

# CONCURRENCY CONTROL FOR SYSTEM WITH DATA BROADCAST<sup>1</sup>

MEI-WAI AU, EDWARD CHAN AND KAM-YIU LAM

*Department of Computer Science  
City University of Hong Kong  
83 Tat Chee Avenue, Kowloon  
email: {csedchan/cskylam}@cityu.edu.hk*

Although data broadcast has been shown to be an efficient method for disseminating data items in mobile computing systems with large number of clients, the issue on how to ensure data consistency observed by mobile transactions has been largely ignored by researchers in the area. While data items are being broadcast, update transactions may install new values for the data items. If the executions of update transactions and broadcast of data items are interleaved without any control, the mobile transactions, which are generated by mobile clients, may observe inconsistent data values. In this paper, we propose a serialization checking method (SCM) for concurrency control between read-only mobile transactions and update transactions. SCM is based on the framework of an earlier algorithm, Update First with Ordering (UFO), but improves on that algorithm by reducing re-broadcast overhead when the probability of data conflict between updates and data broadcast is high.

## 1 Introduction

Recent advances in mobile communication technology have greatly increased the functionality of mobile information services and have made many mobile computing applications a reality. Various innovative applications, such as real-time traffic information and navigation systems, and real-time stock monitoring systems, are emerging [XSGFR97]. However, a number of technical hurdles need to be surmounted before these scenarios can materialize [AFZ95, IB94]. One of the most important issues is efficient dissemination of consistent data items to transactions from mobile clients [AFZ97, IB94]. In recent years, many efficient data dissemination methods have been proposed [AFZ97, DCKV97, LS97, P98, PC99, SNSR98, XSGFR97]. Most of them are based on data broadcast in which the broadcast server continuously broadcasts data items to mobile clients.

In data broadcast, mobile transactions do not need to inform the broadcast server before accessing a data item. They can get the data item from the “air” while it is being broadcast. However, if the updates to the database are done concurrently, the mobile transactions may observe inconsistent data values, e.g., the resulting execution schedule of the mobile and update transactions may be non-serializable.

---

<sup>1</sup> The work described in this paper was partially supported by a grant from the Research Grants Council of Hong Kong Special Administrative Region, China [Project No. CityU1097/99E] and partially supported by a grant from CityU (Project No. 7001069).

In this paper, we study the problem of disseminating consistent data items to read-only mobile transactions while allowing updates to be performed concurrently at the database server. We propose a *Serialization Checking Method (SCM)* to ensure the consistency of data items observed by the mobile transactions. Allowing concurrent execution of data broadcast and update transactions not only can improve the response time of mobile transactions, it can also maintain the freshness of the data items, since most of data items represent real-time information. The SCM is based on the conflict checking framework proposed in the *Update-First with Order (UFO)* algorithm, which has many important advantages for mobile computing systems [LAC99]. The organization of the remaining parts of the paper is as follows. Section 2 reviews the related work on concurrency control in data broadcast. Section 3 defines our transaction models for mobile computing systems with data broadcast. Section 4 discusses the problem of data inconsistency when update and data broadcast are performed concurrently. Section 5 reviews how UFO resolves the concurrency control problem. Section 6 introduces SCM and explains how it avoids a performance problem in UFO. Examples are provided to explain the mechanisms of the SCM. The paper concludes in Section 7.

## 2 Related Work

Mobile computing has been the subject of much research in recent years [AFZ95, DCKV97, LS97, P98]. Unfortunately the important issue of concurrency control in data broadcast has not received adequate attention. Traditional concurrency control protocols are not suitable to mobile computing systems due to their heavy overhead for detecting data conflicts in a mobile environment [P95, P98]. Owing to the relatively poor quality of services of mobile networks, it is not easy to ensure data consistency and to detect data conflicts in a mobile network. One suggested solution is to relax the consistency requirement. In [PB95], a two-level consistency model is proposed. Semantically related data are grouped together into a cluster, and the data inside a cluster are mutually consistent. However, a certain degree of inconsistency is allowed among data items at different clusters.

