

# A Novel Conflict Detection and Resolution Strategy Based on TLRSP in Replicated Mobile Database Systems\*

Zhiming Ding   Xiaofeng Meng<sup>†</sup>   Shan Wang<sup>†</sup>

*Institute of Computing Technology  
Chinese Academy of Sciences, Beijing China, 100080  
dingzhiming@263.net*

<sup>†</sup>*Institute of Data and Knowledge Engineering  
Renmin University of China, Beijing China, 100872  
{xfmeng, suang}@public.bta.net.cn*

## Abstract

*Replication is one of the key technologies in promoting the performance of mobile database systems. In this paper, a novel mobile database replication scheme, Transaction-Level Result-Set Propagation (TLRSP) model, is put forward. The conflict detection and resolution strategy based on TLRSP is discussed in detail, and the implementation algorithm is proposed. In TLRSP model, mobile users are allowed to access local replicas of the database and submit local transactions when system is disconnected. The locally committed transactions are sent to the fixed database server for conflict reconciliation and result-set incorporation when system is reconnected. TLRSP model uses the incremental refresh method to synchronize database replicas and maintains the consistency of the replicated mobile database system.*

## 1. Introduction

Advances in computer and telecommunication technologies have made mobile computing a reality. In the mobile computing environment, users can access information through wireless connections regardless of their physical location. In the last decade, this new kind of computing paradigm has gained great development. However, greater mobility implies a more tenuous network connection and a higher rate of disconnection [1]. In order to support disconnection operations as well as to improve the availability and reliability of the system, it is necessary to employ replication technologies in mobile database systems.

\* This research was supported by the grants of 863 High Technology Foundation and the Natural Science Foundation of China.

However, data replication can cause some problems, such as inconsistencies among different copies of the same data object. In order to ensure consistency, the replicated database system should behave as if there were only one copy of each data object. This feature is called one-copy-equivalence. A replicated database system is one-copy-serializable if it ensures both one-copy-equivalence and transaction serializability. One-copy-serializability is the correctness criterion of the replicated database system [2, 3].

An ideal replicated mobile database system should achieve four goals, that is, availability & scalability, mobility, serializability, and convergence [4], which implies that it should employ the asynchronous group replication method. In the asynchronous group replication, however, conflicts between different sites might be incurred and should be reconciled. Conflict reconciliation is one of the key problems in asynchronous group replication schemes.

In this paper, a new replication model, Transaction-Level Result-Set Propagation (TLRSP) model, is proposed. The conflict reconciliation strategy based on this model is analyzed and the detailed implementation algorithms are presented.

The remaining part of this paper is organized as follows. Section 2 presents a survey of related work. In section 3 we describe the mechanism of the TLRSP model and analyze its conflict reconciliation strategy. Section 4 describes the version number management and the log structure of TLRSP model. Section 5 provides the detailed implementation algorithms. Section 6 concludes the paper.

## 2. Related Work

Recently, a lot of research has been focused on the replicated mobile computing environment, and many models and algorithms have been proposed, such as the two-tier replication algorithm [4], the fault-tolerant quorum consensus scheme [5], BAYOU [6], and the three tier replication architecture [7].

In the two-tier replication scheme, the database is a collection of replicated data objects with the primary copies residing at certain sites known as object masters. The replicated mobile database system is two-tiered. The first tier consists of mobile nodes, which are frequently disconnected from the fixed network; the second tier consists of base nodes, which are steadily connected to each other through the fixed network. Transactions run on the first tier can access local copies of the database and can be committed in a tentative form. When the system is reconnected, the base transactions produced by the tentative transactions are sent to the base nodes to be reprocessed.

The two-tier replication algorithm is an effective replication scheme for the mobile computing environment. However, it can suffer from heavy reprocessing overhead in many circumstances. In order to reduce the overhead of two-tier replication, a semantics based incorporation algorithm is proposed in [8]. The basic idea is merging the tentative history into the base history so that substantial work of tentative transactions could be saved. In addition, Li Lin *et. al.* in [7] introduce broadcasting technologies into two-tier replication and put forward a three-tier replication architecture.

