Handling Mobile Transactions with Disconnections Using a Mobile-Shadow Technique

Osman Mohammed Hegazy
Department of Information Systems
Faculty of Computers and Information
Cairo University, Egypt.
ohegazy@idsc.net.eg

Ali Hamed El Bastawissy
Department of Information Systems
Faculty of Computers and Information
Cairo University, Egypt.
alibasta@hotmail.com

Romani Farid Ibrahim
Department of Information Systems
Faculty of Computers and Information
Cairo University, Egypt
rfarid1@yahoo.com

Abstract

In this paper, we introduce a Mobile-Shadow technique (M-Shadow), as an enhanced shadow paging technique to handle mobile transaction processing and disconnections. It makes use of a new notation called actionability, which differentiates the actions to be taken during the transaction’s validation phase according to the types of affected attributes. The M-Shadow technique raises reasonably the potential success probability of transactions processed under optimistic concurrency techniques.

Keywords: Concurrency Control, Mobile Database, Transaction, Shadow paging, Saga, Caching, Compensation

1. Introduction

Mobile transaction is a transaction performed with at least one mobile host takes part in its execution [4]; also, it may be defined with perspective of its structure as a set of relatively independent (component) transactions, which can interleave in any way with other mobile transactions [5]. The mobile user, by nature, is moving from one place to another so the mobile transaction should follow the user anywhere, which is not supported in distributed database transactions.

We view a transaction as a program in execution in which each write-set satisfies the ACID properties [1], and the program that updates the database as a three folds module (phases): reading phase, editing phase, and validation and write phase. The main question we attempt to answer in this paper is, if the data on the primary server has been changed while the mobile unit (MU) is disconnected or working offline, how can the transaction continue its work?

The proposed M-shadow technique is an optimistic concurrency technique constructed on the shadow paging technique that is used in deferred database recovery and other OS techniques. Shadow paging technique uses two copies of data items, the shadow copy (original), and the edited copy (current). When a transaction commits, the edited copy becomes the current page, and the show copy is discarded, otherwise, the edited copy is discarded and the shadow copy is reinstated to become the current page once more.

In this article, we illustrate a survey of related efforts in section 2, section 3 is dedicated to describe the important points we considered to propose the new technique, however the description of the M-Shadow technique, its relative usages, and advantages are detailed in section 4, and we conclude this work and think about the future in section 5.
transaction. It is intended to undo the visible effects of a previously committed transaction, e.g., cancel car is the compensating transaction for rent car. A problem lies in the fact that compensation does not reserve database consistency [8]: for example, suppose that the account initially has $X$, and then a withdrawal transaction of $Y$ (where $X \geq Y$) is executed and that the transaction will be compensated later. If another transaction commits applying an interest rate on the balance before the compensation has been performed (i.e., when the account has $(X - Y)$). The interest transaction was applied on a kind of dirty data, and therefore database consistency will not be preserved.

Most of the papers assume rarely changing data (Insurance data, Patients data, etc); the mobile unit has replica or caching subsystem. And, the mobile replica is logically removed from the master copy of the object and is only accessible by the transaction on the mobile unit [9], so that they do not consider the case of changing data on the primary server while the transaction processing. In addition, they assume long disconnection or working offline and do not consider short disconnection case.

### 3. Important Considerations

In optimistic methods using shadows, transactions are dependent on all data items by the same degree. A minor change in an item is sufficient to abort a transaction handling hundreds of actions on thousands of data items. Consequently, the probability of a transaction to fail is very high. This failure probability increases with the increase of the number of data items, the disconnection time, and the number of concurrent transactions. This is why the shadow technique is not frequently used in transactions management.

The m-shadow technique we propose; offers a solution for the preceding problem and gives the opportunity to widely manage transactions of difficult types such as long and/or mobile transactions. In m-shadow technique, transaction's validation is not tightly coupled to the eventuality of encountering modifications (done by other transactions) on the values of one or more of its data items.

In this section, we describe the important points we considered to design our technique for handling mobile transaction with disconnection. Which are: the processing of the transaction with and without disconnection, the available mobility adaptation strategies, the enterprise constraints acceptance range, and the effects of attributes types on the transaction behavior (actionability).

