

# Concurrency Control for Mobile Systems 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

## Abstract

Although data broadcast has been shown to be an efficient method for disseminating data items in a mobile computing system with large number of clients, the issue on how to ensure the data consistency observed by mobile transactions, which are generated by mobile clients, 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 updates and broadcast of data items are interleaved without any control, the mobile transactions 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. Simulation experiments have been performed to investigate the performance characteristics of the proposed method.

## 1 Introduction

Recent advances in mobile communication technologies 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].

---

<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 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 through a mobile network. 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.

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. Allowing concurrent execution of data broadcast and update transactions not only can improve the response time of mobile transactions, it can also help to maintain the freshness of the data items, since most of data items represent real-time information. We propose the *Serialization Checking Method (SCM)* to ensure the consistency of the data items observed by mobile transactions. The SCM is based on the conflict checking framework proposed in the *Update-First with Order (UFO)* algorithm, which has several important advantages for mobile computing systems [LAC99]. The additional important advantage of SCM as compared with UFO is that the broadcast overhead of SCM is much smaller especially when the conflict probability between update and the broadcast of data items is high and the sizes of the data items are large.

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 system model for a mobile computing system with data broadcast. Section 4 discusses the problem of data inconsistency when update and data broadcast are performed concurrently. Section 5 reviews how the UFO algorithm resolves the concurrency control problem. Section 6 introduces the SCM and explains how it improves the system performance with examples. Section 7 reports the simulation results of the proposed algorithm as compared with UFO which has been shown to give a better performance than other methods such as the multiversion broadcast method [PC99]. The paper concludes in Section 8.

## **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 overheads for detecting data conflicts in a mobile environment [P95, P98]. Owing to the relatively poor quality of services of a mobile network, it is usually not easy

to ensure data consistency and to detect data conflicts in a mobile network. Two important properties of mobile networks are frequent disconnection and low bandwidth especially the up-link channels. One suggested solution to deal with the poor communication network is to relax the consistency requirement in processing a mobile transaction. In [PB95], a two-level consistency model is proposed. Semantically related data are grouped together into a cluster, and the data items inside a cluster are mutually consistent. Certain degrees of inconsistency are allowed among the data items at different clusters.

In [SNSR98], a control matrix scheme is proposed for data conflict resolution. For a database with  $n$  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 as 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], a multi-version data broadcast method is suggested to resolve the problem of reading inconsistent data values for read-only transactions from mobile clients. In the proposed method, in addition to the most updated version, all the previous versions within a time frame need to be broadcast as well. The main problem of the multiversion method is the additional overhead in broadcasting multiple versions of a data item. Moreover the mobile transactions may observe stale data values since they are allowed to read an “old” version of a data item.

Another method to detect the non-serializability problem is to broadcast serialization graphs [P98, PC99] such that 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 the heavy overhead in broadcasting the serialization graphs since every data conflict at the database server has to be broadcast. Secondly, each client must continuously listen to the transmission channel to maintain and update its serialization graph. This leads to another serious problem, which can seriously affect the correctness of this approach: the needs to maintain the local serialization graph at a 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 broadcast server, 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 broadcast server through the low bandwidth wireless channels of the mobile network.

The database server maintains a database which contains 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 and “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 the last traded 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 write 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.

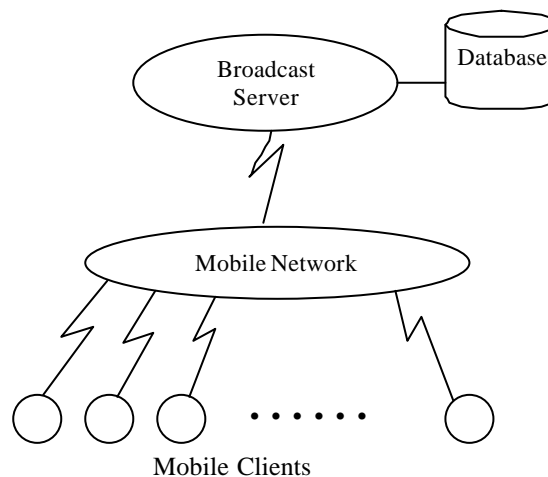


Figure 1. System Model

The database server continuously broadcasts data items from the database one by one to the mobile clients through a mobile network until the end of a broadcast cycle. Each broadcast cycle

follows the preceding cycle immediately. It is assumed that an efficient broadcast algorithm is adopted for selecting data items to broadcast<sup>2</sup>. The length of a broadcast cycle may be fixed or variable. The data items may require different times for broadcast due to different sizes.

In the model, the broadcast process is modeled as a long read-only transaction called the *broadcast transaction (BT)*. The length of the broadcast transaction is defined as the length of a broadcast cycle. The set of data items in a broadcast transaction consists of those data items, which are broadcast in a cycle. The execution of BT and update transactions is interleaved to reduce the blocking delay of 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 read-only transactions, called *mobile transactions (MT)*, to access data items at the database server. It is assumed that 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 have to be completed within 5 seconds after their generations. Transactions, which have missed their deadlines, might still be of some use, but beyond a certain time point they would be considered totally useless. The period between the generation time of a transaction and the time when it is considered to be useless is called its *drop period*. It is obvious that the timing constraints on completion will be tighter if the drop period is smaller.

It is assumed that each MT consists of a set of data requests (read operations). The requests can be performed in any order, i.e. they are unordered. The arrival sequence of the requested data items of the requests in a transaction is unimportant as long as all the requested data items are received before the transaction becomes useless. An example for this type of transactions is the transaction for the weather forecasts of a few cities or the transaction for several stock data. A transaction is issued instead of several separate transactions simply because batching the transactions into a single transaction can reduce the system overhead. When all the requests of a transaction are satisfied, the transaction is completed and the results will be reported to the originating client after the commitment of the transaction.