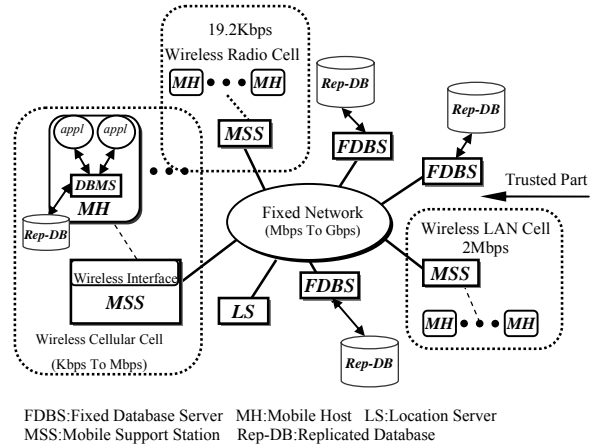
Most of the above work does not focus on the conflict reconciliation problem. Some approaches (such as BAYOU) that do focus on the conflict resolution problem require specialized knowledge of the system for all transactions and special concurrency control models[9], which limits their adaptability.

S. H. Phatak and B. R. Badrinath in [9] put forward an efficient conflict reconciliation model - multiversion reconciliation model, in which data objects have multiple versions at the database server. Mobile users are allowed to access local copies of the data objects and to submit transactions when the system is disconnected. When the system is reconnected, transactions are sent to the server for conflict reconciliation. Multiversion reconciliation model provides snapshot isolation level that does not ensure transaction serializability[10].

In order to solve the above problems, we propose the TLRSP model in this paper. Our aim is to make the mobile database system come closer to Jim Gray's four goals and to make it more adaptive by eliminating unnecessary requirements.

## 3. TLRSP Mobile Replication Model

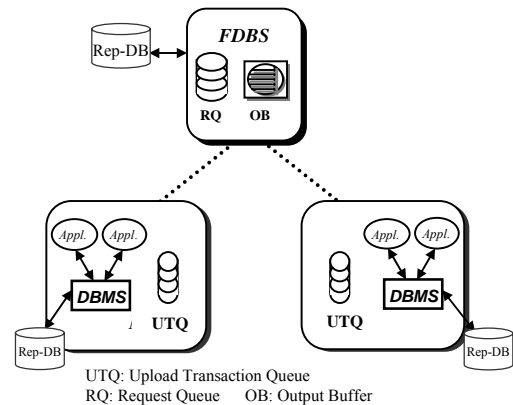
### 3.1 Mechanism of TLRSP Model



**Figure 1: Architecture of the replicated mobile database system**

Figure 1 presents a general architecture of the replicated mobile database system similar to that described in [5]. In this architecture, the trusted part is composed of the fixed network and the fixed hosts residing on it. Some of the fixed hosts, called MSS, are augmented with wireless interface to communicate with mobile hosts. Fixed hosts without wireless interface include LSs and FDBSs. Each FDBS keeps a replica of the database. Each MH keeps a replica of the database too, which is managed by the Embedded DBMS (EDBMS).

In the trusted part, FDBSs are steadily connected to each other through the fixed network. We can use the traditional replica control protocols such as ROWA to ensure consistency of this part. Thus all FDBSs can be combined into one logical entity. However, there can be inconsistencies among different MHs and between a MH and a FDBS, so the MHs can be viewed as two logical entities. The TLRSP model is shown in Figure 2.



## Figure 2: TLRSP replication model

In the TLRSP model, the FDBS ensures transaction serializability and ACID properties through Two-Phase-Locking mechanism. However, transactions locally committed at the MHs need to be verified at the FDBS before they can be globally committed. A MH can go through three different states:

### 1) Consistent State

At the instant of time when a synchronization process is over, all differences between the FDBS and the MH have been reconciled and they have exactly the same data objects. We call this scenario consistent state.

### 2) Accumulating State

The MH enters into accumulating state when it begins to update the local replica of the database and to locally submit transaction operations. In accumulating state, the MH can be disconnected from the FDBS. Local transactions are executed concurrently and serializability is ensured by EDBMS, that is, the concurrent transaction schedule is equivalent to a serial one that is called the Equivalent Serial Schedule.