#### 3.1 Transaction Processing With and Without Disconnections [2]

In table 1, T1 without disconnection and T1 with disconnection, they start from the same initial state $S_0$ and pass through the intermediate states without losing any state, the sequence is the same until the state $S_m$ which is the last state before disconnection and the initial state after disconnection. Both go through the states from $S_{m+1}$ to $S_n$ which is the final state, so both are equivalent. Our goal is to build a technique for mobile database transactions to prove this analysis.

<table>
<thead>
<tr>
<th>Transaction T1 Without Disconnection</th>
<th>Transaction T1 With Disconnection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state: $S_0$, $S_1$, $S_2$, ...</td>
<td>Initial state: $S_0$, $S_1$, $S_2$, ...</td>
</tr>
<tr>
<td>Intermediate State: $S_m$, ...</td>
<td>Intermediate State: $S_m$, ...</td>
</tr>
<tr>
<td>$S_{m+1}$, $S_{m+2}$, ...</td>
<td>$S_{m+1}$, $S_{m+2}$, ...</td>
</tr>
<tr>
<td>$S_n = \text{final}$</td>
<td>$S_n = \text{final}$</td>
</tr>
</tbody>
</table>

Table 1. Transaction Processing with and without Disconnection

#### 3.2 Mobility Adaptation Strategies

Applications for mobile computing require adaptation for best performance under such variable conditions, to make best use of available resources without assuming the minimum set [10]. It must react and dynamically reassign the responsibilities of client and server. In other words, mobile clients must be adaptive [11].

The range of strategies for adaptation is delimited by two extremes. At one extreme, adaptation is entirely the responsibility of individual applications. While this laissez-faire approach avoids the need for system support, it lacks a central arbitrator to resolve incompatible resource demands of different applications and to enforce limits on resource usage. It also makes applications more difficult to write. The other extreme of application-transparent adaptation places entire responsibility for adaptation on the system. This approach is attractive because it is backward compatible with existing applications; they continue to work when mobile without any modifications. Between these two extremes lies a spectrum of possibilities that is referred to as application-aware adaptation (collaborative).
3.3 Enterprise Constraints Acceptance Range

Enterprise constraints also known as business rules, which are additional rules specified by users or database administrators that the data must satisfy. Updates to entities may be constrained by enterprise rules governing the real world transactions that are represented by the updates [12]. For example, sales applications have a rule sold-amount ≤ amount-on-stock that should be satisfied at all times.

Usually, enterprise constraints include relational operators as (<> , < , > , ≤ , ≥), which has a range of values that the data-item can be changed within it. For example, in the above constrain, sold-amount value can be assigned any value from a range of values which ≤ amount-on-stock value.

By using this property of enterprise constraints, in addition to, the characteristics of the attributes, we can build an algorithm that allows transactions to continue their works even if the shared data items at the primary server have been changed. So that we avoid roll-backing of transactions, if the changes are within the acceptable range of the data-item.

3.4 Actionability and Transactions Behavior

In m-shadow technique, transaction's validation is not tightly coupled to the eventuality of encountering modifications (done by other transactions) on the values of one or more of its data items. Transaction behavior at run time depends on some characteristics of its set of data items. We use the new notion actionability to describe how a transaction behaves if a value-change is occurred on one or more of its attributes during its processing time and by other transactions. Other than Key attributes (K), actionability classifies the data items used by a transaction into three types: change-accept, change-aware, and change-reject.

Change-Accept (A): Any attribute retrieved during the read phase to complete and explain the meaning of the transaction. If it is potentially changed (by another transaction) while the transaction is processing, it does not have any effect on the transaction behavior.

Change-Reject (R): This type of attributes is subject of periodical changes (e.g., Currency values, Tax rates, etc.). The value of such attribute remains constant for long period. But once it is changed during the transaction life time (by another transaction), it affects severely the transaction behaviour.

Change-Aware (W): The type of attributes is subject to change more frequently by different concurrent transactions. A modification on the value of this type of attributes may be accepted if the new value still in the acceptance range. Otherwise, the transaction aborts.

