# A Framework for Two Phase Reconciliation in Mobile Databases

Md. Abdur Rahman, Souhail Abdala and Abdulmotaleb El Saddik, *Senior Member, IEEE*

*Abstract*— **With the advent and increasing popularity of mobile devices, new challenges arise in mobile databases. A key challenge is to deal with the reconciliation phenomena when a mobile client reconnects after being disconnected. Widely employed mobile databases use nowadays an extended client-server architecture which suffers from some drawbacks such as scalability; security and traffic congestion and bottleneck around the server. We will introduce a novel three tier architecture for mobile databases to overcome the above mentioned shortcomings. We will also provide two phase reconciliation protocol along with the necessary algorithms to clarify the working principles of the proposed mobile database architecture.**

## I. Introduction

Mobile databases are becoming popular as the use of portable devices such as Laptops, PDAs and Smart phones are increasing. One common characteristic of mobile devices is that they remain disconnected or weakly connected most of the time. Due to this inherent property of these devices, the key aspect of mobile database systems is their ability to handle disconnections. Disconnection refers to the situation when a mobile system is unable to communicate with the server. In such a situation, mobile clients no longer have access to the appropriate shared data. In [1], Phatak and Badrinath focused their research on this issue and illustrated possible conflict resolution and reconciliation techniques with suitable algorithms using a multiversion server database to handle the reconciliation problem. The basic multiversion mobile database architecture implemented an extended client-server model which suffered from some performance problems such as bottleneck and traffic congestion around the server; scalability; and security. To overcome the above mentioned drawbacks we have introduced a three tier mobile database architecture by adding a middle tier between mobile clients and the server database to the architecture proposed by Phatak and Badrinath [1]. In the new proposed architecture, the middle tier is called Group Replica Server (GRS). We also proposed a two phase commit/reconciliation protocol along with an appropriate algorithm to describe the conflict resolution during reconnection after a disconnection. We will also explain the protocol to handle the attempt to disconnect, remain disconnected and reconnect. We organized the remaining of the paper as follows: Section 2 describes the previous and related works. The three tier database architecture and the two phase reconciliation protocol are described in Section 3. Section 4 draws the conclusions and possible future work.

Md. Abdur Rahman is with the Multimedia Communication Research Laboratory, 800 King Edward, Ottawa, Ontario, Canada, K1N 6N5 (Tel: (613) 562-5800 x 6248; Fax: (613) 562-5175; e-mail: rahman@mcrlab.uottawa.ca).

Souhail Abdala is with the Multimedia Communication Research Laboratory, 800 King Edward, Ottawa, Ontario, Canada, K1N 6N5 (Tel: (613) 562-5800 x 6248; Fax: (613) 562-5175; e-mail: sabdala@mcrlab.uottawa.ca).

Abdulmotaleb El Saddik is the director of Multimedia Communication Research Laboratory, 800 King Edward, Ottawa, Ontario, Canada, K1N 6N5 (Tel: (613) 562-5800 x 6277; Fax: (613) 562-5664; e-mail: abed@mcrlab.uottawa.ca).

## II. Previous Work

Multiversion reconciliation for mobile databases [1] presents an optimistic replication approach to deal with the disconnection problem of mobile clients which is allowed to locally replicate shared data and to operate on this data while it is disconnected. The local updates can be propagated to the rest of the system upon reconnection. However, since local updates potentially conflict with other updates in the system, some form of conflict detection and resolution is required. In the client-server architecture presented in [1] and [4], the primary copies of all data items are stored on the server. All transactions must commit on the server to be “globally" committed. The clients are able to locally replicate a subset of the current database state (i.e. the set of all committed versions of all data items currently in the database). Local transactions on the client can operate on this local replica and perform updates. As long as the client is connected to the server, each local transaction is automatically serialized to the server before it is allowed to commit. A transaction that cannot be serialized, because it has read dirty data [2] from another non-serializable transaction, must be rolled back. Otherwise, the transaction is globally committed and its updates applied to the shared database.

The research in [1] only focused on Centralized Mobile Server Database systems. The authors in [6] discussed a reconciliation process which permits cost effective recovery while minimizing roll back costs. In [7] a multiversion

concurrency control in object oriented system was discussed.

A distributed multiversion and two Phase-Locking concurrency control mechanism is used in another approach [8]. The results presented in [8] may not be used to analyze the performance of concurrency control mechanisms for mobile databases. However locking phenomena, although ensures consistent transaction, tends to limit simultaneous concurrent database transactions.