Information of locally committed transactions, including their identifiers and their Read, Write, and Result Sets, is logged at the MH.

**Definition 1 (Read, Write, and Result Sets):** For any transaction  $T$ , we define three sets:  $ReadSet(T)$  consisting of all data objects that are read by  $T$ ,  $WriteSet(T)$  consisting of all data objects that are written by  $T$ , and  $ResultSet(T)$  consisting of values of the data objects in  $WriteSet(T)$  at the instant of time when  $T$  is committed.

### 3) Resolving State

When the MH is reconnected with the FDBS and starts a synchronization process, it enters into resolving state. For brevity, we suppose that there is no transaction run at the MH in this state.

In the resolving state, the MH first sends the locally committed transactions to the FDBS for conflict detection. Only the transactions that do not violate the one-copy-serializability can pass the verification. Next, the result sets of the passed transactions are incorporated, and the FDBS are updated accordingly. Finally, the recently updated data objects are read out from the FDBS and transmitted to the MH to refresh its local replica. After the synchronization process is over, the MH returns to the consistent state and its log is cleared.

## 3.2 Conflict Reconciliation Strategy of TLRSP Model

In resolving state, the MH first generates an Upload Transaction Queue ( $UTQ$ ) by reorganizing its log records. Transactions in the  $UTQ$  are listed in the same order as they are in the Equivalent Serial Schedule.

The synchronization process consists of two phases: the uploading phase and the downloading phase. In the uploading phase, the  $UTQ$  is sent to the Request Queue (RQ) of the FDBS. When this same  $UTQ$  is processed, the transactions in the  $UTQ$  are verified before they are globally committed, because they might have read out-of-date data items at the MH.

Suppose that there are  $n$  transactions in the  $UTQ$ , denoted by  $MT_1, MT_2, \dots, MT_n$ . Since the transaction schedule at the MH is serializable, it is safe to assume that  $MT_1, MT_2, \dots, MT_n$  are executed serially at the MH in the same order. When the FDBS begins to process the  $UTQ$ , it first computes the Access Set of the  $UTQ$ .

**Definition 2 (Access Set of  $UTQ$ ):** For any  $UTQ$ , assume that it contains  $n$  transactions, denoted by  $MT_1, MT_2, \dots, MT_n$ . The Access Set of the  $UTQ$  is defined as:

$$AccessSet(UTQ) = \bigcap_{i=1}^n (ReadSet(MT_i) \cup WriteSet(MT_i))$$

Next, the FDBS locks the data objects in  $AccessSet(UTQ)$ , and verifies  $MT_1, MT_2, \dots, MT_n$  serially. When verifying a transaction, the FDBS needs to determine whether the data seen by the transaction at the MH and at the FDBS are identical. To meet this requirement, we give the following definitions, and assume that when a transaction passes verification, its Result Set is written into the FDBS immediately.

**Definition 3**  $\zeta(Loc, T, X)$  denotes the value of data object  $X$  at site  $Loc$  at time  $T$ .

**Definition 4 (Readable Set):** Assume that  $MT_i (1 \leq i \leq n)$  is a transaction in  $UTQ$ , which begins to be run at the MH and to be verified at the FDBS at times  $t_i, T_i$  respectively. The Readable Set of  $MT_i$  is defined as:

$$ReadableSet(MT_i) = \{X \mid (X \in AccessSet(UTQ)) \wedge (\zeta(MH, t_i, X) = \zeta(FDBS, T_i, X))\}$$

**Theorem 1:** For any transaction  $MT_i (1 \leq i \leq n)$ , if

$$ReadSet(MT_i) \subseteq ReadableSet(MT_i)$$

then  $MT_i$  can be globally committed at the FDBS. That is,  $ResultSet(MT_i)$  can be written into the FDBS without violating the one-copy-serializability. Otherwise,  $MT_i$  cannot be globally committed.

