

# Broadcasting Consistent Data to Mobile Clients with Local Cache

Kam-Yiu Lam, Edward Chan, Hei-Wing Leung and Mei-Wai Au

Department of Computer Science  
City University of Hong Kong  
83 Tat Chee Avenue, Kowloon  
HONG KONG

Email: {cskylam|csedchan}@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 currency and consistency of the data items has not been examined adequately. While data items are being broadcast, update transactions may install new values for them. If the executions of update transactions and the 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 design a broadcast method called *Ordered Update First with Order (OUFO)* for mobile computing systems where a mobile transaction consists of a sequence of read operations. Under OUFO, mobile transactions will receive consistent data values and the newest version of the data items. Besides ensuring consistency and maximizing currency of cached data, OUFO also aims at reducing the access delay of the mobile transactions. We have compared the performance of OUFO with two other efficient schemes, the multi-version broadcast with multi-version cache method and the invalidation method. The results show that OUFO offers better performance in most aspects.

## 1. INTRODUCTION

Owing to the intrinsic constraints of mobile computing systems, such as asymmetric bandwidth, limited power supply and unreliable communication, the design of an efficient and cost effective mobile computing system poses many challenges [2, 6, 9, 15]. One of the most important issues is how to efficiently disseminate consistent data items from a broadcast server to transactions, called *mobile transactions (MT)*, which are generated from mobile clients [6, 9]. In recent years, many efficient data dissemination methods were proposed [2, 8, 13, 14, 23]. Most of them are based on data broadcast in which the broadcast server periodically and continuously broadcasts data items to mobile clients. With 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.

Since many data items in a mobile computing system are used to record the real-time information in the system, e.g., the current traffic conditions of the roads, the last traded prices of stocks and news updates, their values can be highly dynamic. The updates capture the most current information for the system and refresh the values of the data items in the database [21, 22]. Allowing the execution of updates to be interleaved with data broadcast is thus important in maintaining the “freshness” (validity) of the data items. Accessing out-dated (stale) data items is undesirable and will significantly affect the usefulness of the information to mobile clients [11]. However, if concurrent execution of updates and data broadcast is allowed, the problem of concurrency control must be addressed. Otherwise, the resulting execution schedule between the mobile and update transactions may be non-serializable [3] and consistency of data items provided to mobile clients cannot be guaranteed.

The result of providing inconsistent information to a mobile client can be very serious. Consider the case that a mobile client may ask the exchange rates of US dollars to UK pounds in New York and London at the same time. If the returned results are inconsistent, i.e., they are representing the information at different times, the mobile client may make a wrong trading decision. Similarly, if the information of the last traded price of a stock is not consistent with the stock index, i.e., there is a rise in the last traded price of the stock while the stock index drops, the mobile client will be frustrated and does not know which one is correct.

Unfortunately, conventional concurrency control protocols, such as two phase locking, optimistic method and timestamp ordering [3], are not suitable for mobile computing systems as the overhead for setting locks and detecting data conflicts in a mobile environment can be very heavy [20].

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 broadcast server. In recent years, concurrency control for data broadcast has received growing interests [12, 15, 16, 20]. An efficient and pioneering method is by broadcasting multiple versions of data items [18]. Consistent data items are provided to mobile transactions by requiring the mobile transactions to read data items committed at the same point of time, i.e. the last broadcast cycle. The basic multi-version broadcast method is extended in [16, 17] for systems with client caches where multiple versions of data items are maintained. By reading cached data items, which are

committed at the same time, the data access delay can be much reduced and at the same time data consistency is ensured. Another efficient method for concurrency control between read-only mobile transactions and update transactions is a data re-broadcast scheme called *Update First with Ordering (UFO)* [12]. Although UFO can provide the most updated values of data items to mobile transactions and at the same time maintain the serializability of the execution between update and mobile transactions, it is designed for mobile transactions in which operations are unordered. In this paper, we extend the UFO protocol to *Ordered UFO (OUFO)* for mobile transactions whose data requirements are ordered. The issue of accessing consistent cached items is also addressed. We have performed extensive simulation experiments to compare the performance of OUFO with the multi-version broadcast method [16, 17] and the invalidation method where invalidation reports are broadcast from the broadcast server to the mobile clients to validate the accessed data items of mobile transactions [16]. The following summaries the contributions of the paper:

- (1) the UFO protocol is extended for mobile transactions in which read operations are ordered;
- (2) enhancements based on an invalidation scheme are suggested for accessing consistent data at the client caches; and
- (3) extensive simulation experiments are performed and the results are shown that OUFO outperforms the multi-version broadcast method and the invalidation method.