C. Gollmick [14] proposed a client oriented replication service for mobile Client/Server relational database environments. His model however does not support the targeted distributed architecture as it will be discussed in this paper.

Nitin et al. [15] discussed a modified two-tier replication model for concurrency control in mobile database systems. The proposed model imposes a constraint on the amount of change in local replica to ensure reduced transactions' commit time and number of rejections. There is also no discussion about the scalability issues that need to be addresses in distributed mobile databases.

YanSheng and XiongKai [16] presented two methods to improve the performance of disconnected operation. They discussed the issue of submitting offline transactions by probability and transferring transaction logs in group to reduce transaction failure and replica logs upload's time respectively.

The above mentioned works provide useful information related to the multiversion reconciliation schemes as well as conflict resolutions and replication models in two tier database architecture, however they have some basic limitations [3] [5]. They did not investigate in the issue of distributed mobile databases, they did not consider security models to detect and avoid threats and they suffer from server performance such as network congestion server power to process multiple client requests. Recognizing the potential of previous researches and their limitations, we decided to investigate the distributed nature of mobile databases along with suitable protocols and algorithms to support that model.

## III. Proposed Model and Protocol

### A. Database Model

To overcome the limitations mentioned above, we introduced the three tier architecture shown in Figure 1. The middle tier has a similar functionality to a proxy server. In our proposed architecture clients can be grouped together according to some given criteria such as region; data access; and application query. Replicas are fragmented among Group Replica Servers (GRSs) in such a way that each GRS host the replica relevant to that client group. Clients can connect directly to the GRS via wired or wireless link (to support mobility) to download the needed replicas when they want to disconnect. Each GRS has a global schema which tells the name of the GRSs holding the fragment of the same data object or item. If clients pose any query not found in their local GRS, a request is sent to the main server. Server then communicates with the GRSs holding relevant fragment via network at the expense of reduced performance. Detailed surveys [3, 9, 10, 11, 12, 13] regarding fragmentation and allocation of database objects show that fragments/replicas can be located in the GRSs in such a way that most of the time each client will have his data available in local sites. GRS asynchronously communicates via a fixed communication channel with the main database.

The proposed database architecture to support mobile clients ensures some noticeable advantages over the older models. Client transactions are not stopped, even if the main database is down as each client has access to the local GRS. With the help of GRS, mobile clients are able to synchronize their data with the main server which is now merely a database repository and does not have to handle all the clients' transactions.

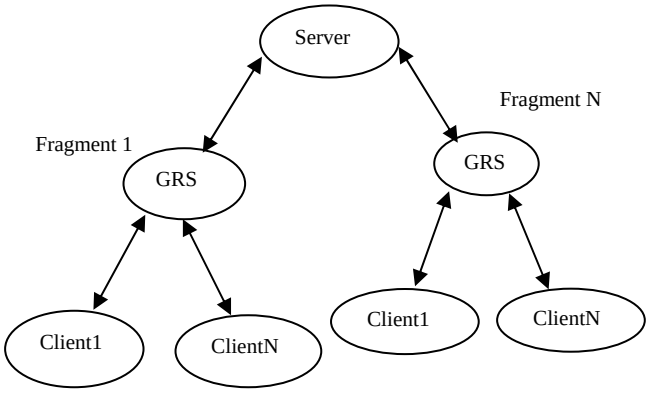


Fig. 1.Three tier architecture of the Mobile database.

The main server, however, has only to communicate with GRS's and run second phase reconciliation algorithm and protocol in case the first phase reconciliation in the local GRS fails to commit due to the fact that the data accessed by the client is shared by other GRSs. GRS can be optimized for client oriented replication and processing. Thus server processing load is reduced.

Since clients do not have direct access to the main server database, security threat is reduced. Indeed, both GRS and the main server have their own firewalls to ensure secure transactions. It is therefore easy to add more GRS between the server and the clients for scalability issues.

### B. The Two Phase Reconciliation Protocol

Figure 2 presents a composite state diagram for the three tier architecture. The proposed modified reconciliation method for the three tier architecture will be further discussed and described in more details.