The assumptions of the system model are summarized below:

---

<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 algorithm does not depend on the broadcast algorithm selected and can be applied to any of these broadcast algorithms.

1. All update transactions are at the database server.
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 and are processed at mobile clients.
4. Mobile transactions can be *unordered*.
5. The results of a mobile transaction will report to its client upon its commitment.
6. 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 update transaction, U, updates data item  $d_5$  and then data item  $d_2$ , and 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 thus 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 acyclic, 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  $U_1$  updates  $d_2$  and then  $d_1$ , and  $U_2$  updates  $d_1$  and then  $d_5$ . If the schedule is:

- i) Broadcast transaction (BT) broadcasts  $d_2$

---

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

- 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 serializability of the serialization graphs, 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  $BT \rightarrow MT$  if they have any conflicts.)

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 orders of the transactions follow their time-stamps, i.e., after a transaction has started, the next transaction can start its execution only after the completion of the last transaction. In this way, the final schedule will be serial. However, this method has 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 a 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 the design of concurrency control protocols for mobile computing systems 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. This makes data conflict detection very difficult. In a mobile environment, the bandwidths of the mobile channels, especially the up-link channels, are very limited. Therefore, it is important to system performance to minimize the number of communications between the mobile clients and the broadcast server especially using the up-link channels. Instead of detecting data conflicts between mobile transactions and update transactions, the UFO algorithm checks data conflicts between broadcast and update transactions [LCA99].

In the UFO, the length of a broadcast transaction is re-defined to be the length of the set of data items which are broadcast in the period from (current time – the life span of a mobile transaction) to current time such that no mobile transaction reads data items from more than one broadcast transaction. 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 between a BT and a MT is always  $BT \rightarrow MT$  if they have any conflict. Thus, serialization order between the update transactions and the mobile transactions will always be  $U \rightarrow MT$  if they have any conflict. Therefore, the final serialization graph can be ensured to be serializable [BHG87].

Basically, the UFO algorithm 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 any data conflicts with other update transactions are resolved using a conventional concurrency

control protocol, such as 2PL [BGH87]. 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. Data conflict with the 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 of dividing the execution of an update transaction 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 following 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 an update transaction and a broadcast transaction will occur only when the update transaction is in the update phase, which is much shorter. Secondly, the detection of data conflicts between an update transaction and a broadcast transaction will become much simpler. At the update phase, the system knows 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 and resolved when the update transaction enters its update phase. Data 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. Note that the set of data items in a broadcast transaction is not fixed since 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. After the broadcast of a data item, the last broadcast data item will be included in the broadcast transaction 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 and the sizes of the data items are large, the re-broadcast overhead could be quite significant leading to significant degradation in overall system performance. In the 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 solely on data re-broadcast.

As can be seen in Section 4, the main data inconsistency problem in a broadcast environment is that a data item, which is captured from the air, may later be updated by an update transaction (example 1 in Section 4). If the update transaction has further data conflict with the broadcast transaction, it may cause a cyclic schedule with a mobile transaction. Another reason for a cyclic schedule is due to transitive dependencies among update transactions (example 2 in Section 4). To deal with these two problem cases, mobile clients may be provided with data conflict information of the update transactions to construct their local serialization graphs. In case a cyclic schedule is formed, the mobile client will dispose the data item that is read before the update transaction, which causes the non-serializable schedule. To solve the first case, the SCM 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 which is the length of the broadcast transaction. To handle the case of transitive dependencies, the SCM simply broadcasts the information of the update transactions that have transitive dependencies with the data items which are broadcast in the last drop period.

### **6.1 Algorithm for the Server**

Similar to the UFO algorithm, the execution of an update transaction in the SCM is also divided into two phases and any conflict between an update transaction and the broadcast transaction will be detected when the update transaction enters its update phase. In case of a data conflict, the dependency between the update transaction and the broadcast transaction will be broadcast instead of re-broadcasting the conflicting items. Note that the identity of a transaction is much smaller than the

data itself. At the same time, the dependencies between the update transaction and other update transactions in the last drop period, which is the length of the current broadcast transaction, will also be broadcast.

After the completion of update transaction  $U$ , the following algorithm will be 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 and  $U$ 's ID

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 and they also listen to the messages on update transactions in order to build their local serialization graphs. Once a serialization graph is updated, a search 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 causes the non-serializable schedule. The disposed data items will be captured from the 'air' again when it is broadcast the next time. The process will continue until all the required data items are captured or the drop period of the transaction is expired.

```

read-set =  $\{\}$ 
loop until (read-set =  $D_{MT}$ ) or (drop period of MT has expired)

case: a data item  $d_i$  is broadcast
  if  $d_i \in D_{MT}$ 
    then read-set = read-set  $\cup \{d_i\}$ 

case: an update transaction  $U$  and its data item are 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 read-set  $\cap D_U \neq \{\}$ 
          add  $MT \rightarrow U$  in the serialization graph
        if  $\exists U_i \in T_S$  s.t.  $D_U \cap D_{U_i} \neq \{\}$ 

```

```

        add  $U_i \rightarrow U$  in the serialization graph
    search the updated serialization graph
    if a cyclic schedule is formed
        then disposes the data item getting before
            the update transaction in conflict
    endloop

```

where  $D_{MT}$  = the set of data items requested by MT

$T_S$  = the set of update transactions in the serialization graph

### 6.3 An Example

In the following, we use an example to illustrate the mechanism of the SCM algorithm.