**Proof:** According to definition 4, for transaction  $MT_i$ , if:

$$ReadSet(MT_i) \subseteq ReadableSet(MT_i)$$

then  $MT_i$  sees the same data at the MH and at the FDBS. According to the conclusion of [9],  $MT_i$  can be globally committed at the FDBS. Otherwise,  $MT_i$  can not be globally committed.

□

**Theorem 2:**  $ReadableSet(MT_i)$  is a subset of  $AccessSet(UTQ)$  which consists of the data objects that

have not been updated by the FDBS since it synchronized with the MH last time.

**Proof:** Assume that the time of last synchronization between the MH and the FDBS is  $t_0$ , and  $MT_i$  begins to be run at the MH and to be verified at the FDBS at times  $t_i$ ,  $T_i$  respectively.

According to Definition 4, we have:

$$\begin{aligned} & \text{ReadableSet}(MT_i) \\ &= \{X \mid (X \in \text{AccessSet}(UTQ)) \wedge \\ & \quad (\zeta(MH, t_i, X) = \zeta(FDBS, T_i, X))\} \end{aligned} \quad (1)$$

Since  $MT_i$  is the first transaction executed at the MH, the data objects at the MH have the same value at time  $t_i$  as they did at time  $t_0$ . That is, for  $\forall X \in \text{AccessSet}(UTQ)$ ,

$$\begin{aligned} \zeta(MH, t_i, X) &= \zeta(MH, t_0, X) \\ &= \zeta(FDBS, t_0, X) \end{aligned} \quad (2)$$

Combining (1) and (2), we have:

$$\begin{aligned} & \text{ReadableSet}(MT_i) \\ &= \{X \mid (X \in \text{AccessSet}(UTQ)) \wedge \\ & \quad (\zeta(FDBS, t_0, X) = \zeta(FDBS, T_i, X))\} \end{aligned}$$

In other words,  $\text{ReadableSet}(MT_i)$  is a subset of  $\text{AccessSet}(UTQ)$  which consists of the data objects that have not been updated by the FDBS since it was synchronized with the MH last time.

□

$\text{ReadableSet}(MT_i)$  can be computed through the comparison of primary version numbers ( see Section 4 ).

**Theorem 3:** If  $\text{ReadSet}(MT_{i-1}) \subseteq \text{ReadableSet}(MT_{i-1})$

$$\text{Then } \text{ReadableSet}(MT_i) = \text{ReadableSet}(MT_{i-1}) \text{ Y } \text{WriteSet}(MT_{i-1})$$

$$\text{Else } \text{ReadableSet}(MT_i) = \text{ReadableSet}(MT_{i-1}) - \text{WriteSet}(MT_{i-1})$$

**Proof:** Assume that  $MT_i (1 \leq i \leq n)$  begins to be processed at the MH and the FDBS at times  $t_i$ ,  $T_i$  respectively. Consider the times when  $MT_{i-1}$  is about to be executed at the MH and to be verified at the FDBS respectively. According to Definition 4, we know that in this scenario,  $\text{ReadableSet}(MT_{i-1})$  is composed of all data objects that have identical values at the MH and at the FDBS.

Now consider the times when  $MT_{i-1}$  has just finished execution and verification at the MH and at the FDBS respectively. In this scenario,  $\text{ResultSet}(MT_{i-1})$  has been written into the MH, but whether or not it has been written into the FDBS depends on the result of verification. There are two cases:

1. If  $\text{ReadSet}(MT_{i-1}) \subseteq \text{ReadableSet}(MT_{i-1})$

According to Theorem 1,  $MT_{i-1}$  has been globally committed, and its Result Set has been written into the FDBS. In this case,  $\text{ReadableSet}(MT_i)$  consists of:

a)  $\text{ReadableSet}(MT_{i-1}) - \text{WriteSet}(MT_{i-1})$ , which has not been touched by  $MT_{i-1}$ , and

b)  $\text{WriteSet}(MT_{i-1})$ , which has been written by  $MT_{i-1}$  at both sites.