The organization of the remaining parts of the paper is as follows. Section 2 reviews related work on broadcasting consistent data items. Section 3 describes the system model. In Section 4, we define the correctness criteria and illustrate the problem of data inconsistency in data broadcast using examples. Section 5 introduces the OUFO protocol with a detailed discussion on its correctness, properties and implementation overhead. Section 6 compares the performance of OUFO with the multi-version broadcast with multi-version cache method and the invalidation method using simulation. The paper concludes in Section 7.

## 2. RELATED WORK

Mobile computing has been the subject of much research in recent years and a lot of effort has been devoted to the design of data broadcast algorithms [1, 4, 5, 7, 8, 14, 23]. Unfortunately, the important issue of broadcasting consistent data items has not received adequate attention. Uncontrolled data broadcast may result in the client reading inconsistent data items. Providing consistent data items to transactions is one of the most important requirements of a transaction processing system. However, traditional concurrency control protocols are not suitable for mobile computing systems due to their heavy overhead for detecting data conflicts in a mobile environment [15, 21]. To our knowledge, few studies until now have been done in this area. One suggested solution is to relax the consistency requirement. In [15], 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. A certain degree of inconsistency is allowed among data items at different clusters.

In [20], 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. In 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.

Another method to detect the non-serializability problem is to broadcast serialization graphs [18]. Each client maintains its local serialization graph to ensure that the schedules of its committed transactions are serializable with the update transactions at the broadcast server. 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 listen to the transmission channel continuously 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 graph at the client even when it is disconnected. The mobile network is unreliable and disconnection is frequent. When disconnection occurs between a mobile client and its base station, the mobile client cannot obtain the updated serialization information about its transaction, making it virtually impossible to ensure the serializability of transaction execution.

A method similar to the broadcast serialization graph method is the invalidation method [16] in which an invalidation report will be periodically broadcast before each broadcast cycle. The invalidation report consists of a list that includes all the data items that are updated at the broadcast server during the previous broadcast cycle. The validity of data items accessed by a mobile transaction is ensured by checking with the invalidation report. A transaction has to be restarted in case any of its accessed data items is invalid.

In [16, 17], a multi-version broadcast method is proposed in which the server broadcasts previous versions of data items in addition to the committed version of the data items at the last broadcast cycle. When a mobile transaction wants to access a data item, it will get the latest version for its first read operation. The subsequent read operations of the transaction will read the data items with the largest version number which is smaller than or equal to the data version of the first operation. By allowing a transaction to read an older version of a data item, data consistency can be ensured at the expense of currency, i.e. a mobile transaction may not receive the latest value of a data item. In order to reduce the number of versions to be broadcast and to facilitate the checking of consistency, update transactions will update the database only at the end of a broadcast cycle even though they arrive in the middle of a broadcast cycle. The number of versions to be broadcast for a data item is determined from the life-span of the transactions. It is defined as the maximum number of broadcast cycle from which the transaction reads data. Another important assumption of the multi-version broadcast method is that at least one value (the current one) is broadcast for each data item in each cycle.

The multi-version broadcast method is extended for systems with client caches [16, 17]. In addition to broadcasting multiple versions of a data item, the clients also maintain the previous

versions of data items at their caches. The number of versions to be maintained at a client cache is again determined by the life-span of the transactions. The same rule for accessing broadcast data is used for accessing cached items. The efficiency and characteristics of the multi-version broadcast method as compared with other methods such as the serialization graph broadcast has been examined in [16]. The multi-version broadcast method is very useful for systems where the mobile clients are frequently disconnected from the mobile network since the mobile clients may access the cached data items while it is disconnected. Although the data items may not be the most up-to-date values, they are consistent.

### 3. SYSTEM MODEL

The mobile computing system model consists of a broadcast server, a number of mobile clients and a mobile network as shown in Figure 1. The broadcast server communicates with the mobile clients through low bandwidth wireless channels of the mobile network. The broadcast server maintains a database. An important property of the data items in a mobile computing system is that their values can be highly dynamic as they are used to maintain the useful information in the system, e.g., last traded price of the stocks and the location of a moving object [22]. To maintain the validity of the data items, update transactions are generated to refresh the data values. It is assumed that the updates are generated from some external capture devices, sensors, or provided from a data vendor, i.e., Reuters. Each update transaction is time-stamped which is its generation time by the sensor device. The time-stamp is used to represent at which snapshot of the external environment the update is generated. The time-stamp is recorded with the new value into the data item as its version number.