The previous three types of attribute actionability are to be declared for each transaction type. If omitted, the complete set of attributes will be handled as Change-Reject type (the default actionability type), a case in which the M-Shadow works like the traditional Shadow technique. Also, they are retrieved at the read phase to be edited and to apply the function of the transaction on it. It is also important to note that a transaction may generate a new data item (G) as a function of the three previous types of attributes. The M-Shadow technique handles these calculated attributes exactly as before:

- If a Change-Reject attribute(s) is modified during the transaction processing, the complete transaction aborts.
- But else, if a Change-Aware attribute(s) is the modified attribute and the changes are within the acceptance ranges, the transaction is recalculated and continues, otherwise it aborts.
- But else, if a Change-Accept attribute(s) is the modified attribute, the transaction continues and writes values.

<table>
<thead>
<tr>
<th>Change in</th>
<th>Integrity Constraints Violation</th>
<th>T Succeed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change-Accept Attribute(s)</td>
<td>Change-Reject Attribute(s)</td>
<td>Change-Aware Attribute(s)</td>
</tr>
<tr>
<td>Y / N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Y / N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Y / N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Actionability Truth Table

Table (2) illustrates the applied validation rules. If the Change-Accept attribute is changed or not, it doesn't have any effect on the transaction behavior that updates the Change-Aware attributes. Also, Change-Accept attributes are very rarely changing attributes, for example, item-description, employee-name; Birth-Date, etc., are approximately fixed value attributes. Table (2) illustrates the applied validation rules. If the Change-Accept attribute is changed or not, it doesn't have any effect on the transaction behavior that updates the Change-Aware attributes. Also, Change-Accept attributes are very rarely changing attributes, for example, item-description, employee-name; Birth-Date, etc., are approximately fixed value attributes.

**Rule:** If T1, T2 are concurrent transactions, T1 changes a shared Change-Reject attribute and T2 changes a shared Change-Aware attribute that belong to a normalized database then:

- If T1 commits before T2 then T2 must abort.

1 NA means Not Available
• If T2 commits before T1 then T1 can continue its processing.  

The reasons behind using the actionability include:
• A transaction usually update a part of the data set it uses, the other part of the data elements is asked by the transaction to control the transaction. These data items are read only items and a change in such elements should not prevent the execution of the transaction.
• Our concern is on the transactions that update Change-Aware attributes, which have acceptable range. An encountered change in these attributes may affect the outcomes of the transaction but not aborts entirely its execution.
• The usage of mobile transactions is still limited to salesman and inventory applications which are, by nature, applying short transactions with little attributes. This fortunately complies well with the M-Shadow concept.

3.4.1 Actionability advantages. The classification of attributes according to the actionability type has the following advantages:
• The DBMS can perform the update process automatically based on the actionability type of the data for all applications.
• A reasonable increase in the succeeded transactions ratio.
• This technique can be useful for some types of real time applications.

4. Mobile Shadow (M-shadow) Technique Description

We assume that the system is partially replicated distributed database system, because it is the most practical environment. We also assume that the mobile unit has a software package that can contact with the primary server and send and receive data from it. We classified the computers that are involved in the update transaction into two groups:
• The basic group: consists of primary site and mobile unit. They are enough to complete the transaction.
• The complementary group: consists of all the remaining sites (replicas) that are involved in the update operation and we assume using lazy replication protocol for refreshment.

The Mobile-Shadow technique (M-Shadow) is based on the shadow paging technique and it is basically designed to solve the problem of disconnection in a compound transaction (CT) which consists of two subtransaction groups, one is an independent group that is a collection of independent subtransactions (independent case), and the other is a dependent group that is a collection of vital subtransactions (dependent case). There is no dependency relationship between these two groups, but the compound transaction can include any number of groups.

4.1 Description of Validation Test

The validation test compares the original values of some data-items with its current values on the primary server, which succeeds in three cases:
• No change, which means that the original values are equal to the current values on the primary server.
• Constrained change, which means that some Change-Aware attributes has been changed by other transactions during disconnection (working offline) time but still these changes within the integrity constraint acceptance range.
• Insignificant change, which means that some Change-Accept attributes has been changed by other transactions during disconnection (working offline) time or during the execution of the transaction, but these Change-Accept data-items does not affect the current transaction.

The validation test fails in the following two cases:
• Significant change, in which we detect that some Change-Reject data items have been changed during the transaction processing and/or disconnections.
• Out-of-Constraints change, in which we detect that one or more Change-Aware data items have been updated in such a way that the global changes put the stored values out of the acceptance ranges.