In [SNSR98], a control matrix is proposed for data conflict resolution. For a database of  $n$  data items, a matrix of size  $n \times n$  is used. For each broadcast cycle, the control matrix is broadcast together with the data items. A mobile client performs consistency checking using the matrix before reading any data item from the "air". This method can handle read-only transactions well as update transactions. The write operations are performed on local copies of the data items at the client. At the end of a transaction, the whole transaction including all of the read and write operations and the cycle numbers in which they are performed will be sent to the server for commitment.

In [P98, PC99], multi-version data broadcast approaches are suggested to resolve the problem of reading inconsistent data values. In addition to the most updated version, all the previous versions within a time frame need to be broadcast as well. Another method to detect the non-serializability problem is to broadcast serialization graphs [P98, PC99]. Each client maintains its local serialization graph to ensure that the schedules of all the committed transactions are serializable. The first drawback of this method is also the heavy overhead in broadcasting the

serialization graphs since every data conflict at the database server has to be broadcast. Secondly, each client must listen to the transmission channel to maintain and update its serialization graph. This leads to another serious problem, which can affect the correctness of this approach: the need to maintain the serialization graphs at the client mobile even when it is disconnected. The mobile network is unreliable and disconnection is frequent. When disconnection occurs, the mobile client cannot obtain new serialization information about its transaction, making it virtually impossible to ensure serializability of transaction execution.

### 3 System Model

The mobile computing system model used throughout this paper consists of a base station, a number of mobile clients and a mobile network such as the GSM cellular radio system, which provides limited bandwidth for data broadcast [H94]. The mobile clients, which represent users equipped with mobile machines, communicate with the base stations through low bandwidth wireless channels of the mobile network.

At the base station, a database server is maintained. The database contains various useful information about the external environment such as stock updates, current traffic information and news. Their values can be highly dynamic. To maintain the validity or “freshness” of the data items, update transactions are generated to refresh the data values whenever the status of the corresponding objects in the external environment has changed such as when there is a change in stock price. Each update transaction is time-stamped. The time-stamp is used to indicate at which snapshot the update is generated. It is recorded with the new value into the data item as its version number. It is assumed that stale data items will be much less useful.

In the model, it is assumed that update transactions are short transactions, consisting mainly of one to several operations. Some update transactions may also contain read operations. Furthermore, a well-formed concurrency control protocol, such as two phase locking [BHG87], is used for concurrency control among the update transactions at the database server.

The database server continuously broadcasts data items from the database one by one until the end of a broadcast cycle. The length of a broadcast cycle may be fixed or variable. Each broadcast cycle follows the preceding cycle immediately. The data items may require different time for broadcast due to different sizes. A broadcast algorithm is adopted for selecting data items to broadcast<sup>2</sup>. In the model, the broadcast process is modeled as a long read-only transaction called the *broadcast transaction (BT)*. The set of data items in a broadcast transaction consists of those data items, which are broadcast in the period from (current time – life span of a mobile transaction) to current time. The execution of the BT and the update transactions are interleaved to reduce the

---

<sup>2</sup> In the last few years, various broadcast algorithms based on the deadlines of the transactions and the access frequencies of the data items have been proposed [LS97, XSGFR97]. As we shall see later, our protocol does not depend on the broadcast algorithm selected and can be applied to any of these broadcast algorithms.

blocking delay due to the installation of update transactions. (Note that treating the broadcast process as a transaction is mainly to ease the discussion of the algorithm. It actually does not process any transaction features such as the ACID properties.)

The mobile clients issue transactions, called *mobile transactions (MT)*, to access data items at database server. It is assumed that each MT is a set of data requests (read operations). The requests can be performed in any order. When all the requests are satisfied, the transaction is completed. An example for this type of transactions is a request for the weather forecasts of a few cities at the same time or a request for several stock data. A single request is issued instead of several separate requests simply because batching the requests can reduce the system overhead. The arrival sequence of the data items is unimportant as long as all requested data items are received before the deadline.