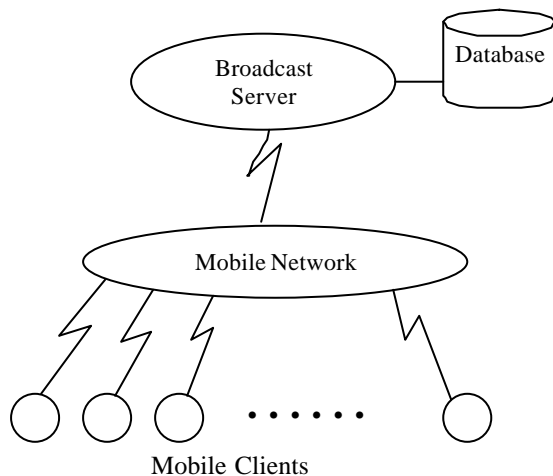


Figure 1. System Model

In the model, it is assumed that update transactions are short transactions, consisting of one to several write operations, e.g., stock quotes and news updates. Its arrival rate can be very high. It is assumed that a well-formed concurrency control protocol, such as two phase locking [3], is used for concurrency control amongst the update transactions at the broadcast server.

The broadcast server periodically broadcasts data items one by one continuously until the end of a broadcast cycle. The next

broadcast cycle follows the preceding cycle immediately. The length of a broadcast cycle may be fixed or variable depending on the adopted broadcast scheduling algorithm. A broadcast algorithm is adopted for selecting data items to broadcast. In the last few years, many efficient broadcast algorithms have been proposed based on the deadlines of transactions or the access frequencies of data items [4, 14, 23], but we shall see later, our method does not depend on the broadcast algorithm adopted and can be applied to most of these broadcast algorithms.

Mobile clients issue *mobile transactions (MT)* to access data items at the database. It is assumed that each MT consists of a sequence of data requests (read operations). To simplify the discussion, an operation is assumed to require access to one data item. Each MT is defined with a deadline. The period between the arrival time of a transaction and its deadline is called its *life-span*. It is assumed that the set of mobile transactions has the same life-span, defined as a system performance requirement. It is also assumed that a MT will become totally useless after its deadline. Thus, it will be aborted after its deadline. It is further assumed that the results of a mobile transaction will report to the requesting mobile client only at its commit time. Thus, if a mobile transaction has to be aborted or restarted from its beginning, the atomicity property still can be maintained by performing an undo operation for the transaction.

In our model, the broadcast process is modeled as a long read-only transaction, called *broadcast transaction (BT)*. The length of a BT is defined based on the life-span of a mobile transaction such that the time required to broadcast its data items is equal to the life-span of a mobile transaction. Thus, the data item set of a BT includes all the data items which are broadcast during the period from (current time – life-span of a mobile transaction) to current time. The reason of choosing the life-span of a mobile transaction to define the length of a broadcast transaction is that for normal cases a mobile transaction will be finished within its life-span. Note that the broadcast transaction actually does not possess any characteristics of a transaction. The reason for treating it as a transaction is to facilitate the discussion of mechanism and correctness of the OUFO protocol which will be introduced in the later sections.

The assumptions of the system model are summarized below:

- (1) All update transactions are processed at the broadcast servers.
- (2) The arrival rate of update transactions is high and sporadic.
- (3) All mobile transactions are *read-only*.
- (4) The read operations in a MT are *ordered*.
- (5) The result of a mobile transaction is reported to the requesting client at its commit time.
- (6) Each mobile transaction has a deadline, i.e., arrival time + life-span. It is important to complete a mobile transaction before its *deadline*. Otherwise, it is useless.
- (7) The broadcast process is modeled as a *broadcast transaction*.

## 4. CORRECTNESS CRITERIA AND PROBLEMS

### 4.1 Correctness

Before the introduction of the strategies for broadcasting consistent data items to mobile transactions, we would like to

identify the correctness requirements of the protocol first. There are two fundamental requirements for disseminating data items to mobile transactions: *consistency* and *currency*.