#### Example 4:

This example shows a 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 by $U_1$	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 by $U_2$	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 the above example the server checks 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 for the update transaction. So the mobile client 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  $U_1$ .

## 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 energy 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 algorithm. 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 broadcast server, 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 to update its local serialization graph. One simple method to solve the problem is to broadcast additional information in a special header at the beginning of a broadcast cycle. The 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 the UFO, it also shares many desirable properties of the UFO. For example, it can be applied to different broadcast algorithms since it does not have any specific requirements on the adopted broadcast algorithm. Furthermore, all the data items observed by 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 a data conflict. Of course, the additional overhead of the SCM is that the mobile clients need to maintain their local serialization graphs. 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 a lower broadcast overhead in the SCM is that the mobile transactions at different clients may observe different serialization orders since each mobile only checks the serializability based on its local serialization graph. However, we would remark that for most cases, this is not a big problem since the transactions from different mobile clients are assumed to be independent.

**Example 5:**

This example illustrates that different mobile transactions may observe different serialization orders with an update transaction. It is assumed that the mobile transaction  $MT_1$  reads  $d_1$ ,  $d_3$  and  $d_4$  while  $MT_2$  reads  $d_1$  and  $d_2$ . The update transaction  $U_1$  updates  $d_2$  and  $d_3$ , and  $U_2$  updates  $d_1$ .

Steps	Serialization graph for $MT_1$	Serialization graph for $MT_2$
i) $MT_1$ begins execution	$MT_1$	-
ii) Server broadcasts $d_1$	$MT_1$	-
iii) $MT_1$ reads $d_1$	$MT_1$	-
iv) $MT_2$ begins execution	$MT_1$	$MT_2$
v) Server broadcasts $d_2$	$MT_1$	$MT_2$
vi) $MT_2$ reads $d_2$	$MT_1$	$MT_2$
vii) $U_1$ updates $d_2$ & $d_3$	$MT_1$	$MT_2$
viii) Server broadcasts message: $d_2$ and $d_3$ are updated	$MT_1$	$MT_2$
ix) $MT_1$ and $MT_2$ read message: $d_2$ and $d_3$ are updated	$MT_1$	$MT_2 \rightarrow U_1$
x) Server broadcasts $d_3$	$MT_1$	$MT_2 \rightarrow U_1$
xi) $MT_1$ read $d_3$	$U_1 \rightarrow MT_1$	$MT_2 \rightarrow U_1$
xii) $U_2$ updates $d_1$	$U_1 \rightarrow MT_1$	$MT_2 \rightarrow U_1$
xiii) Server broadcasts message: $d_1$ is updated	$U_1 \rightarrow MT_1 \rightarrow U_2$	$MT_2 \rightarrow U_1$
xiv) $MT_1$ and $MT_2$ read message: $d_1$ is updated	$U_1 \rightarrow MT_1 \rightarrow U_2$	$MT_2 \rightarrow U_1$
xv) Server broadcasts $d_4$	$U_1 \rightarrow MT_1 \rightarrow U_2$	$MT_2 \rightarrow U_1$
xvi) $MT_1$ reads $d_4$	$U_1 \rightarrow MT_1 \rightarrow U_2$	$MT_2 \rightarrow U_1$
xvii) $MT_1$ commits	$U_1 \rightarrow MT_1 \rightarrow U_2$	$MT_2 \rightarrow U_1$

xviii)	Server broadcasts $d_1$	-	$MT_2 \rightarrow U_1$
xix)	$MT_2$ reads $d_1$	-	$U_2 \rightarrow MT_2 \rightarrow U_1$
xx)	$MT_2$ commits	-	$U_2 \rightarrow MT_2 \rightarrow U_1$

The serialization order of update transaction  $U_1$  and  $U_2$  observed by  $MT_1$  and  $MT_2$  are different. However, both of them are serializable. Please note that in our algorithm, the local serialization graphs maintained by  $MT_1$  and  $MT_2$  will be  $MT_1 \rightarrow U_2$  and  $MT_2 \rightarrow U_1$  respectively since a mobile transaction will only maintain update information on data items that it has already read. This is because if an update transaction is before a mobile transaction, it will never cause a cyclic schedule.

## 7 Performance Studies

In this section, we study the performance of the SCM algorithm using extensive simulation experiments. We compare its performance with UFO which has been shown to give a better performance than other methods such as the multiversion broadcast method [LCL00]. A mobile computing system using data broadcast has been implemented in CSIM which is a simulation language implemented in the C programming language. The experiments cover the different access patterns of the mobile and update transactions.

### 7.1 Simulation Model

The simulation model, shown in Figure 2, is developed based on the model introduced in Section 3. Basically, the model consists of three entities and four processes. The three entities are the database server, the air media and the mobile clients. Three of the four processes are at the database server: the broadcast process, the server update process and the update database process. It is assumed that the database server is a multi-processor system and the processes can be executed concurrently provided that they are not accessing the same data item at the same time. For the latter case, blocking will be used to solve the data synchronization problem. The last process is the client process at each mobile client for generating mobile transactions.

#### Database Server

The database server maintains a database. To simplify the model parameters, it is assumed that the data items have similar size and each data item has a unique identifier.

## Air Media

The air media is a collection of air channels. To simplify the case, we assume that there is only one broadcast channel for both scheduled broadcast data items and for data re-broadcast in the SCM algorithm.

## Mobile Clients

There are a large number of mobile clients in the system. They generate read-only transactions, the mobile transactions. Each time a mobile client will generate a mobile transaction. A mobile client generates the next mobile transaction after a think time upon the completion or abort of a mobile transaction. The think time is exponentially distributed.