Each MT is defined with a (soft) deadline. Although they are not real-time transactions, meeting their deadlines is an important performance objective. The requirement may be a statistical one such as 95% of the mobile transactions has to be completed within 5 seconds after the submissions. Transactions, which have missed their deadlines, might still be of some use, but beyond a certain time interval they would be considered totally useless. The period between the arrival time of a transaction and the time when it is considered to be useless is called *drop period*.

The assumptions of the system model are summarized below:

1. All update transactions are at the database server (base station).
2. The arrival rate of the update transaction is high due to the dynamic nature of the external environment.
3. All mobile transactions are read-only.
4. Mobile transactions can be *unordered*.
5. Each mobile transaction has a drop period. A mobile transaction will be aborted if it cannot get all the required data items from the broadcast program within the drop period.

#### 4 Sample Inconsistent Cases

In this section we briefly describe the data inconsistency problem in data broadcast and illustrate the nature of the problems with some representative examples.

##### **Example 1:** Data conflict between a MT and an update transaction

Suppose the update transaction, U, updates data item  $d_5$  and then data item  $d_2$ , and a mobile transaction, MT, wants to read  $d_2$  and  $d_5$ . If the schedule is:

- i) Broadcast transaction (BT) broadcasts  $d_2$
- ii) MT reads  $d_2$
- iii) U updates  $d_5$  &  $d_2$
- iv) Broadcast transaction (BT) broadcasts  $d_5$
- v) MT reads  $d_5$

The mobile transaction MT may observe inconsistent data values. The serialization graph is cyclic such as  $MT \rightarrow U \rightarrow MT$  and is non-serializable. The reason is that MT reads a data item,  $d_2$ , which is in conflict with U before the update from U and it reads a conflicting data item,  $d_5$ , after the update from U.

**Example 2:** A MT conflicts with two (or more) update transactions

Even though the serialization order between an update transaction and a mobile transaction is not cyclic, the final serialization graph can still be cyclic due to transitive dependencies<sup>3</sup>. Suppose there are two updates  $U_1$  and  $U_2$  such that  $U_1$  will update  $d_2$  and then  $d_1$ , and  $U_2$  will update  $d_1$  and then  $d_5$ . If the schedule is:

- i) Broadcast transaction (BT) broadcasts  $d_2$
- ii) MT reads  $d_2$
- iii)  $U_1$  updates  $d_2$  &  $d_1$
- iv)  $U_2$  updates  $d_1$  &  $d_5$
- v) Broadcast transaction (BT) broadcasts  $d_5$
- vi) MT reads  $d_5$

The serialization graph is cyclic such as  $U_2 \rightarrow MT \rightarrow U_1 \rightarrow U_2$ .

**Example 3:** Non-serializability involving two or more broadcast transactions.

A MT may not be able to get all its required data items from a single broadcast cycle. Non-serializability of transaction execution may occur over more than one broadcast transaction:

- i)  $BT_1$  broadcasts  $d_2$
- ii) MT reads  $d_2$
- iii) End of  $BT_1$
- iv) U updates  $d_5$  &  $d_2$
- v)  $BT_2$  broadcasts  $d_5$
- vi) MT reads  $d_5$

The serialization graph is cyclic such as  $MT \rightarrow U \rightarrow MT$ .

Note that in the determination of the serializability of a serialization graph, we ignore the broadcast transactions as they are considered as “intermediate transactions”. They do not have any real effect on database consistency. Their only function is to provide data items for the mobile transactions. (The reason of treating it as a transaction is to facilitate the analysis. Since the mobile transactions read data items from broadcast transactions, their serialization order will always be  $BT \rightarrow MT$ .)

The data inconsistency problem in the first two examples can be solved by adopting a *serial* execution method. For example, each transaction, either a broadcast or an update transaction, is defined with a unique time-stamp based on its arrival time. The execution order of the transactions follows their time-stamps. In this way, the final schedule will be serial. However, this method has

---

<sup>3</sup> If a transaction  $T_j$  reads an uncommitted data item from another transaction  $T_i$ ,  $T_j$  will be dependent on  $T_i$ .

three serious problems. Firstly, the concurrency of the system will be lowered. Secondly, the waiting time of update transactions can be very long as the broadcast transactions are long transactions. Blocking update transactions for long time may affect the “freshness” of the data items and the objective of broadcasting the most updated data values to mobile transactions may not be achieved. Thirdly, even if the serial approach is used, the problem in the third example above is still unresolved.