#### Consistency:

In this paper, we adopt serializability as the correctness criterion of database consistency [3] as it has been commonly used for conventional database systems and widely accepted in the database community. If the serialization graph of a set of transactions is acyclic, then the schedule is serializable. Since concurrency control amongst update transactions are assumed to be done by a well-formed concurrency control protocol, e.g., 2PL, the only inconsistent problem is the data conflicts between mobile transactions and update transactions. Our proposed protocol will concentrate on resolving this type of data conflicts to ensure that all mobile transactions will read consistent data values.

#### Currency:

An important property of the database in a mobile computing application is that the values of the data items are highly dynamic and the arrival rates of the updates can be very high. While data values are being broadcast from the database server to the mobile clients, updates are arriving and new values are being installed into the database. In such a dynamic environment, it is difficult to maintain a tight consistency between the status of the objects in the external environment and the corresponding values of the data items in the database, i.e., due to the delays in completion of the update process. This is especially true when the updates are generated at a remote site. For example, in a stock monitoring system, updates are generated at the stock exchange. They are sent to the data vendor which maintains a database for the stocks. Even if we assume that there is no delay in transmitting updates from their generation sites, delay may still be significant due to resource and data contention at the database server.

For most cases, stale information is much less useful to mobile clients. In order to minimize the staleness of the information, the system has to process update transactions as soon as possible and broadcast the most up-to-second information, e.g., the latest committed value of a data item. Since providing the most updated values of data items in a mobile network can be very expensive, an alternative is to bound the degree of “staleness” of the data items within a pre-defined bound. For some data items, e.g., stock information, the data values will be totally useless if they are “older” than a certain pre-defined time bound [19]. Note that this requirement has been generally ignored in most of the previous studies [18, 16, 17, 20] although it is important due to the real-time properties of the data items [19, 23].

## 4.2 Data Inconsistency Problems

In this section, we briefly describe the inconsistency problem in data broadcast using some representative examples.

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

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

- i) Broadcasts  $d_2$
- ii) MT reads  $d_2$
- iii) U updates  $d_5$

- iv) U updates  $d_2$
- v) Broadcasts  $d_5$
- vi) MT reads  $d_5$

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

**Example 2** A MT may conflict 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>1</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 broadcasts  $d_2$
- ii) MT reads  $d_2$
- iii)  $U_1$  updates  $d_2$
- iv)  $U_1$  updates  $d_1$
- v)  $U_2$  updates  $d_1$
- vi)  $U_2$  updates  $d_5$
- vii) Broadcast broadcasts  $d_5$
- viii) 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 cycles.

A MT may not be able to find all its required data items in a single broadcast cycle. Non-serializability may occur over more than one broadcast transaction. For example if the schedule is:

- i)  $d_2$  is broadcast in first broadcast cycle
- ii) MT reads  $d_2$
- iii) End of the first broadcast cycle
- iv) U updates  $d_5$
- v) U updates  $d_2$
- vi)  $d_5$  is broadcast in the second broadcast cycle
- vii) MT reads  $d_5$

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

## 5 ORDERED UPDATE FIRST WITH ORDER (OUFO)

### 5.1 Method Overview

In [12], the *Update First with Order (UFO)* protocol is proposed for broadcasting consistent data items to mobile transactions with unordered read operations, i.e. each mobile transaction consists of a set of read operations and the operations can be executed in any order. In this paper, we propose an extension of the UFO protocol, which we call *Ordered UFO (OUFO)* for mobile transactions in which the read operations are ordered. The issues of accessing consistent cached data items and processing operations under disconnection will also be addressed.

<sup>1</sup> If a transaction  $T_j$  reads an uncommitted data item from another transaction  $T_i$ ,  $T_j$  will be dependent on  $T_i$ .

The basic principle of the OUFO protocol is to ensure that if a data conflict occurs between a broadcast transaction (BT) and an update transaction (U), the serialization order between them will always be  $U \rightarrow BT$ . Since mobile transactions (MT) read data items from broadcast transactions, the serialization order between broadcast transactions and mobile transactions is always  $BT \rightarrow MT$ <sup>2</sup>. Thus, the serialization order between update transactions and mobile transactions will always be  $U \rightarrow MT$  and the schedules will be serializable. (The correctness of the OUFO protocol will be discussed in more detail in Section 5.5.1.)