Therefore:

$$\begin{aligned} & \text{ReadableSet}(MT_i) \\ &= (\text{ReadableSet}(MT_{i-1}) - \text{WriteSet}(MT_{i-1})) \text{ Y } \text{WriteSet}(MT_{i-1}) \\ &= \text{ReadableSet}(MT_{i-1}) \text{ Y } \text{WriteSet}(MT_{i-1}) \end{aligned}$$

2. If  $\text{ReadSet}(MT_{i-1}) \subseteq \text{ReadableSet}(MT_{i-1})$  is not true, According to Theorem 1,  $MT_{i-1}$  has not been globally committed. As a result,

$$\text{ReadableSet}(MT_i) = \text{ReadableSet}(MT_{i-1}) - \text{WriteSet}(MT_{i-1})$$

□

The Readable Set of each transaction can be computed according to Theorem 2 and Theorem 3. Meanwhile, transactions can be verified according to Theorem 1. When the verification process is over, transactions in the  $UTQ$  fall into two sets: the Commit Set and the Cancel Set. Previously we assumed that when a transaction passes verification, its Result Set is written into the FDBS immediately. For the sake of efficiency, however, the write operation to the FDBS is actually postponed until after all transactions have been verified in order that Result Sets can be incorporated. If more than one transaction in the Commit Set update the same data object, only the last write needs to be considered. In this way, at most one write operation is needed for each data object. Locks are released after the results are written into the FDBS.

Deletion is not considered in the above discussion. In asynchronous replication, deletion of the data object from tables with a foreign key definition may violate reference integrity [11]. In order to avoid this from happening, data objects of this kind are not really deleted but are specially labeled. Deletion of the data objects of other kinds can be dealt with in a way similar to Update. Data objects to be deleted at the FDBS are given a deletion label. When the data object with deletion label is replicated to the destination, the same data object is deleted.

## 4. Version Numbers and Log Structure of TLRSP Model

In TLRSP model, any data item at the FDBS has only one version, with a primary version number associated with it. Whenever the FDBS updates a data item, the primary version number of the data item will increase by

1. However, data items at the MH can have multiple versions. When a mobile transaction reads a data item, the latest version is returned, and when a mobile transaction updates a data item, a new version of the data item is generated. All versions of the data item are kept in the MH and are managed by the EDBMS.

In TLRSP model, every data item has a version number associated with it, with the form of "PVN[•SVN]". PVN is the primary version number required for every data item both at the FDBS and at the MH, while SVN is the secondary version number that is optional for data items at the MH.

When a new data item is generated at the FDBS, its PVN is set to 0 and its SVN is set to  $\langle null \rangle$ . When a new data item is created at the MH, its PVN is set to  $-1$  and its SVN is set to the identifier of the mobile transaction creating the data item.

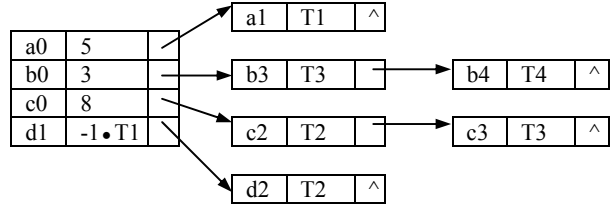
In consistent state, data items at the MH only have PVNs that are equal to the corresponding PVNs at the FDBS. In accumulating state, updating transactions at the MH label new data versions with their identifiers, leaving the PVNs unchanged. In this way, the system can determine whether a data item has been updated at the FDBS during accumulating state of the MH though comparison of corresponding PVNs.

The multiple versions of a data item are organized through the Multiple-Version-Link structure, which is used to generate *UTQ* when next synchronization process starts. Assume that there is a serializable transaction schedule at the MH which is shown in Figure 3.