When a client tries to commit, it sends its transaction to the local GRS for the first phase reconciliation. In fact, there may be two (2) cases in this scenario:

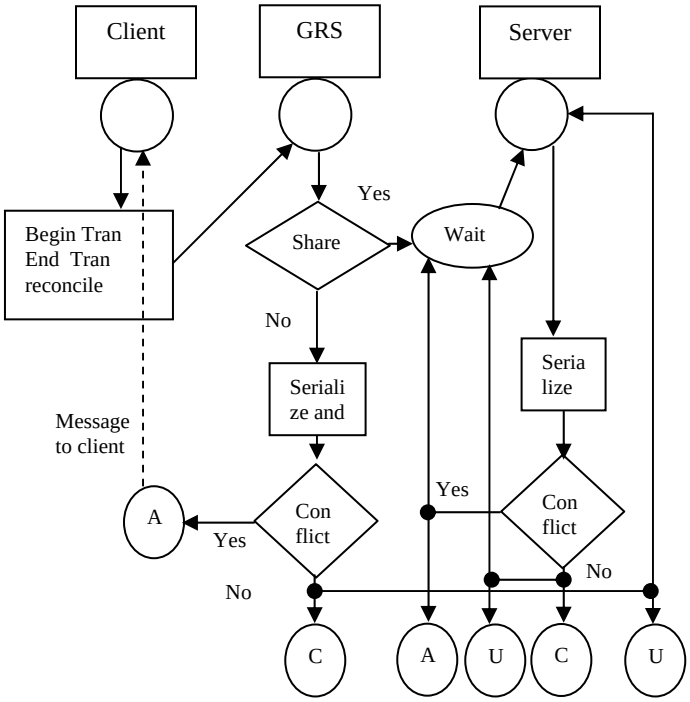


Fig. 2.Two Phase Commit protocol actions for mobile databases.

Case1: If the data items accessed by the client are non-shared i.e. present only in the local GRS, then the first phase reconciliation process as described in [1] is initiated at the local GRS. If a client transaction can be successfully reconciled at a GRS, it sends an asynchronous message to the main database server to only update its database state. No reconciliation process needs to be carried out in the main server as the transaction is non-shared. Indeed, it has been already reconciled at the local GRS. If the GRS cannot reconcile, it aborts the transaction, notifies the client of the status but does not contact the main database server about the aborted client transaction, thus reducing the load on the server.

Case2: If the data accessed by a specific client is shared which means that the data is also present in other remote GRS fragments, then the commit command must go through conflict resolution phase at the main server. In this case the first phase reconciliation is done at the local GRS. If the first phase reconciliation process is successful, the transactions are sent to the main database server for the second phase reconciliation immediately, and the reconciliation algorithm as shown in listing 3 is called at the server site. The server then sends messages to the GRSs which hold the shared data for reconciliation. After getting successful or abort message from all the GRSs, the server notifies the originator GRS. When the client is again connected with his GRS, he gets the notification of all his transaction.

Figure 2 portrays the functionality of our proposed algorithm and describes the interactions between the different tiers. The Client starts by downloading the replica from the appropriate GRS. Then it begins the transaction process. Once the transaction is ended the client reconciles with its corresponding Group Replica Server (GRS). The GRS functions according to the algorithm presented in Listing 1.

```
If message from client=”reconcile” Then
      If accessed data=”non-shared” Then
            Call Reconciliation algorithm
            If Reconciliation successful Then
               Update main server asynchronously
            Else
               Send Abort notification to client
            End If
       Else
      Send Reconcile message to main Server
            Wait (message from server)
            If Reconciliation is successful Then
               Update GRS Database state
               Send Successful message to client
            Else
               Send Abort message to client
            End If
      End If
 End If
```

Listing 1: Reconciliation algorithm in GRS for a mobile client

The main database server provides data update to a GRS upon request according to the algorithm presented in Listing 2.

```
If message from GRS =”successful reconciliation” Then
      Update server database state
   Send Successful message to GRS
      Else
   Send Abort notification to GRS
      End If
End If
```

Listing 2: The main database update algorithm when client accesses non-shared data items present in his connected GRS.

The main database server uses the algorithm in listing 3 when any mobile client tries to reconcile in his connected GRS with any shared data item that is located at others GRS’s.

```
If data = “shared” Then
If message from GRS =”reconcile” Then
   Send the data item to GRS’s holding the data items
   If that/those GRS can reconcile Then
         Update server database state
   Send Successful message to originator GRS
   Else
   Send Abort notification to originator GRS
  End If
End If
End If
```

Listing 3: The main database update algorithm when any mobile client accesses shared data items not present in his connected GRS.

### C. Client States

To cope with offline transactions, we assume that mobile clients remain in one of the three states. The first state represents the preparation for disconnection. The second state is the offline transactions on the local replica, while the third state is the reconnection with the GRS to send the offline transactions to the GRS to be reconciled.