Basically, the OUFO protocol consists of two parts:

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

## 5.2 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 of a transaction are written in a private workspace of the transaction instead into 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 which is performed in a critical section. 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 the 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 and delay of the broadcast transactions. If a broadcast transaction wants to read a data item, which is already locked by an update transaction during its update phase, 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

<sup>2</sup> Note that since both of the mobile and broadcast transactions are read-only, normally there should not be any data conflicts between them. However, to facilitate the discussion of the protocol OUFO, we assume they may have data conflicts so that transitive dependency relationships may occur between mobile transactions and update transactions.

comparing the write set of the update transaction with the read set of the broadcast transaction, the system can easily determine whether there is any data conflict between them.

## 5.3 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 details of the algorithms at both the broadcast server and mobile clients are shown in the following sections.

### 5.3.1 Algorithm at the Broadcast Server

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  $d_i$ 
endif
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 as described in Section 3, the set of data items in the 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. 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.

### 5.3.2 Algorithm at the Mobile Client

The data items requested by a MT is represented by a sequence of read operations. Processing of a MT starts from the first read operation in the sequence. Each data item received from a BT is matched with the requesting data item of the executing operation. If there is a match, the MT will read the data item and the operation will be processed. The process is repeated for the next read operation until there is no more operation in the sequence. In case a data item, which is already read, is re-broadcast while the MT is waiting for other data items, the MT will be restarted from the operation which requests that data item. It will use the re-broadcast value for the execution of the operation. There are two reasons for the restart. Firstly, it is to ensure the serializability order  $U \rightarrow MT$ . The second reason is to provide the most updated data values to the mobile transactions. The algorithm at the mobile client is shown below where it is assumed that the MT reads all its data items from BT and there is no cache at the client. We will discuss the case of accessing cached data items in the next sub-section.

```

 $d_c$  = the data item required by the first operation in  $L_{MT}$ 
loop until  $L_{MT}$  exhausted
    read data item  $d_i$  from BT ( $d_i$  is the currently broadcasting item)

```

```

if    di = dc
then  MT processes di and updates LMT
else
    if  di ∈ SMT
        MT repeats the processing on di, updates LMT
        Restarts its execution from the operation
        which requires di.
    endif
endif
dc = the data item requested by the next operation in LMT
end loop
where LMT = sequence of read operations in MT
      SMT = set of data items have been accessed by MT

```

## 5.4 Consistency and Currency Checking for Cached Data

### 5.4.1 Benefit of Caching Data items

If a mobile transaction reads all its requested data items directly from the broadcast transaction, the mobile transaction may need to wait for a long time. The waiting time will depend on the broadcast algorithm adopted and the size of the database if a flat broadcast disk is used. Another potential performance problem of OUFO is that a mobile transaction has to be restarted if any one of its previously accessed data items is broadcast again before its commitment. Restarting a mobile transaction will greatly increase its response time as it has to wait for the data items from the broadcast transaction again.

To solve these problems and reduce the restart overheads, a mobile client may maintain some data items at its cache. In case a mobile transaction is restarted due to the arrival of a new version of one of its accessed data items, the restarted transaction can immediately access the data items from the cache instead of waiting for them again from the broadcast transaction. After a mobile transaction has accessed a data item, the data item will also be placed in the client cache. If the cache is full, a cache replacement algorithm, e.g., the Least Recent Used (LRU) method, will be invoked. If a mobile transaction later finds its required data item in the client cache, it will access the data item immediately, resulting in a much shorter access delay. It is assumed that an *auto-refreshment scheme* is adopted to maintain the validity of the cached data items. Whenever a data item is being broadcast and the cache has a copy of the data item, the cached copy will be refreshed with the version currently broadcast.

### 5.4.2 Invalidation Report Based Consistency Checking

Although caching data items can effectively improve the performance of OUFO, a transaction has to determine the currency and consistency of a cached item before accessing the data item. In the following sub-sections, we describe an invalidation report method to combine with the OUFO protocol so that the consistency of the data items accessed by mobile transactions can be ensured.