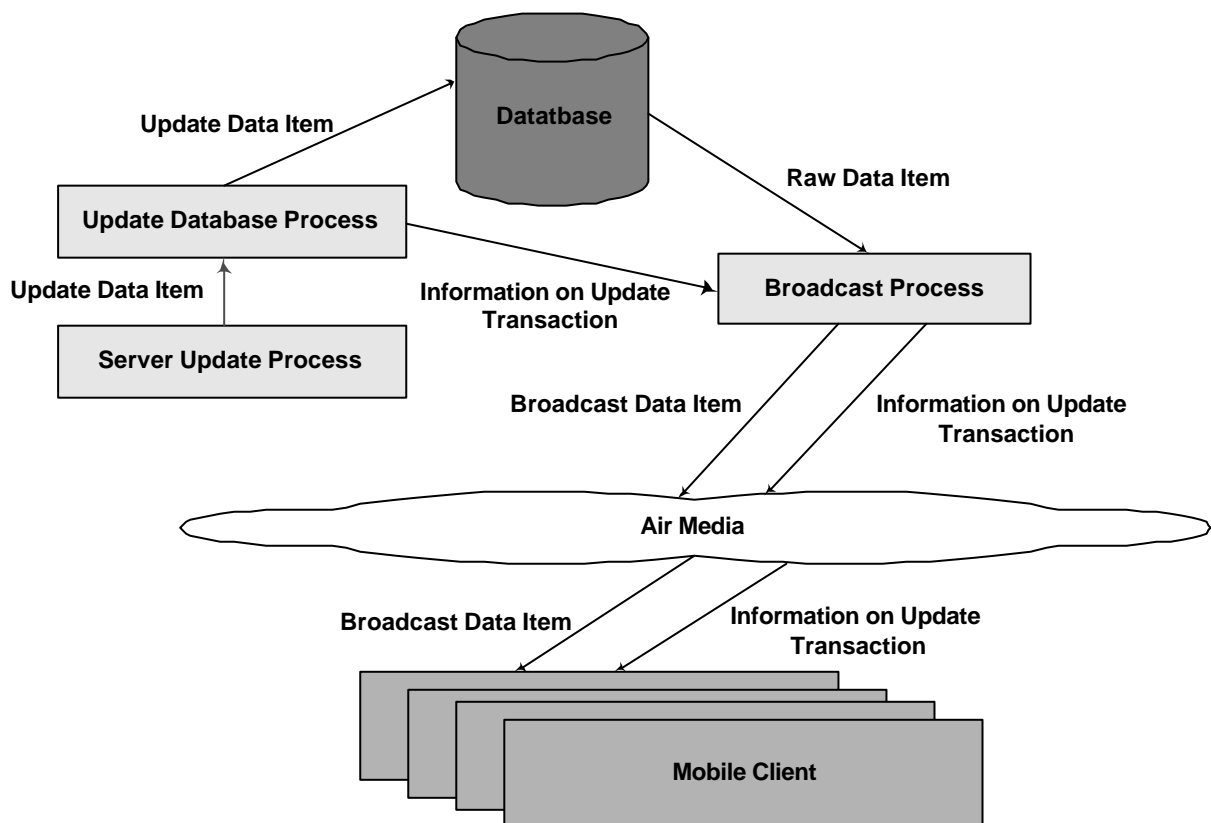


Figure 2: Simulation model for the SCM algorithm

## Broadcast Process

The broadcast process selects data items from the database for broadcast. To simplify the model, we assume a flat broadcast disk model. The broadcast cycle is static and includes all the data items in the database. Thus, the length of a broadcast cycle equals to the size of the database.

The broadcast process receives update transactions from the update database process. For each time slot in a broadcast cycle, the broadcast process first checks whether there is any update transaction information waiting for broadcast. If there is, the data items in the transaction and their identities will be broadcast first. Then, it broadcasts the next data item in the pre-defined broadcast schedule based on the flat broadcast disk model.

### **Server Update Process**

The server update process generates update transactions. We assume that all update transactions are processed at the database server. Each update transaction may read and write data items. The data items to be updated by an update transaction are assumed to be spread over the database uniformly. (In the experiments, we also test the system when the required data items of the update transactions following the Zipf distribution.) Each update transaction is assigned a unique time-stamp using the current time of its generation. The server update process uses a deferred update mechanism to update the database such that the actual update on the database is performed at the end of an update transaction in batch. Since the focus of this paper is on how the SCM algorithm can solve the inconsistency problem between mobile and update transactions and study the cost of the SCM algorithm, we are not going to simulate the process for concurrency control among the update transactions.

### **Update Database Process**

The update database process performs permanent update on the database and sends the update information to the broadcast process if the update transaction has any dependency with broadcast in the last drop period. The dependency includes any conflict with data items or update transactions.

### **Mobile Client Process**

The mobile client process generates read-only mobile transactions. Each mobile transaction consists of a set of data requests. The data requests are unordered i.e. they can be processed in any order. The data access pattern of the requests may be uniformed or skewed with the Zipf distribution. Mobile clients listen to the broadcast channel and capture the data items they required. They also listen to the broadcast channel for any update transaction information. They maintain their local serialization graphs. If a cycle is formed in a local serialization graph, then the mobile client will dispose the data items that are captured before the update transaction in conflict to resolve the cycle. Deadline of a mobile transaction is set to be the drop period plus their generation time. If its deadline is missed, it is abort.

## 7.2 Model Parameters and Performance Measures

### 7.2.1 Model Parameters

Table 1 lists the model parameters and their baseline values.