## 5 Update-First with Order (UFO)

The biggest problem in designing a concurrency control protocol for a mobile computing system using data broadcast is that the mobile transactions may read a data item at any time while the data item is being broadcast and the database server in general does not notice this. Thus, instead of detecting data conflicts between mobile transactions and update transactions, the UFO algorithm checks data conflicts between broadcast and update transactions [LCA99]. Although the UFO algorithm has been shown to give a good performance, its performance is highly affected by the probability of data conflicts between mobile transactions and update transactions since it resolves the data conflicts by data re-broadcast.

The basic principle of the UFO algorithm is to ensure that if a data conflict occurs between a broadcast transaction and an update transaction, the serialization order between them will always be  $U \rightarrow BT$ .  $BT \rightarrow U$  will be allowed only if it is impossible for a mobile transaction, which reads from  $BT$ , to be dependent on  $U$  directly or transitively. Since mobile transactions read data items from broadcast transactions, the serialization order is always  $BT \rightarrow MT$ . Thus, serialization order between the update transactions and the mobile transactions will always be  $U \rightarrow MT$  and the final serialization graph will be serializable [BHG87].

Basically, the UFO protocol consists of two parts:

1. Execution of update transactions; and
2. Conflict resolution between update and broadcast transactions.

### 5.1 Execution of Update Transactions

The execution of an update transaction is divided into two phases: the *execution phase* and the *update phase*. During the execution phase, the operations of an update transaction are executed and data conflicts with other update transactions are resolved using a conventional concurrency control protocol such as 2PL. The new values from the write operations are written in a private workspace of the transaction instead of updating the database immediately. When all the operations of the update transaction have been completed, it enters the update phase in which permanent updates of the database will be performed by copying the new values from its private workspace into the database. The data conflict with broadcast transaction will be checked in the update phase after the completion of the updates. (So, we can see that the update transactions adopt 2PL for resolving data conflicts with other update transactions and use an optimistic approach to detect conflicts with broadcast transaction.)

There are two important advantages in dividing the execution of the update transactions into two phases. Firstly, it can significantly reduce the blocking probability of the broadcast transactions. If a broadcast transaction wants to read a data item, which is locked by an update transaction, the broadcast transaction will be blocked until the update transaction is committed based on the principles of 2PL. At the same time, the update transaction, which is holding the lock, may be blocked due to data conflicts with other update transactions. Due to transitive blocking, the blocking time of the broadcast transactions can be very long. Dividing the execution of an update transaction into two phases can greatly reduce the blocking probability and blocking time of broadcast transactions since data conflicts between update and broadcast transactions will occur only when the update transaction is in the update phase, which is much shorter. Secondly, the detection of data conflicts between the update and broadcast transactions will become much easier. At the update phase, the system will know which data items have been accessed by the update transaction. By comparing the write set of the update transaction with the read set of the broadcast transaction, the system can determine whether there is any data conflict between them.

## 5.2 Conflict Resolution between Update and Broadcast Transactions

Data conflict between an update transaction and a broadcast transaction is detected when the update transaction enters its update phase. Re-broadcast is used to resolve the conflict. The purpose is to reverse the serialization order from  $BT \rightarrow U$  to our desirable order,  $U \rightarrow BT$ . The following defines the algorithm at the broadcast server. It is performed when an update transaction enters the update phase.

```

if  $O_{BT} \cap O_U = \{\}$ 
    BT and U have no dependency
else for each data item  $d_i \in \{O_{BT} \cap O_U\}$ 
    re-broadcast data item  $d_i$ 
where  $O_{BT}$  = set of data items of broadcast transaction, BT
       $O_U$  = set of data items of update transaction, U

```

By re-broadcasting the conflicting data item, the serialization order between the broadcast transaction and the update transaction is reversed. Noted that as described in Section 3, the set of data items in a broadcast transaction consists of those data items, which are broadcast in the period from (current time – the life span of a mobile transaction) to current time. Thus, the set of data items in the broadcast transaction is not fixed. After the broadcast of a data item, the data item will be included and the last data item in the broadcast transaction will be removed.