#### 5.4.2.1 Generation of Invalidation Report

An invalidation report is prepared and periodically broadcast by the broadcast server. An invalidation report includes the latest update time-stamps and identities of the set of data items, which

are updated during the interval from (current time – report duration) to current time, i.e. the updates occur in the last report duration. It is assumed that a data item, whose update time is beyond the report duration, will be useless and too “old” to be useful. Each invalidation report will be shifted by a period, called report period, from the last report time. The report duration is chosen to be much greater than the report period. Using such a sliding window approach for generating the reports can help to solve the problem of validation under disconnection. A mobile transaction can still validate its accessed data items if it has disconnected from the network for a period not longer than the report duration since the identity of an updated data item will be repeated in several invalidation reports for a period equal to the report duration. Note that even though a large value is used for the report duration, the report size will not be increased greatly since its size is also depends on the number of updates occurred during the report period. In the worst case, the report will include all the data items which are updated in the report duration.

For example in Figure 2, data items  $x$ ,  $y$  and  $z$  are broadcast at a time prior to the broadcast transaction and are within the report duration. If they are updated within the current report duration (e.g., data items  $x$  and  $z$ ), the invalidation report will include them. If a data item is updated several times in the period, only the last updated time-stamp will be included.

#### 5.4.2.2 Validation

It is assumed that whenever a mobile client puts a broadcast data item into the cache, it will also record down the broadcast time of the data item. Based on the broadcast time, the cached data items are divided into two groups: newest version and unknown version. A data item belongs to the newest version group if its broadcast time + length of the broadcast transaction > current time. Otherwise, they are classified into the unknown version group.

When a transaction wants to access a data item, the transaction can access the data item immediately if it can be found at the cache. Otherwise, the data item must be retrieved from the broadcast transaction. If a mobile transaction has completed all its operations, it will check the validity of its accessed data items. If all of them belong to the newest version group, the transaction can commit immediately. Otherwise, if any of them belongs to the unknown version group, the mobile transaction cannot commit until it receives an invalidation report and has validated its accessed data items.

Validation is done by comparing the time-stamps of its accessed data items at the cache at the time when it accesses the data items with the time-stamps of the data items in the invalidation report. A data item is invalid if it is found in the invalidation report and the time-stamp of the same data item in the validation report is greater. For this case, the mobile transaction has to be restarted from the operation, which has read the invalid data item. Note that if we want to increase the number of data items belonging to the newest version group so as to reduce the blocking delay of mobile transactions, we can increase the length of the broadcast transaction. Essentially, it means to use a larger set of data items for checking of data re-broadcast between the update transactions and the broadcast transaction. Of course, the tradeoff is between smaller number of mobile transactions needed to validate against the invalidation report, and a larger number of re-broadcasts. The

decision can be based on the total re-broadcast overhead. If it is low, a longer broadcast transaction may be used.

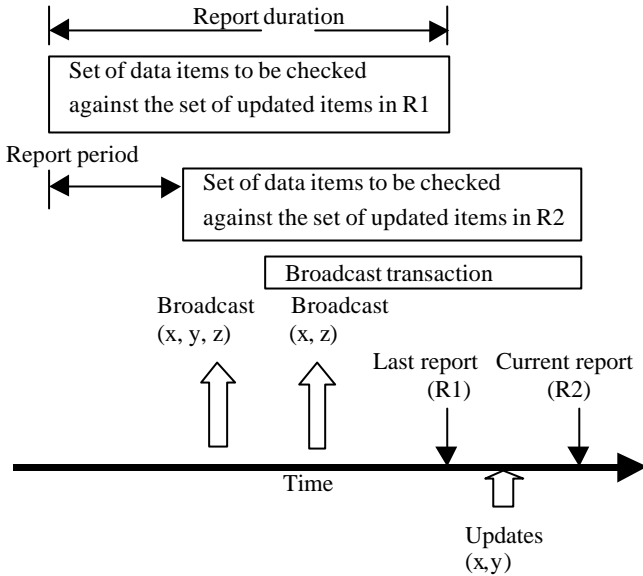


Figure 2. Generation of the Invalidation Report

A practical problem in implementing the periodic invalidation report is the definition of report period. This parameter actually represents a tradeoff between amount of bandwidth (which is a valuable resource) used for broadcasting the invalidation report and the waiting time of transactions for the report. Since the report size should be small and is not expected to consume much bandwidth, the period for broadcasting the report can be set to a small value in order to reduce the waiting time for validation.