Parameter	Baseline Value
Database size	1000 data items
Number of client	100
Bandwidth of broadcast channel	128 KB / second
Size of data item	5 KB
Size of data item ID	10 bits
Number of data item in a mobile client transaction	1 to 4
Number of data item in an update transaction	1 to 2
Drop period	30 seconds
Client think time	10 seconds
Update interval	0.1 to 20 seconds
Skew coefficient (for skewed access only)	1.0

Table 1: Model parameters and baseline values

### 7.2.2 Performance Measures

The performance measures for comparing the two algorithms, UFO and SCM, are miss rate, mean response time and channel utilization for the algorithm. The miss rate is defined as the number of mobile transactions, which miss their deadlines, divided by the total number of mobile transactions generated. It measures the capability in meeting the timing requirements of the transactions. The mean response time is measured for all mobile transactions including those aborted. Channel utilization measures the percentage of the broadcast channel used for the algorithm to maintain data consistency provided to mobile transactions. It measures the overhead of the algorithm in terms of its consumption of scarce downlink bandwidth.

## 7.3 Performance Results and Discussion

We have performed four sets of simulation experiments. Each simulation run consists of 400,000 mobile transactions. The simulation length is determined from a number of trial runs until the output results are stable.

### 7.3.1 Uniform Data Access

In this set of experiments, the data access patterns of both the mobile transactions and the update transactions at the database server are uniform. Figures 3 to 5 are results on miss rate, mean response time and channel utilization. Figure 3 shows that the miss rate of SCM is much lower than UFO especially at heavy update workload e.g. when update interval is less than 2 seconds. This is consistent with our expectation since SCM consumes less bandwidth than UFO for re-broadcast.

Therefore there is less data blocking in broadcast in SCM comparing with UFO. The bandwidth consumption is confirmed in Figure 5, which shows that the bandwidth used for UFO increases sharply when the update interval is less than 2 seconds. On the other hand, overhead in SCM is very steady across the whole range of update workload. Consistent with the miss rate in Figure 3, the mean response time of mobile transaction in SCM is more or less the same under different update workloads as shown in Figure 4, while the mean response time of the transactions in UFO decreases with a reduction in update transaction workload.

This set of experiments shows that SCM performs better than UFO especially when the update workload on the database server is heavy. It also illustrates that SCM is more adaptable to different update workloads as its processing overhead is much lower than that of UFO.

### **7.3.2 Skewed Data Accesses of MT**

In this set of experiments, the data access pattern of the mobile transactions is skewed with the skew coefficient set to 1. The data access pattern of the update transactions is uniform. Figures 6 to 8 are the results on miss rate, mean response time and channel utilization, respectively. As shown in Figure 6, the miss rate of SCM is much less than that of UFO under all update workloads. The reason is that SCM consumes much less bandwidth than UFO. This is confirmed with the channel utilization graph shown in Figure 8. The difference is much more significant when the update workload is higher, e.g. when the update interval is less than 2 seconds, since the overhead of UFO is more significant for a higher update workload. In addition the mobile transactions did not always benefit from the data re-broadcast mechanism in UFO since there is a data mismatch between the mobile transactions and update transactions. Consistent with the miss rate in Figure 6, the mean response time in Figure 7 also shows that the mean response time for UFO is much higher than that of SCM especially when the update workload is greater than 2 seconds.