## 6 The Serialization Checking Method (SCM)

A performance problem of the UFO algorithm is that its performance is sensitive to the probability of data conflicts between update and broadcast transactions. If the conflict probability is high, the re-broadcast overhead will be quite significant leading to serious degradation in overall system

performance. In SCM, this problem is alleviated because the consistency of data values observed by a mobile transaction is ensured by performing a check on its local serialization graph instead of relying on a data re-broadcast.

As can be seen from Section 4, the main data inconsistency problem in a broadcast environment is that a data item captured from the air may later be updated by an update transaction. If the update transaction has further data conflict with the broadcast transaction, then it may cause a cyclic schedule in a mobile client. Another reason for a cyclic schedule is due to transitive dependencies among update transactions. To deal with these problems, mobile clients can be provided with data conflict information of update transactions to construct their local serialization graph. In case a cyclic schedule is formed, the mobile client will dispose the data item that is read before the update transaction in order to maintain a serializable schedule. To solve the first case, the SCM algorithm broadcasts the identities of the data items that are updated by a transaction at the database server if any of the data items are broadcast in the last drop period. To handle the case of transitive dependencies, SCM simply broadcasts information of the update transactions that has transitive dependencies with the broadcast data items.

#### 6.1 Algorithm for the Server

Similar to the UFO algorithm, the execution of the update transactions is divided into two phases and conflict between an update transaction and the broadcast transaction will be detected when the update transaction enters its update phase. In case of data conflict, the dependency between the update transaction and the broadcast transaction will be broadcast instead of re-broadcasting the conflicting items. At the same time, the dependencies between the update transaction and other update transactions in the last drop period will also be broadcast.

After finishing an update transaction  $U$ , the following algorithm is performed:

if  $D_B \cap D_U \neq \{\}$

then broadcasts message:  $D_U$  is updated

else if  $D_U \cap D_{U_i} \neq \{\}$  where  $U_i \in T_B$

then broadcasts message:  $D_U$  is updated

where  $D_B$  = set of data items that are broadcast in the last drop period

$D_U$  = set of data items of update transaction,  $U$

$T_B$  = set of update transactions in the last drop period

#### 6.2 Algorithm for Mobile Transaction

Mobile transactions listen to the broadcast transaction for their required data items. They also listen to the message on update and build their local serialization graphs. Once the serialization graph is updated, a searching on the graph will be performed. If a cyclic schedule is formed, the mobile transaction resolves it by disposing the data item that it captures before the update transaction that caused the conflict. The data items will be captured from the 'air' again when it is broadcast in the next time. The process continues until all the required data items are captured or the drop period expired.

loop until (read-set =  $D_{MT}$ ) or (drop period of MT has expired)  
if a data item  $d_i$  is broadcast  
then if  $d_i \in D_{MT}$   
then read-set = read-set  $\cup$   $\{d_i\}$   
else if an update transaction  $D_U$  is broadcast  
then if (read-set  $\cap D_U \neq \{\}$ ) or ( $\exists U_i \in T_S$  s.t.  $D_U \cap D_{U_i} \neq \{\}$ )  
then  $T_S = T_S \cup \{D_U\}$   
if a cyclic schedule is formed  
then disposes the data item get before the update transaction in conflict  
where  $D_{MT}$  = set of data items requested by MT  
 $T_S$  = set of update transactions in the local serialization graph

### 6.3 Examples

In the following, we use two examples to illustrate the mechanism of the SCM algorithm.