## 5.5 Correctness, Implementation Overhead and Characteristics

### 5.5.1 Correctness

**Theorem.** The schedules of the committed transactions produced from OUFO protocol are always serializable.

**Proof:** The proof consists of two parts. Firstly, we will show that the serialization order between all committed mobile transactions and their conflicting update transactions will always be  $U \rightarrow MT$ , if the mobile transactions read the conflicting data items from the broadcast transaction. In the second part, we will prove that the serialization order between mobile transactions and update transactions will always be  $U \rightarrow MT$ , if the mobile transactions read the conflicting data items from the cache.

Let  $MT$ ,  $BT$  and  $U$  be mobile transaction, broadcast transaction and update transaction, respectively. If there is a data conflict between  $BT$  and  $U$ , and the broadcast of the conflicting data item is before the update of the data item, the serialization between  $BT$  and  $U$  will be  $BT \rightarrow U$ . According to OUFO, the conflicting data item will be re-broadcast immediately and the serialization order will be reset to  $U \rightarrow BT$ . If the broadcast is after the update, the serialization order will be  $U \rightarrow BT$ . Since  $MT$  reads data items from  $BT$ , their

serialization order will always be  $BT \rightarrow MT$ . Since serialization order is transitive, the serialization order between  $MT$  and  $U$  will always be  $U \rightarrow MT$  for the data conflict between  $MT$  and  $U$  and the conflicting data item is broadcast during the length of the broadcast transaction.

If  $MT$  reads a data item, which is at the cache or if the data item is broadcast prior to  $BT$ , the serialization order between  $MT$  and  $U$  may be  $MT \rightarrow U$ . If  $MT \rightarrow U$ ,  $MT$  will be restarted after checking with the invalidation report. Thus, the serialization order between  $MT$  and  $U$  will always be  $U \rightarrow MT$ . For a non-serializable schedule, the serialization graph must be cyclic, e.g., there must be an edge such that  $MT \rightarrow U$ . This contradicts to the OUFO. Q.E.D.

It is obvious to see that the currency of data items observed by mobile transactions can be maximized due to the data re-broadcast mechanism of OUFO since all the data items which are broadcast within the length of the broadcast transaction are the most updated version.

### 5.5.2 Implementation Overhead and Characteristics

In this section, we discuss the implementation overhead and characteristics of the OUFO protocol. The OUFO protocol does not require any changes in the mobile clients except in getting re-broadcast data items and maintaining the cached items. The main overheads of the OUFO protocol are:

- (1) division of the execution of update transactions into two phases;
- (2) checking of the data sets of a broadcast transaction and an update transaction whenever an update transaction wants to enter the update phase;
- (3) generation of the invalidation report; and
- (4) re-broadcast of conflicting data items if these items are broadcast before the start of the update phase of the update transaction.

Dividing the execution of update transactions into two phases is trivial and should not incur much additional overhead. (This is similar to the deferred update approach [3].) The overhead for checking conflicting data sets and the probability of data conflict should be low as the number of data items to be updated by an update transaction is usually small. To speed up the checking process, the data items to be updated may be sorted according to their IDs.

The main overhead of the protocol is data re-broadcast. The number of re-broadcast depends on the probability of data conflict which in turn depends on the broadcast schedule, arrival rate and the update pattern of the update transactions.

The algorithm at the mobile clients is simple and does not incur much additional overhead for checking. The only additional work is to replace the old version of a data item in case it is re-broadcast, record down the broadcast time and check with the invalidation report in case a transaction has accessed some data items which belong to the unknown version group.

The main advantages of OUFO as compared with the multi-version broadcast method proposed in [16, 17] are summarized below. The performance between the two methods will be examined in details in the performance evaluation experiments as reported in Section 6.

**Consistency.** Both OUFO and the multi-version broadcast method can ensure that the execution schedules of the transactions are serializable.

**Currency.** OUFO can maximize the currency of the data items read by mobile transactions by re-broadcast. However, the data items provided from the multi-version broadcast method are usually not the most updated versions. In general, it can only provide the committed values at the last broadcast cycle as all the database updates are performed at the end of a broadcast cycle.

**Main Overhead.** The main overhead of multi-version broadcast are due to (i) broadcasting the multiple versions of data items and (ii) keeping multiple versions of data items in the cache. The first factor significantly increases the broadcast overhead and the second factor reduces the cache hit probability since the client cache is filled with multiple versions of the same data items rather than distinct items. This can be a serious drawback in a mobile environment, where cache size can be quite small and must be utilized efficiently. On the other hand, the main overhead of OUFO is due to data re-broadcast.