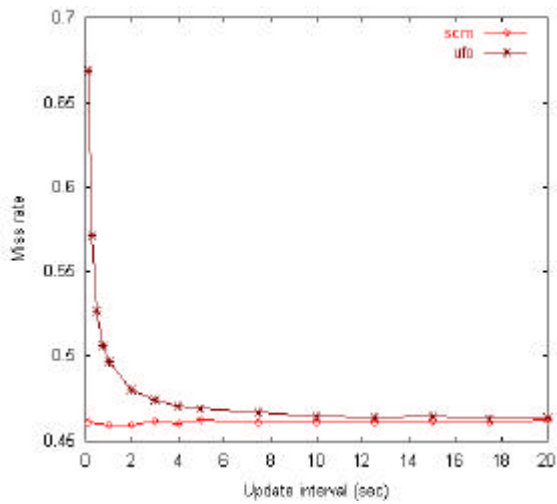


Figure 3: Miss rate for uniform data access

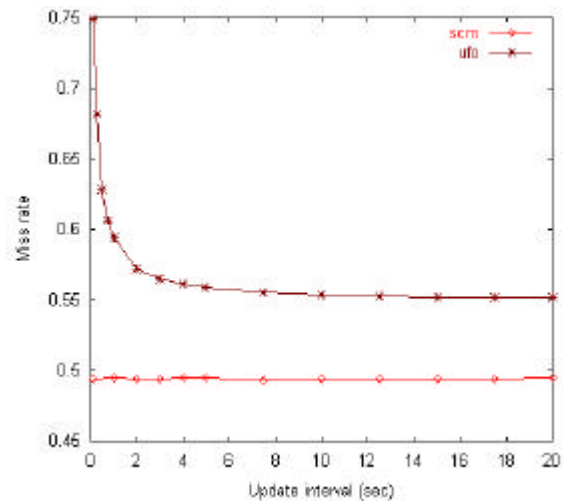


Figure 6: Miss rate for skewed data access of mobile client

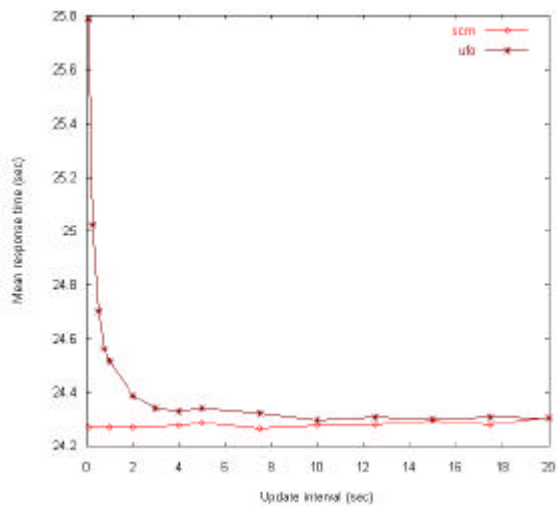


Figure 4: Mean response time for uniform data access

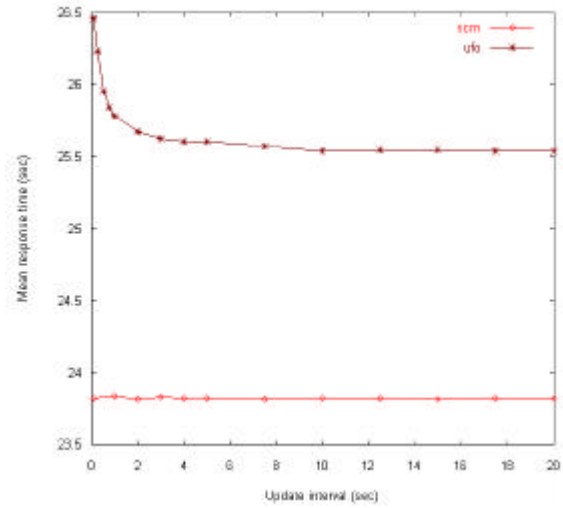


Figure 7: Mean response time for skewed data access of mobile client

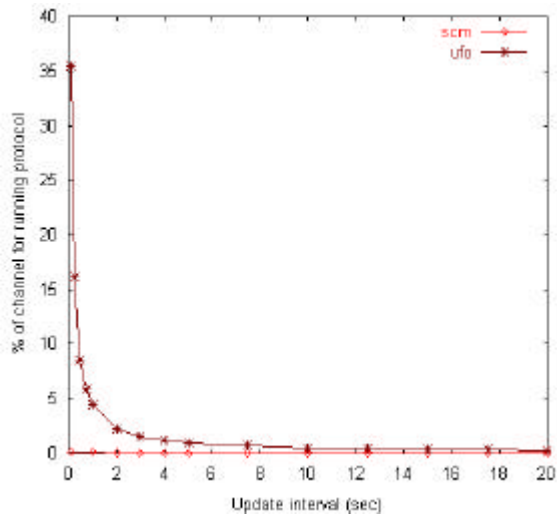


Figure 5: Channel utilization for uniform data access

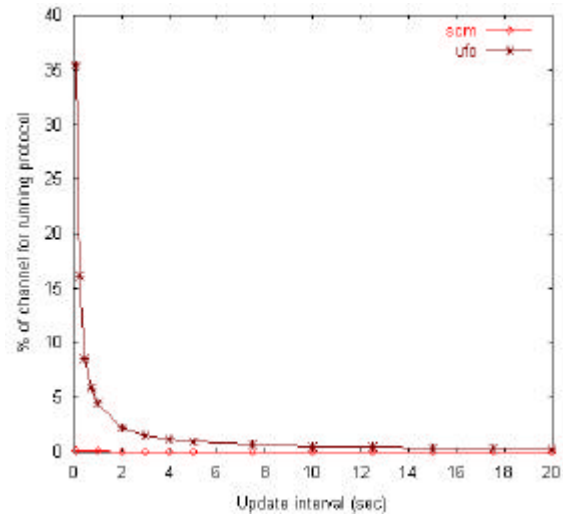


Figure 8: Channel utilization for skewed data access of mobile client

This set of experiments shows that when there is a mismatch between the access pattern of the mobile transactions and the update transactions, SCM performs much better than UFO for any update workload and not just for high update workload only. Comparing the miss rate in Figure 6 to the miss rate for uniform data access in Figure 3, it can be seen that there is a small difference in miss rate between SCM and the UFO when the update interval is greater than 2 seconds in Figure 3. But for Figure 6, there is about 5% difference in miss rate between SCM and UFO when the update interval is greater than 2 seconds. Similar observations can be seen in the mean response time in Figure 7 and those for uniform data access in Figure 4. In Figure 4, the difference in mean response time for SCM and UFO is not significant when the update interval is greater than 2 seconds. But in Figure 7, the difference between the two algorithms is clearly significant when the update interval is greater than 2 seconds.

### **7.3.3 Skewed Data Access of Both MT and U (No Offset)**

In this set of experiments, the data access patterns of the mobile and update transactions are both skewed with coefficient set to 1 and no offset between their data access patterns, i.e. the mobile and the update transactions have the same set of hot data items. Figures 9 to 11 summarize the results on miss rate, mean response time and channel utilization, respectively. As shown in Figure 9, the miss rate of UFO is less than that of SCM when the update interval is less than 5 seconds. This is because data items being re-broadcast in the UFO algorithm are also the hot data items required by mobile transactions. The re-broadcast data items can satisfy new data requests from the mobile transactions besides refreshing the old data items. The SCM does not have this advantage since it only broadcasts the identities of updated data items and transactions. Therefore the UFO performs better than the SCM at high update rate. However for very high update rate such as when the update interval is less than 1 second, the miss rate of UFO rises again as shown in Figure 9. This is because of data blocking by UFO since there are a lot of data waiting for re-broadcasts. This is shown in Figure 11, where it is clear that SCM consumes less than 1% of the channel for re-broadcast no matter how heavy is the update workload. On the other hand, the overhead of the UFO depends on the update workload. The UFO uses less than 2.5% of the channel when the update workload is less than 2 seconds, but rises quickly when the workload is greater than 2 seconds.