#### Example 1:

This example is on cycle schedule formed by transitive dependencies. There are two update transactions in this example,  $U_1$  and  $U_2$ .  $U_1$  will update  $d_2$  and  $d_1$  and  $U_2$  will update  $d_1$  and  $d_5$ . The mobile transaction, MT, reads  $d_2$  and  $d_5$ .

#### Steps

#### Serialization graph for MT

- |  |   |
|--|---|
| i) Server broadcasts $d_2$                                     | -   |
| ii) MT reads $d_2$   | MT  |
| iii) $U_1$ updates $d_2$ & $d_1$                               | MT  |
| iv) Server broadcasts message: $d_2$ and $d_1$ are updated     | MT  |
| v) MT reads message: $d_2$ and $d_1$ are updated               | MT $\rightarrow U_1$                                |
| vi) $U_2$ updates $d_1$ & $d_5$                                | MT $\rightarrow U_1$                                |
| vii) Server broadcasts message: $d_1$ and $d_5$ are updated    | MT $\rightarrow U_1$                                |
| viii) MT reads message: $d_1$ and $d_5$ are updated            | MT $\rightarrow U_1 \rightarrow U_2$                |
| ix) Server broadcasts $d_5$                                    | MT $\rightarrow U_1 \rightarrow U_2$                |
| x) MT reads $d_5$  | MT $\rightarrow U_1 \rightarrow U_2 \rightarrow MT$ |
| xi) MT detects a cyclic schedule and disposes $d_2$ to resolve | $U_1 \rightarrow U_2 \rightarrow MT$                |

In this example the server checks for transitive dependencies between the update transaction  $U_2$  and the broadcast data items at step vii). Since the server detects the dependencies, it broadcasts a message on the update transaction. So MT can construct its local serialization graph at step viii). When MT reads  $d_5$ , it detects that a cyclic schedule is formed at step x). MT resolves this by disposing data item  $d_2$  that it reads before  $U_1$ .

#### Example 2:

This example illustrates that different mobile transactions may observe update transactions in different serialization order. The mobile transaction  $MT_1$  reads  $d_1$ ,  $d_3$  and  $d_4$  while  $MT_2$  reads  $d_1$  and  $d_2$ . The update transactions  $U_1$  updates  $d_2$  and  $d_3$ ,  $U_2$  updates  $d_1$ .

Steps	Serialization graph for MT <sub>1</sub>	Serialization graph for MT <sub>2</sub>
i) MT <sub>1</sub> begins transaction	MT <sub>1</sub>	-
ii) Server broadcasts d <sub>1</sub>	MT <sub>1</sub>	-
iii) MT <sub>1</sub> read d <sub>1</sub>	MT <sub>1</sub>	-
iv) MT <sub>2</sub> begins transaction	MT <sub>1</sub>	MT <sub>2</sub>
v) Server broadcasts d <sub>2</sub>	MT <sub>1</sub>	MT <sub>2</sub>
vi) MT <sub>2</sub> read d <sub>2</sub>	MT <sub>1</sub>	MT <sub>2</sub>
vii) U <sub>1</sub> updates d <sub>2</sub> & d <sub>3</sub>	MT <sub>1</sub>	MT <sub>2</sub>
viii) Server broadcasts message: d <sub>2</sub> and d <sub>3</sub> are updated	MT <sub>1</sub>	MT <sub>2</sub>
ix) MT <sub>1</sub> and MT <sub>2</sub> reads message: d <sub>2</sub> and d <sub>3</sub> are updated	MT <sub>1</sub>	MT <sub>2</sub> → U <sub>1</sub>
x) Server broadcasts d <sub>3</sub>	MT <sub>1</sub>	MT <sub>2</sub> → U <sub>1</sub>
xi) MT <sub>1</sub> read d <sub>3</sub>	U <sub>1</sub> → MT <sub>1</sub>	MT <sub>2</sub> → U <sub>1</sub>
xii) U <sub>2</sub> updates d <sub>1</sub>	U <sub>1</sub> → MT <sub>1</sub>	MT <sub>2</sub> → U <sub>1</sub>
xiii) Server broadcasts message: d <sub>1</sub> is updated	U <sub>1</sub> → MT <sub>1</sub> → U <sub>2</sub>	MT <sub>2</sub> → U <sub>1</sub>
xiv) MT <sub>1</sub> and MT <sub>2</sub> reads message: d <sub>1</sub> is updated	U <sub>1</sub> → MT <sub>1</sub> → U <sub>2</sub>	MT <sub>2</sub> → U <sub>1</sub>
xv) Server broadcasts d <sub>4</sub>	U <sub>1</sub> → MT <sub>1</sub> → U <sub>2</sub>	MT <sub>2</sub> → U <sub>1</sub>
xvi) MT <sub>1</sub> read d <sub>4</sub>	U <sub>1</sub> → MT <sub>1</sub> → U <sub>2</sub>	MT <sub>2</sub> → U <sub>1</sub>
xvii) MT <sub>1</sub> commits	U <sub>1</sub> → MT <sub>1</sub> → U <sub>2</sub>	MT <sub>2</sub> → U <sub>1</sub>
xviii) Server broadcasts d <sub>1</sub>	-	MT <sub>2</sub> → U <sub>1</sub>
xix) MT <sub>2</sub> read d <sub>1</sub>	-	U <sub>2</sub> → MT <sub>2</sub> → U <sub>1</sub>
xx) MT <sub>2</sub> commit	-	U <sub>2</sub> → MT <sub>2</sub> → U <sub>1</sub>