SCHEDULE	T1	T2	T3	T4
$r_2[C]=c0$		$r_2[C]=c0$		
$r_2[B]=b0$		$r_2[B]=b0$		
$r_1[A]=a0$	$r_1[A]=a0$			
$r_3[B]=b0$			$r_3[B]=b0$	
$r_1[C]=c0$	$r_1[C]=c0$			
$w_1[A]=a1$	$w_1[A]=a1$			
$r_4[A]=a1$				$r_4[A]=a1$
$r_3[A]=a1$			$r_3[A]=a1$	
$w_2[C]=c2$		$w_2[C]=c2$		
$i_1[D]=d1$	$i_1[D]=d1$			
$c_1$	$c_1$			
$r_2[D]=d1$		$r_2[D]=d1$		
$w_3[C]=c3$			$w_3[C]=c3$	
$w_3[B]=b3$			$w_3[B]=b3$	
$r_4[B]=b3$				$r_4[B]=b3$
$c_3$			$c_3$	
$w_2[D]=d2$		$w_2[D]=d2$		
$c_2$		$c_2$		
$w_4[B]=b4$				$w_4[B]=b4$
$c_4$				$c_4$

Figure 3: A Serializable Mobile Transaction Schedule

The corresponding Multiple-Version-Link structure is shown in Figure 4.



1.  $X_0$  is the value of X in consistent state, while  $X_i$  ( $i>0$ ) is the value of X in *ResultSet* ( $MT_i$ )
2. PVNs of data items are only present in header nodes for they are kept unchanged during accumulating state

Figure 4: Structure of the Multiple-Version-Link

Storing transaction log at the MH in traditional ways is not appealing because of resource restrictions. TLRSP model uses a simplified log structure (see Figure 5). When the MH initiates a synchronization process, it generates *UTQ* (see Figure 6) from the transaction log and the Multiple-Version-Link structure.

TID	SN	ReadSet	WriteSet
T2	2	B,C,D	C,D
T1	1	A,C	A,D
T3	3	A,B	B,C
T4	4	A,B	B

Figure 5: Log Structure at the MH

TID	ReadSet	WriteSet	ResultSet
T1	A,C	A,D	a1,d1
T2	B,C,D	C,D	c2,d2
T3	A,B	B,C	b3,c3
T4	A,B	B	b4

Figure 6: Structure of UTQ

## 5. TLRSP Replication Algorithms

The TLRSP replication algorithms are mainly composed of two independent procedures: the uploading procedure and the downloading procedure. These two

procedures are executed at the FDBS as independent transactions.

The uploading procedure consists of three phases (see Algorithm1). In the initiation phase, the environment variables are initiated, and the related data objects are locked. In the conflict detection phase, transactions are verified one after another. In the third phase, result sets of the transactions that have passed the verification are incorporated, and the result values are written into the FDBS. Locks are released at the end of the procedure.

---

#### Algorithm1: TLRSP Upload Algorithm

---

Input:  $UTQ$  and  $AccessSet(UTQ)$   
*//initiation*

1. CommitSet=CancelSet= $\phi$ ;
2. ReadableSet=CommitWSet =  $\phi$ ;
3. For All  $X \in AccessSet(UTQ)$  do
4.   If ( $\exists MT \in UTQ \wedge (X \in WriteSet(MT))$ )
5.     Then XLock( $X$ );
6.     Else SLock( $X$ );
7.   Endif;
8.   If ( $PVN(X, MH) == PVN(X, FDBS)$ )
9.     Then ReadableSet=ReadableSet  $\cup \{X\}$ ;
10.   Endif;
11. Endfor; *//now ReadableSet == readableSet( $MT_i$ )*  
*//conflict detection*
12. For  $i=1$  to  $|UTQ|$  do *//|UTQ| is the number of transactions in UTQ*
13.   If  $ReadSet(MT_i) \subseteq ReadableSet$  then
14.     CommitSet=CommitSet  $\cup \{MT_i\}$ ;
15.     CommitWSet=CommitWSet  $\cup WriteSet(MT_i)$ ;
16.     ReadableSet=ReadableSet  $\cup WriteSet(MT_i)$ ;
17.   Else
18.     CancelSet=CancelSet  $\cup \{MT_i\}$ ;
19.     ReadableSet=ReadableSet -  $WriteSet(MT_i)$ ;
20.   Endif;
21. Endfor;
22. Unlock All SLock;

*//incorporate the result sets and update the FDBS*

23. For All  $X \in CommitWSet$
24.    $MT_{cx} =$  (the last transaction in CommitSet which updates  $X$ )
25.   Change the value of  $X$  to that in  $ResultSet(MT_{cx})$
26. Endfor;
27. Unlock All XLock ;
28. Transmit CancelSet to OB.

