need special logging of the base transaction value updates and complicates the consistency management. Because actually there is no control over what updates a base transaction had performed.

Third, merging of histories may be provided instead of generating transactions on the base tables. This would be beneficial if a large number of aggregate copies of the base tables are carried around and need to reduce the transaction processing overhead at the base tables. But, we assume right now that this kind of summary data is mostly carried around by only a few executives of companies etc.

8. Conclusion

In this paper, we have introduced a new type of mobile transactions we call *HiCoMo* transactions, that is expected to be very useful in the mobile computing environment. HiCoMo Transactions allow for aggregate data carried on mobile hosts to be updated during disconnection and to reintegrate the update operations performed on the aggregate data into the base tables existing in the fixed network. As these updates are aggregate updates, several base transactions which can bring the same effect upon the base tables need to be generated. We have provided a formal definition for the concepts related to HiCoMo's, including a transformation algorithm. Future work will focus on reducing the conflict between the base transactions and HiCoMo's, and will examine the effect of granularities of partitioning base tables in enhancing the extended nested transaction processing.

References

1. R. Chow and T. Johnson, Distributed Operating Systems and Algorithms, Addison-Wesley: Reading, MA, 1997.
2. M. Dunham, A. Helal, and S. Balakrishnan, "A mobile transaction model that captures both the data and movement behavior," ACM-Baltzer Jounal on Mobile Networks and Applications (MONET), vol. 2, no. 2, pp. 149–162, 1997.
3. P. Evaggelia and B. Bharat, "Revising transaction concepts for mobile computing," in Proceedings of the First IEEE Workshop on Mobile Computing Systems and Applications, 1994, pp. 164–168.
4. J. Gray, P. Helland, P.O' Neil, and D. Shasha, "The dangers of replication and a solution," in Proc. Of ACM-SIGMOD Int'l Conf. on Management of Data Montreal, Canada, 1996, pp. 173–182.
5. A. Helal, B. Haskell, J. Carter, R. Brice, D. Woelk, and M. Rusenkiewicz, Any Time, Anywhere Computing: Mobile Computing Concepts and Technology, Kluwer Academic Publisher: Boston, MA: 1999.
6. P. Liu, P. Ammann, and S. Jajodia, "Incorporating transaction semantics to reduce reprocessing overhead in replicated mobile data applications," in Proceedings of the International Conference in Distributed Computing Systems, Austin, Texas, 1999.
7. Q. Lu and M. Satyanaranyanan, "Isolation-only transactions for mobile computing," Operating Systems Review, vol. 28, no. 2, 1994.
8. M.T. Ozsu and P. Valduriez, Principles of Distributed Database Systems, Prentice Hall: Englewood Cliffs, NJ, 1991.
9. E. Pitoura and G. Samaras, Data Management for Mobile Computing, Kluwer Academic Publisher: Boston, MA, 1998.
10. G. Walborn and P.K. Chrysanthis, "Supporting semantics-based transaction processing in mobile database applications," in Proceedings of the Fourteenth Symposium on Reliable Distributed Systems, 1995.