The order of update transaction U<sub>1</sub> and U<sub>2</sub> observed by MT<sub>1</sub> and MT<sub>2</sub> are different. However both of them are correct. Please note that in our algorithm, the local serialization graphs kept by MT<sub>1</sub> and MT<sub>2</sub> will be MT<sub>1</sub> → U<sub>2</sub> and MT<sub>2</sub> → U<sub>1</sub> respectively since a mobile transaction will only keep update information on data items it has already read. This is because if an update transaction is before a mobile transaction, it will never cause a cyclic schedule.

#### 6.4 Disconnection

An important property of mobile network is frequent disconnection, which may be voluntary or involuntary. In voluntary disconnection, the mobile client initiates a disconnection in order to conserve power of the mobile machine. Voluntary disconnection of a mobile client will only occur after the completion of its transaction. Thus, it neither affects the mechanism of SCM nor creates any problem regarding to the correctness of the protocol. Involuntary disconnection results from instability in mobile network. For example, in a cellular radio network, the strength of signal received by a mobile client is affected by a number of factors such as the distance between the mobile clients and the base station, as well as the height of the surrounding buildings. Disconnection may occur once the signal received by the mobile client is lower than a threshold level. Although the disconnection is usually temporary, its impact on the consistency of transaction execution can be very serious.

The main effect of disconnection on SCM is that at the time of broadcasting conflicting update transaction identities, a mobile client may be disconnected from the network. For this case, the mobile client cannot get the new data conflict information. One simple method to solve the problem is to broadcast additional information in a special header at the beginning of a broadcast cycle. This information consists of the identities and the time-stamps of all the data items which have been involved in any conflict between the update and broadcast transactions within the length of the broadcast transaction. When a mobile client reconnects, it has to wait until the start of a broadcast cycle. Then, it checks the header to see whether any of its accessed data items conflict with any update transaction while it is disconnected from the network. All such items will be marked invalid and the transaction must retrieve the current version of these data items again.

#### 6.5 *Comparison between UFO and SCM*

Since SCM is based on the execution framework of UFO, it also shares many desirable properties of the UFO. For example, it can be applied to different broadcast algorithm since it does not affect the broadcast algorithm; All the data items observed by the mobile transactions are the most updated versions. On the other hand, the overhead of SCM is much smaller than UFO in case of high data conflict probability. The UFO algorithm resolves data conflict by data re-broadcast. However, in SCM, only the identities of the transactions will be broadcast in case of data conflict. Of course, the additional overhead of the SCM is that the mobile transactions need to maintain their local serialization graph. Considering that the bandwidth is the often the bottleneck resource in a mobile computing system, the most important performance objective should aim to minimize the bandwidth overhead instead of other processing overheads at the server or client sides.