The mean response times shown in Figure 10 are consistent with results on the miss rates. The mean response time for UFO is less than that of SCM when the update interval is less than 3 seconds. But it is greater than that of SCM when the update interval is greater than 3 seconds. On the other hand, the mean response time for SCM is not influenced by the update workload and remains steady across the whole range of update workload.

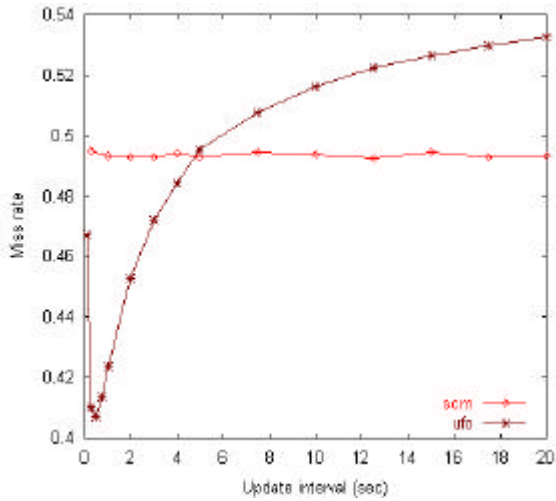


Figure 9: Miss rate for skewed data access of mobile client and update transaction without offset

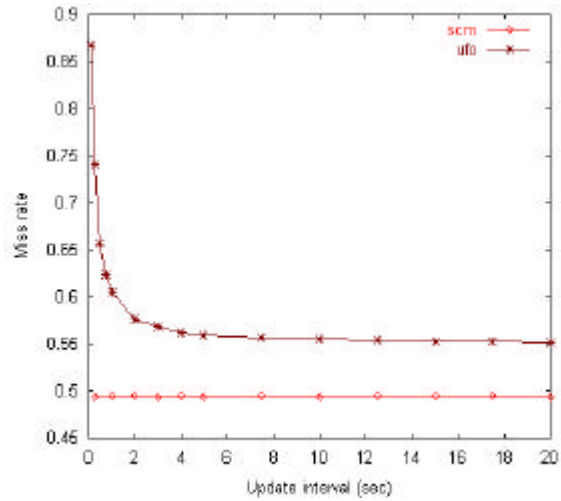


Figure 12: Miss rate for skewed data access of mobile client and update transaction with offset = 10%

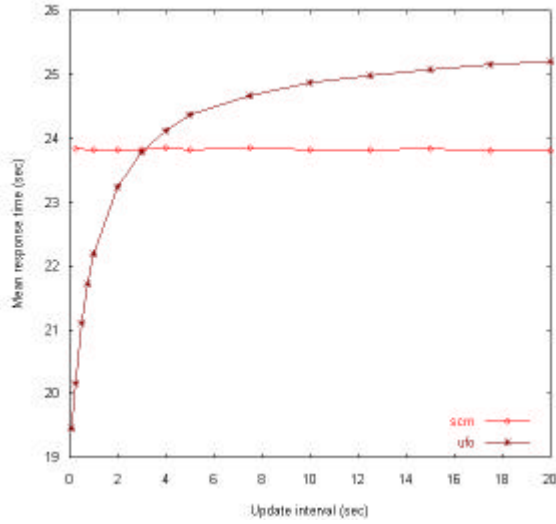


Figure 10: Mean response time for skewed data access of mobile client and update transaction without offset

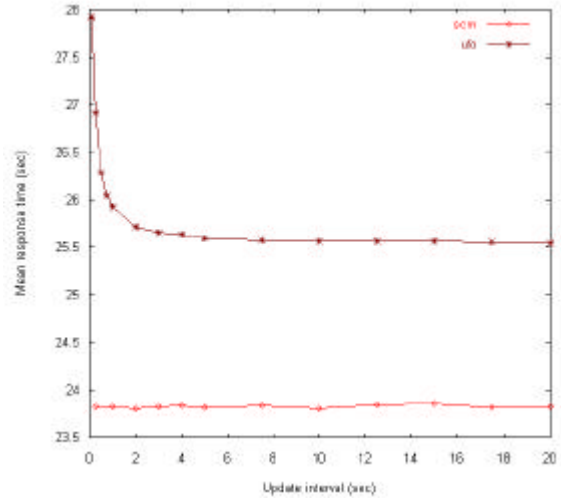


Figure 13: Mean response time for skewed data access of mobile client and update transaction with offset = 10%

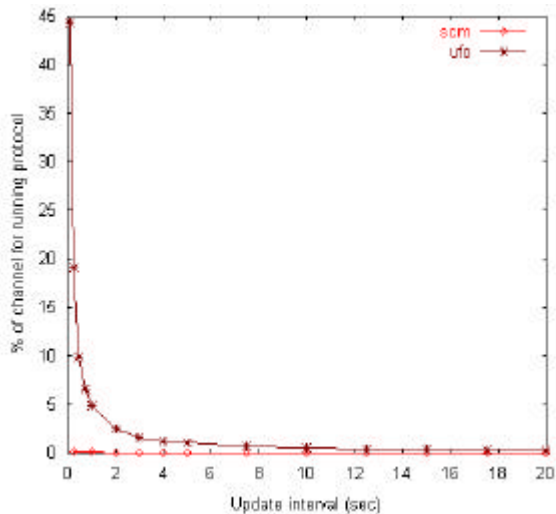


Figure 11: Channel utilization for skewed data access of mobile client and update transaction without offset