The applications that use the Mobile-Shadow technique (M-Shadow) can be classified as:
• Laissez-faire Applications. In which, the validation and write phase of the M-Shadow technique is implemented as stored procedures at the primary server. In this case, the programmer writes the validation and write phase procedures for every application according to the semantic of the application and by using the names of the attributes, not its properties.
• Collaborative applications. In which, the validation and write phase of the M-Shadow technique is implemented as a part of the DBMS, it requires modification in the current existing databases structure. Every attribute should has a new property called Actionability type, which can be used by the DBMS or the application program to perform validation test. Also, programming languages that handle database operations as (VB.Net, PL/SQL, Transact-SQL, etc.) should support looping not only on the tuple level but also on the attribute level. Another useful implementation for collaborative applications can be achieved when the programmer determines the actionability type for the set of attributes as part of the application and passes it to the DBMS. The advantages of this implementation are: it doesn't require change in
the database structure and increase the technique flexibility.

Actionability can be declared while creating or altering the table attributes. But this way imposes a static actionability in which an attribute will react the same way in all applications and transactions types. Imposing dynamic actionability is by declaring the actionability type of each used data item for each transaction type. Actionability declaration is something similar to data types description in any program. One data element may be of different actionabilities when it is used by different transaction types.

4.2 Summary of the M-Shadow Technique Steps

1. Retrieve the current dataset from the primary server (Reading phase)
2. Copy the retrieved dataset as a shadow copy.
3. The user edits the dataset on the shadow copy [modify, add, delete] (Editing phase)
4. Send the original read-set, the edited-set (shadow copy changes), the read-query and, and the update query to the primary server (subtransaction by subtransaction).

At Primary Server Side:

5. Implement the validation and write phase:
   - Call validation-write-1 procedure (as a part of the DBMS) or
   - Call validation-write-2 procedure (as a stored procedure at the primary server).

(a) Independent Case.

6. If one subtransaction fails (disconnection, integrity constraints, etc.)

At Primary Server Side:

- Discard the current write-set subtransaction.

At Mobile Unit side:

- Removes the subtransaction shadow data-set from the shadow copy.
- Send next subtransaction data to the primary server.
- Short disconnection: Try to reconnect.
- Long disconnection: The program saves the data-sets (the original data-set and the shadow data-set) as XML files on the mobile unit secondary storage.

When reconnection with the primary server is available

After short disconnection:

The program resends the write-set data for the subtransaction, which the disconnection happened through its update only. The primary server restarts the write-set subtransaction as in step 5.

After long disconnection:

The program loads the XML files and starts a new independent write-set group transaction for the loaded data-sets (original and shadow) as in step 4.

(b) Fully Dependent Case.

7. If one subtransaction fails:

At Primary Server Side:

- Rollback the current and all the previous write-set subtransactions of the group.

At Mobile Unit side: because of

- Integrity constraints violation: Drops its data-sets and clears the memory to start a new transaction.
- Short disconnection: Try to reconnect.
- Long disconnection: The program saves the data-sets (the original data-set and the shadow data-set) as XML files on the mobile unit secondary storage.

When reconnection with the primary server is available

After short disconnection:

The program reissues the dependent-write-set group transaction as a new transaction as in step 4.

After long disconnection:

The program loads the XML files and starts a new fully dependent write-set group transaction for the loaded data-sets (shadow and original) as in step 4.

Validation-Write Procedure-1 (A General Validation Algorithm to Be Put as a Part of the DBMS)

Validation-Write-Phase (Record original, Record shadow, String read-query, String update-query)

In what follows we show the core functions of the technique, which use the actionability rules to perform the validation test. Its inputs are original data-set, shadow dataset (shadow-rec), read-query, update query, and the actionability types for attributes if they are not declared while tables creation. If the validation test succeeds, the transaction commits, otherwise the transaction aborts.

Aware-Update (integer flag)

For each change-reject-attribute(i) in shadow-rec

If Current.R(i) <> Shadow.R(i) then
   Flag = -1
   Goto par-out
End if
Next-For

For each change-aware-attribute(i) in shadow-rec

$\Delta W(i) = Shadow.W(i) - Original.W(i)$

$Current.W(i) = \Delta W(i) + Current.W(i)$

If (check-constraints(current.W(i) ) = False ) then
   Flag = -2
   Goto par-out
End if
Next-For
Par-out:
Return (flag)
The following two examples describe how the validation and write phase can be applied in different cases. The first example shows a sales transaction that decreases a stock quantity by 50 units. The second example shows a bank transaction that transfers $400 from account X to account Y. We use the notations of actionability, K denotes the Key attribute, A denotes a Change-Accept attribute, R denotes a Change-Reject attribute, W denotes a Change-Aware attribute, G denotes a generated attribute, and the subindexes o denotes the original value, s denotes the shadow value and c denotes the current value at the primary server.