The cost for the lower broadcast overhead in SCM is that the mobile transactions may observe different serialization order since the checking is only performed on their local serialization graphs. For most cases, this is not a big problem since the transactions from different mobile clients are assumed to be independent.

### 7 **Conclusions**

Data broadcast has been shown to be an efficient method for disseminating data items to transactions generated by mobile clients. Although research in the design of broadcast algorithms has received a lot of attention in previous years, the concurrency control issue has been greatly ignored. If the broadcast of data items and the execution of update transactions, which are important to maintain the freshness of the data items at the database, are uncontrolled, the mobile transactions may observe inconsistent data values. In this paper, we study the concurrency control between update transactions and mobile transactions, which consist of read-only operations. Based on the framework proposed in the UFO algorithm, a serialization checking method has been proposed. Similar to the UFO algorithm, the protocol is simple and can effectively maintain the schedule update and mobile transactions to be serializable. Unlike the UFO algorithm, overhead for ensuring data consistency observed by the mobile transactions is much lower since

no data items need to be re-broadcast. This is because conflict is detected by simply searching the local serialization graph of the mobile transactions instead by data re-broadcast.

## References

- [AFZ95] Acharya, S., Alonso, R., Franklin, M., Zdonik, S., “Broadcast Disks: Data Management for Asymmetric Communications Environments” in *Proceedings of ACM SIGMOD*, 1995.
- [AFZ97] Acharya, S., Franklin, M., Zdonik, S., “Balancing Push and Pull for Data Broadcast”, in *Proceedings of ACM SIGMOD*, Tucson, Arizona, May 1997.
- [BHG87] Bernstein, P.A., Hadzilacos, V., Goodman, N., *Concurrency Control and Recovery in Database System*, Addison-Wesley Publishing Company, 1987.
- [DCKV97] Datta, A., Celik, A., Kim, J. and VanderMeer, D.E., “Adaptive Broadcast Protocol to Support Power Conservant Retrieval by Mobile Users”, in *Proceedings of 13th International Conference on Data Engineering*, 1997.
- [H94] Haug, T., “Overview of GSM: Philosophy and Results”, *International Journal of Wireless Information Networks*, vol. 1, no. 1, pp. 7-16, 1994.
- [IB94] Imielinski, T. and Badrinath, B.R., “Mobile Wireless Computing: Challenges in Data Communications of the ACM”, vol. 37, no. 10, Oct. 1994.
- [LCA99] Lam, K.Y., Chan, Edward and Au, Mei-Wai, “Broadcast of Consistent Data to Read-Only Transactions from Mobile Clients”, in *Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, Louisiana, USA, 1999.
- [LS97] Leong, H.V. and Si, A., “Database Caching over the Air-Storage”, *The Computer Journal*, vol. 40, no. 7, pp. 401-415, 1997.
- [PB95] Pitoura, E. and Bhargava, B. “Maintaining Consistency of Data in Mobile Distributed Computing Systems”, pp. 404-413, 1995.
- [P98] Pitoura, E. “Supporting Read-Only Transactions in Wireless Broadcasting”, in *Proceedings of the DEXA '98 Workshop on Mobility in Databases and Distributed Systems*, August 1998.
- [PC99] Pitoura, E. and Chrysanthis, P.K., “Scalable Processing of Read-Only Transactions in Broadcast Push”, in *Proceedings of the 19th IEEE International Conference on Distributed Computing System*, pp. 432-441, 1999.
- [SNSR98] Shnmugasundram, J., Nithrakashyap, A., Sivasankaran, R., Ramamritham, K., “Efficient Concurrency Control for Broadcast Environments,” *Technical Report, 1997-062*, Department of Computer Science, University of Massachusetts, Amherst, 1998.
- [XSGFR97] Xuan, P., O. Gonzalez, J. Fernandez and Ramamritham, K., “Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments”, in *Proceedings of 3rd IEEE Real-Time Technology Application Symposium*, 1997.