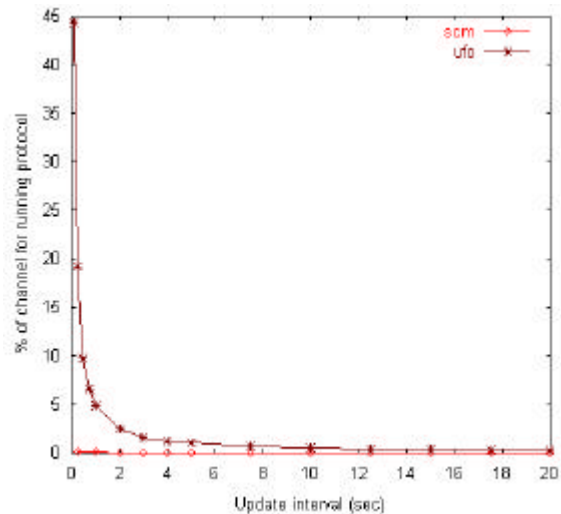


Figure 14: Channel utilization for skewed data access of mobile client and update transaction with offset = 10%

To briefly summarize the results of the experiments, it is clear the performance of SCM as measured by the miss rate and the mean response time is not influenced by update workload when the mobile transactions and update transactions have the same set of hot data items. Moreover, its performance is *not* affected by a skewed data access pattern. When comparing the miss rate of SCM in Figure 9 with miss rate of UFO for uniform data access in Figure 3, there is a 3% difference only. When comparing the mean response time of SCM for skewed access in Figure 10 with those for uniform access in Figure 4, there is only a difference of 0.5 second. On the other hand, the impact of skewed data access on UFO is very significant. When compared with the miss rate and mean response time of UFO for uniform data access in Figure 3 and 4, the trend of those for skewed access in Figure 9 and 10 are in the opposite directions. Therefore although UFO has some advantage in skewed data access with no offset, special attention should be paid to the update rate since UFO is very sensitive to it.

#### **7.3.4 Skewed Data Access of Both MT and U (Offset = 10%)**

This set of experiments is the same as the previous one except there is a 10% offset between the skewed data access of the mobile and update transactions, i.e. they have different sets of hot data items. This case is very similar to the set of experiments in section 7.3.2 in the sense that there is a mismatch in data access between the mobile and update transactions. Therefore the results in Figures 12 to 14 are very similar to those in Figures 6 to 8 and a similar explanation can be applied. As shown in Figures 12 and 13, the miss rate and the mean response time of SCM are less than that of the UFO for any update workload. When the update interval is greater than 2 seconds, the differences in miss rate and mean response time between SCM and UFO are about 5% and 1.7 seconds respectively. The difference is much more significant when the update interval is less than 2 seconds. Since the percentage of the channel overhead of UFO grows rapidly when the update workload is greater than 2 seconds as shown in Figure 14, while SCM consumes less than 1% of channel at any update workload.

This set of experiments shows that when the data access patterns of the mobile transactions and the update transactions are both skewed with an offset, SCM performs better than UFO for any update workload. The result is consistent with those obtained in the second set of experiments in section 7.3.2, where there is also a difference in data access pattern between the mobile transactions and update transactions. These once again confirm that SCM is adaptable to any type of data access pattern and update workload.

## 8 Conclusions

Data broadcast has been shown to be an efficient method for disseminating data items to transactions generated by mobile clients. Although the 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, mobile transactions may observe inconsistent data values. In this paper, we focus on concurrency control between update transactions and mobile transactions, which consist of unordered read-only operations. Based on the framework proposed previously for the UFO algorithm, the serialization checking method (SCM) algorithm has been proposed. The SCM algorithm retains the strength of the UFO algorithm in that it is simple and can effectively maintain the schedule between update and mobile transactions to be serializable. Unlike the UFO algorithm, the overhead for ensuring data consistency observed by the mobile transactions is much lower since the number of data items needed to be re-broadcast is greatly reduced. This is because data conflicts are detected by simply searching the local serialization graphs of the mobile clients instead by data re-broadcast. In addition, performance of the UFO algorithm will be slightly degraded when the update and the mobile transactions have different hot data items. Performance of the SCM algorithm, on the other hand, is very steady for different types of data access pattern of the update and mobile transactions.

The SCM algorithm proposed in this paper is for mobile clients with read-only transactions. They can be extended for read-write mobile transactions. Since up-link bandwidth is limited in a mobile environment, one possible extension is to batch the updates at the end of a mobile transaction. The batch update request is then sent to the database server for commitment, and the database server can notify results of the transaction through data broadcast. This can be easily incorporated in the SCM algorithm since it is designed for broadcast of update transaction information.

## References

- [AFZ95] Acharya, S., Alonso, R., Franklin, M. and Zdonik, S., “Broadcast Disks: Data Management for Asymmetric Communications Environments” in *Proceedings of ACM SIGMOD*, 1995.
- [AFZ97] Acharya, S., Franklin, M. and Zdonik, S., “Balancing Push and Pull for Data Broadcast”, in *Proceedings of ACM SIGMOD*, Tucson, Arizona, May 1997.
- [BHG87] Bernstein, P.A., Hadzilacos, V. and 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 Management," *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.
- [LCL00] Kam-Yiu Lam, Edward Chan, Hei-Wing Leung and Mei-Wai Au, "Broadcasting Consistent Data to Mobile Clients with Local Cache", in *Proceedings of 2000 International Conference on Management of Data (COMAD2000)*, India, December 2000.
- [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 Environment," in *Proceedings of the 15<sup>th</sup> International Conference on 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. and 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, Fernandez, J. and Ramamritham, K., "Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments", in *Proceedings of 3<sup>rd</sup> IEEE Real-Time Technology Application Symposium*, 1997.