*1) The first or preparing for disconnection stage*

A typical mobile client consists of a Client Front End (CFE) and an application that handles client transactions. The CFE handles the caching of objects from the GRS and the transaction process. The application operates on cached objects at the CFE and commits transactions through the CFE to the GRS. The client needs to prefetch replica objects into the cache by processing queries specified by the application in order for the disconnection process from the GRS to work properly. The application may need a special prefetch query to ensure that all the relevant data are cached at the client. Application specific hoarding queries are used to prepare for clients disconnection. A hoarding query is an operation on the persistent objects in the GRS that causes objects to be fetched or hoarded from the GRS into the client cache prior to disconnection.

*2) The second or the disconnection stage*

Once the client is disconnected from the GRS, an application will attempt to commit transactions as it normally would while connected. In the disconnection state, a commit becomes an offline commit meaning that it could potentially be aborted by the GRS upon reconnection. These operations will change the state of cached replica at the CFE. The CFE logs offline transactions in a specific transaction log. This log contains enough state per offline transaction information allowing each of the transactions to replay once the CFE is reconnected with the GRS. The application gives an ID to each offline transaction to be associated with the operations performed during that transaction. This information can be used later to assist the user in recovering from an abort.

The client CFE maintains transaction information on a per transaction basis. This information, also known as commit sets, consists of the Read Set (RS), the Modified Set (MS), and the New Set (NS) created during a transaction. However, while disconnected, these sets are maintained in the offline transaction log. In an offline transaction the RS may contain both persistent objects and objects that are persistent while the system is offline. We differentiate between persistent and offline-persistent object in following manner: an object is offline-persistent if it was created by some offline transaction but not yet committed to the GRS and an object is persistent if the mobile client is online and accesses any data item online. This also applies to the MS in an offline transaction: it can have both persistent and offline-persistent objects.

Figure 3 depicts two transactions: OT1 and OT2, of a mobile client while he is disconnected. In case of the Offline Transaction OT1, a client reads the data items x, y, z, l, m and n which are shown as Read Set RS1. During disconnection, the client also performs some offline transactions on the data items x and y as shown by the Modified Set MS1. OT1 also creates some new data items m, n, o, p and q in this transaction. The Offline Transaction OT2 contains an object "o" in Read Set RS2 inherited from OT1 which we call offline-persistent from OT1. In an offline-transaction, commit sets must have all of their references to objects before they are sent to the GRS. Temporary object references are assigned to objects that are created by offline transactions. When OT2 is stored into the offline transaction log, references to "o" must be updated to the correct temporary object reference assigned to it in OT1. To be able to handle the abort of an offline transaction, each offline transaction in the log must also save the state of each object in the MS prior to its first modification. Objects in the MS may be modified multiple times. It is noted that only the initial state of the object before any modifications is saved in the log and that only the state after the final modification in the duration of that transaction is saved in the MS.

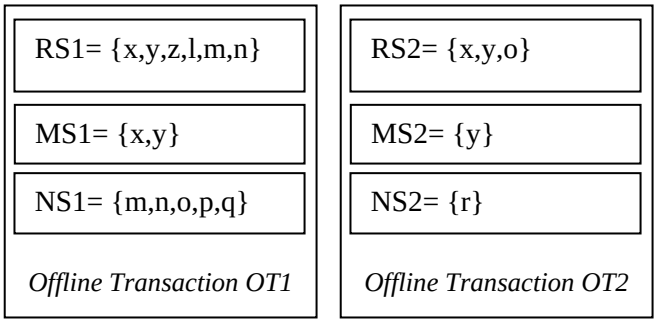


Fig. 3. An extract from the disconnected client transaction log

*3) The third or the reconnection stage*

When the CFE reconnects to the GRS, the synchronization of the transaction log occurs before the CFE can proceed with any reconciliation operation. The synchronization process with the GRS consists of replaying each offline transaction in the order in which they were committed while disconnected. Before sending a commit request to the GRS, it is necessary to update temporary object references in the NS to permanent object references, which are assigned by contacting the GRS. In addition, the MS and NS may contain temporary object references for offline-persistent objects and these must be updated as well. Once all the updates are done, the CFE sends to the GRS a commit request containing the commit sets stored during the offline transaction.

## IV. Conclusions and Future Work

In this paper we have introduced a new mobile database architecture with the necessary two phase reconciliation

protocol and algorithm. This novel architecture intends to minimize the performance problems such as bottleneck and traffic congestion around the server and scalability by introducing a middle tier between client and server database. In our approach, mobile clients no longer directly communicate with the main server, rather they first reconcile at the Group Replica Server. For the most of the cases, the GRS is able to reconcile the disconnected-client transaction if the data items or objects accessed by the client are non-shared i.e. only present in the local GRS. Upon successful reconciliation in GRS, GRS then asynchronously updates its data to the main server database. In this case, no reconciliation takes place in the server. If the mobile clients access a shared data item/object which is fragmented in other sites too, the first phase reconciliation takes place in the local GRS. The GRS then sends the client transaction to the main server for the second phase reconciliation as the transaction is dependent on the data items located on remote GRSs. This reconciliation process is referred to as “two (2) phase reconciliation”.