### Table 3. Sales transaction (Example-1)

<table>
<thead>
<tr>
<th>Read-Phase:</th>
<th>10, Ashraf, 5000, 20, 3000</th>
</tr>
</thead>
<tbody>
<tr>
<td>K(X), A(X), W(X)o, K(Y), W(Y)o</td>
<td></td>
</tr>
<tr>
<td>Edit-Phase:</td>
<td>10, Ashraf, 7000</td>
</tr>
<tr>
<td>F1(5000) = 400</td>
<td></td>
</tr>
<tr>
<td>F2(400) = 400</td>
<td></td>
</tr>
<tr>
<td>5000 - 400 = 4600</td>
<td></td>
</tr>
<tr>
<td>3000 + 400 = 3400</td>
<td></td>
</tr>
<tr>
<td>Validation and Write Phase:</td>
<td>10, Ashraf, 4600, 20, 3400</td>
</tr>
<tr>
<td>Validation Test for account X: Current Value at Primary Site: K(X), A(X), W(X)</td>
<td>10, Ashraf, 7000</td>
</tr>
<tr>
<td>( \Delta_{W(X)} = W(X)_o - W(X) )</td>
<td>-400 = 4600 -5000</td>
</tr>
<tr>
<td>( W(X) = W(X)<em>o + \Delta</em>{W(X)} )</td>
<td>6600 = 7000 -400</td>
</tr>
<tr>
<td>If(check-constraints(W(X))) then</td>
<td>check-constraints(6600) = True</td>
</tr>
<tr>
<td>Accept W(X)o, G</td>
<td>Accept 6600, F1(5000), -400</td>
</tr>
<tr>
<td>Commit (t)</td>
<td>Commit (t)</td>
</tr>
<tr>
<td>Else</td>
<td></td>
</tr>
<tr>
<td>Rollback (t)</td>
<td></td>
</tr>
<tr>
<td>End if</td>
<td></td>
</tr>
<tr>
<td>Validation Test for account Y: Current Value at Primary Site: K(Y), W(Y)</td>
<td>20, 2000</td>
</tr>
<tr>
<td>( \Delta_{W(Y)} = W(Y)_o - W(Y) )</td>
<td>400 = 3400 -3000</td>
</tr>
<tr>
<td>( W(Y) = W(Y)<em>o + \Delta</em>{W(Y)} )</td>
<td>2400 = 2000 + 400</td>
</tr>
<tr>
<td>If(check-constraints(W(Y))) then</td>
<td>check-constraints(2400) = True</td>
</tr>
<tr>
<td>Accept W(Y)o, G</td>
<td>Accept 2400, F2(-400), 400</td>
</tr>
<tr>
<td>Commit (t)</td>
<td>Commit (t)</td>
</tr>
<tr>
<td>Else</td>
<td></td>
</tr>
<tr>
<td>Rollback (t)</td>
<td></td>
</tr>
<tr>
<td>End if</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4. Transferring amount from account X to account Y (Example-2)

<table>
<thead>
<tr>
<th>Validation Test:</th>
<th>25 : 25</th>
</tr>
</thead>
<tbody>
<tr>
<td>If ( R_c &lt;&gt; R_o ) then</td>
<td>Rollback (t)</td>
</tr>
<tr>
<td>Else</td>
<td></td>
</tr>
<tr>
<td>( \Delta_{W(X)} = W(X) - W_o )</td>
<td>-50 = 750 -800</td>
</tr>
<tr>
<td>( W(X) = W_o + \Delta_{W(X)} )</td>
<td>550 = 600 -50</td>
</tr>
</tbody>
</table>

4.3. Why Validation-Write Phase should be run in Exclusive-Lock Mode?
The following analysis describes the problems of using shared lock.