---

In the downloading procedure, the FDBS first scans its log file to determine the data objects that need to be downloaded according to the last synchronization time. Next, it applies for the locks of these data objects in order to ensure consistency. Finally, the data objects are read out and transmitted to the Output Buffer for the MH to fetch back later on to refresh its local database replica (see Algorithm 2).

---

#### Algorithm2: TLRSP Download Algorithm

---

Input:  $T_{Last}$ ; *//the time of last synchronization*  
 $LOG_{FDBS}$ ; *//log file of FDBS*  
 $UID$ ; *//User ID*

1. According to  $T_{Last}$  and  $LOG_{FDBS}$ , compute:  
FcommitSet= (transactions committed at FDBS since  $T_{Last}$ );
2. DownDataSet= $\phi$ ; *//data objects to be downloaded*
3. For All  $FT \in FCommitSet$  do
4.   DownDataSet=DownDataSet  $\cup WriteSet(FT)$ ;
5. Endfor;
6. For All  $X \in DownDataSet$  do
7.   Slock( $X$ );
8.   Read( $X$ );
9.   Transmit  $X$  and its value to OB;
10. Endfor;
11. Unlock All SLock.

---

## 6. Conclusion

We proposed a novel conflict reconciliation model, TLRSP, in this paper. By providing conflict reconciliation and allowing mobile users to access local replicas of the database, TLRSP model provides an efficient support for mobility and disconnection operations. Besides, through the use of incremental refresh method, TLRSP cuts down communication cost. The use of the Request Queue and the Output Buffer further reduces connection time, cutting down resource consumption and lowering the probability that errors could occur.

In addition, the TLRSP model ensures atomicity and serializability of transactions, overcoming the weakness of most asynchronous mobile replication schemes. Incorporation of the Result Sets reduces reprocessing overhead. Moreover, specialized knowledge of the system

is not needed in the synchronization process, which makes the model more adaptive.

## References

- [1] G.D. Walborn, P.K. Chrysanthis. Supporting Semantics-based Transaction Processing in Mobile Database Applications. In *Proceedings of the 14th Symposium on Reliable Distributed Systems*, Sep. 1995, pp.31-40.
- [2] G. Coulouris, J. Dollimore, et al. *Distributed systems concepts and design*, UK:Addison-Wesley, 1996, pp.312-349
- [3] A.K. Elmagarmid. *Database transaction models for advanced applications*. CA:Morgan Kaufmann, 1992, pp.16-19
- [4] J. Gray, P. Helland, et al. The dangers of replication and a solution, In *Proceedings of ACM SIGMOD*, Montreal, Canada, June 1996, pp.173-182.
- [5] S. Byun, S. Moon. Resilient data management for replicated mobile database systems, *Data & Knowledge Engineering* 1999, Vol.29, pp. 43-55
- [6] A. Demers, K. Petersen, M. Spreitzer, et al. The bayou architecture: support for data sharing among mobile users. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Dec. 1994, pp.2-7,
- [7] X.G. He, C.J. Tang, L. Li, et al. *Special Database Technologies*, Beijing:Science Press, 2000, pp.71-78
- [8] L. Peng, P. Ammann, S. Jajodia. Incorporating transaction semantics to reduce reprocessing overhead in replicated mobile data applications. In *Proceedings of ICDCS'99*, Austin, TX, USA, May 1999, pp.414-423.
- [9] S.H. Phatak, B.R. Badrinath. MultiVersion reconciliation for mobile databases, In *Proceedings of ACM SIGMOD*. San Jose, CA, USA, May, 1995, pp.1-10.
- [10] H. Berenson, P. Bernstein, J. Gray, et al. A critique of ANSI SQL isolation levels. In *Proceedings of ICDE'99*, Sydney, Australia, March 1999, pp.582-589.
- [11] R. Goldring. Things every update replication customer should know. In *Proceedings of ACM SIGMOD*, San Jose, CA, USA, May 1995, pp. 439-440