We have also discussed three client states namely preparing to disconnect, being at disconnection and reconnection along with appropriate steps to be taken in each state.

However, the proposed architecture still requires some future research work. First of all, two phase reconciliation protocol needs to go through comparison analysis with its counterpart single phase reconciliation protocol [1] to prove its excellence. Also, in order to measure the efficiency of the architecture, the framework should be implemented which is left for future work.

## REFERENCES


[1] S. H. Phatak and B. R. Badrinath, “Multiversion reconciliation for mobile databases” in *Proc. 15th International Conference on Data Engineering*, Mar. 1999, pp. 582–589.

[2] P. Bernstein, H. Berenson, Jim Gray, Jim Melton, Elizabeth O’Neil and Patrick O'Neil, “A Critique of ANSI SQL Isolation Levels,” in *Proc. ACM SIGMOD 95*, San Jose CA, June 1995, © ACM, pp. 1-10.

[3] M. T. Ozsu and P. Valduriez, “Principles of Distributed Database System”. Prentice Hall Inc., 1991.

[4] B. R. Badrinath and S. H. Phatak, “An architecture for mobile databases” Department of Computer Science Technical Report DCS-TR-351, Rutgers University, New Jersey, 1998.

[5] S. Ceri and G. Pelagatti, Distributed Databases – Principles and Systems, McGraw-Hill, 1984.

[6] P.C.J. Graham and K.E. Barker, "Effective Optimistic Concurrency Control in Multiversion Object Bases" Proceedings of the International Symposium on Object Oriented Methodologies and Systems (ISOOMS), Palermo, Italy, September 1994. In Springer-Verlag LNCS-858, pp. 313-328.

[7] A. Reza-Hadaegh, P.C.J. Graham and K.E. Barker, "An Architecture and Model for Processing Transactions in Multiversion Object Base Systems," in *Proc. 2nd Mid-Continent Information and Database Systems Conference (MIDAS'94)*, Fargo, USA, May 1994, pp. 99-112.

[8] A. Burger, V. Kumar and M. Hines, “Performance of Multiversion and Distributed Two-Phase Locking Concurrency Control Mechanisms in Distributed Databases.” Inf. Sci. 96(1&2): 129-152 (1997).

[9] S. Ceri, M. Negri, G. Pelagatti, “Physical database design: Horizontal data partitioning in database design,” in *Proc. 1982 ACM SIGMOD international conference on Management of data*, June 1982.

[10] S. Ceri, S.B. Navathe and G. Wiederhold, "Distribution Design of Logical Database Schemas," *IEEE Trans. on Software Engineering*, Vol. SE-9, No. 3, July 1983, pp. 487-504.

[11] S.B. Navathe and M.Y. Ra, "Vertical Partitioning for Database Design: a Graphical Algorithm," in *Proc. 1989 ACM-SIGMOD Int. Conf.*, Portland, OR, May 1989.

[12] S.B. Navathe and S. Ceri, "A Comprehensive Approach to Fragmentation and Allocation of Data," in IEEE Tutorial on Distributed Database Management (J.A. Larson, S. Rahimi, eds.), 1985.

[13] J. Shepherd, B. Harangsri, H. Chen, and A. H.H. Ngu “A two-phase approach to data allocation in distributed databases,” Fourth International Conference on Database Systems for Advanced Applications, Singapore, April 1995.

[14] C. Gollmick, “Replication in Mobile Database Environments: A Client-Oriented Approach,” in *Proc. 6th International Workshop on Mobility in Databases and Distributed Systems (MDDS 2003) at DEXA 2003*, Prague, September 2003.

[15] N. Prabhu, V. Kumar, I. Ray and G.C. Yang, "Concurrency Control in Mobile Database Systems," in *Proc. 18th International Conf. on Advanced Information Networking and Applications (AINA’04)*, Fukuoka, Japan, March 2004.

[16] L. YanSheng, S. XiongKai, “Improve Performance of Disconnected Operation through Submitting by Probability and Transferring Transactions in Groups,” in *Proc. 2003 International Conference on Computer Networks and Mobile Computing (ICCNMC’03)*, Shanghai, China, October 20 - 23, 2003.