(a) Deadlock 1 – A New Type of Deadlock

Time | T1 | T2 | T3 ... | Tn
---|---|---|---|---
1 | Read-lock (x) |  |  |  
2 | Read (x) | Read-lock (x) |  |  
3 | Validate (x) | Read (x) | Read-lock (x) |  
4 | Write-lock(x) | Validate (x) | Read (x) |  
5 | Wait | Write-lock(x) | Validate (x) |  
6 | Wait | Wait | Write-lock(x) |  
7 | Wait | Wait | Wait |  
N | Wait | Wait | Wait | Wait

Table 5. Deadlock (1)

This problem decreases the performance of the system, especially in real-time applications; it can cause harmful effects to the system.

In table (6), T1, T2 read the value of x concurrently. T1 changes the value of x, T2 adds x (before change) to y and commit. The result of T2 is wrong because it reads x before change and it has write-lock on y not on x. T1 reads y after change (which include the old value of x) and adds the new value of x to y, so x has two different values (10, 15) in T1 which is wrong.

From the previous analysis, we can say that, the validation-write-phase must be run in exclusive-lock mode.

(b) Related Data Problem

<table>
<thead>
<tr>
<th>T1 example</th>
<th>T2 example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-lock(x)</td>
<td>Read-lock(x)</td>
</tr>
<tr>
<td>Read(x)</td>
<td>Read(x)</td>
</tr>
<tr>
<td>x = x + 5</td>
<td>Read-lock(y)</td>
</tr>
<tr>
<td>Read-lock(v)</td>
<td>Read(y)</td>
</tr>
<tr>
<td>Read(v)</td>
<td>Read(y)</td>
</tr>
<tr>
<td>y = y + x</td>
<td>y = y + x</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>v = v + x</td>
<td>Write-lock(y)</td>
</tr>
<tr>
<td>write-lock(v)</td>
<td>Write(y)</td>
</tr>
<tr>
<td>write(v)</td>
<td>commit</td>
</tr>
<tr>
<td>read-lock(y)</td>
<td></td>
</tr>
<tr>
<td>read(y)</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>y = y + x</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>write-lock(y)</td>
<td></td>
</tr>
<tr>
<td>write(y)</td>
<td></td>
</tr>
</tbody>
</table>

Table 6. Related Data Problem

7. It differentiates between the short disconnection and the long disconnection.
8. Management of the unpredictable disconnection.
9. No storage lost on the primary server or on the mobile unit, because after the transaction committed or rolled back, the program deletes the XML files.
10. The locking of data-items at the primary server is very short, only at the validation and writing time, which increase the performance of the system.
11. The load on the primary server would be more lite.
12. More control over the network disconnection, especially in wireless networks which its property is frequently disconnection.
13. All ACID properties are reserved in the dependent case, and semantic ACID properties are reserved in the independent case.
14. This technique decreases the deadlock rate, or approximately, avoids the deadlock problem, because the locking of data-items at the primary server is very short and does not use shared lock. Therefore, it increases the performance of the system.

The limitations of the M-Shadow technique are: it is designed for commercial applications that have a few shared data-items among transactions and the validation test is not suitable for some real-time applications.

We implemented a sales application that uses the M-Shadow technique using Visual Basic .Net and SQL Server 2000 because they support many new features as...
writing and reading XML files. We assume that the replication handling is solved as a distributed database problem using the lazy replication technique among fixed hosts.

5. Conclusion and Future Works

The mobile computing and moving objects area is very interesting and a rich area for research. In this paper, we introduced a Mobile-Shadow technique (M-Shadow) to handle mobile transaction processing. Shadow technique is the most suitable technique for mobile transactions, but it suffers from one big disadvantage which is the expected high transaction-failure probability. In M-Shadow we increase the transaction success probability, this by consequence, raises the performance of the system.

Actionability classifies the data elements handled by a transaction according to how much a change on these elements affect the transaction behaviour. It doesn’t transfer logs or transaction history among sites and it isn’t based on compensation concept. It differentiates between short disconnection and long disconnection. It decreases the programming time for applications. So, it is suitable for handling mobile transaction with disconnection.

Finally, we described why validation and write phase should be run under exclusive lock.

Future research will extend this work to support complex business applications that include a big number of shared data items and complex computations, and parallel processing and real-time environments. Also, what is the optimal solution for selecting the next server in a shared area among many servers to decrease the number of disconnection?

References