**Broadcast algorithm.** The multi-version broadcast requires each cycle to include all the data items in the database, while OUFO can be applied with many other broadcast algorithms which may only broadcast a subset of the data items in the database.

## 5.6 Disconnection

An important property of mobile networks 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 OUFO nor creates any problem regarding the correctness of the OUFO 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 impact of network disconnection on OUFO is that at the time of re-broadcast a mobile client may be disconnected from the network. For this case, the mobile client cannot get the new version of the data item and the serializability order between the mobile transaction and the update transaction cannot be reversed. If they have any further data conflicts, the resulting schedule may be non-serializable. Therefore, once a mobile client has been temporarily disconnected from the network, its mobile transaction may consider all its accessed data items to belong to the unknown

version group and have to be validated with the invalidation report. Of course, if the disconnection is long, i.e., greater than the check window size, a mobile transaction will not be able to identify the validity of its accessed data items from the invalidation report. For this case, all the cached data items will be considered to be invalid. Discarding all the cached data items for this case should not significantly affect the system performance since they are quite "old". It is likely that most of them are out-dated.

## 5.7 Broadcast Index Structure

An important issue in the design of data broadcast algorithm is how to minimize the tune in time for getting broadcast data since this will affect the power consumption of the mobile machines. As suggested from previous works on the design of data broadcast algorithm, an index for the organization of broadcast cycle can be inserted at the head of each broadcast cycle. Each mobile client tunes into the index first. Based on the broadcast index, it can then determine when the required data items of its mobile transaction will be broadcast. Before the broadcast of the data items, it may turn to doze mode in order to minimize the power consumption of the mobile machine and wake up just before the broadcast of its required data items.

The re-broadcast mechanism of the OUFO protocol will affect the efficiency of broadcast index since the broadcast of a data item may be delayed due to re-broadcast. In order to minimize the impact of data re-broadcast on the broadcast schedule, we can set a maximum broadcast bandwidth for data re-broadcast, i.e., 5%, in each broadcast cycle. Therefore, the maximum additional delay of a data item from its original broadcast schedule will be bounded by this maximum broadcast bandwidth, i.e., equals to 5% of a broadcast cycle.

However, setting a maximum re-broadcast bandwidth on the other hand will affect the correctness of the OUFO protocol since the conflict order cannot be reversed in case there is a conflict between an update transaction and a broadcast transaction after the maximum re-broadcast bandwidth is reached. To ensure the correctness of OUFO, once the maximum re-broadcast bandwidth has been reached, the broadcast server may broadcast two additional pieces of information in the broadcast cycle. Firstly, the system broadcasts a signal periodically to inform the just to awaken mobile clients that re-broadcast has stopped so that the currently executing mobile transactions need to check with the next invalidation report for their accessed data items. The period for broadcasting the signal should be very short such that all mobile transactions will be informed before its commitment. Secondly, for any conflicts between an update transaction and the broadcast transaction, the broadcast server may broadcast the identities of the conflicting items instead of the values of the items as in the original OUFO protocol. For those mobile clients, which are not in doze mode, they can determine the validating of the cached data items and the data items accessing by their transactions based on the broadcast data identity. Since the identity of is very small and should not affect the broadcast schedule. After ensuring the currency of its accessed data items, a mobile transaction may commit immediately without waiting for the invalidation report.

Since the value of the maximum re-broadcast bandwidth will affect the power consumption in the mobile machines, it should be chosen based on the probability of data conflicts (number of re-

broadcast), the amount of power saved in the doze mode as compared to the power consumption level in the active mode, and the total power supply of the mobile machines. If the power supply is not a highly critical issue, a larger maximum re-broadcast bandwidth may be used in order to gain the advantages of the re-broadcast mechanism of the OUFO protocol.

## 6. PERFORMANCE STUDIES

In this section, we perform a simulation study to investigate the performance of the OUFO protocol and compare it with the multi-version broadcast with multi-version cache method (MV), and the periodic invalidation report method (IR). The reason for comparing OUFO with these two algorithms is that they represent two extremes between the tradeoff of concurrency and currency. As shown in [16], MV provides high concurrency but low currency while the IR provides high currency but low concurrency. Following the definition in [16], the update in both MV and IR will be performed into the database at the end of a broadcast cycle. In MV, the number of versions to be broadcast is the number of versions created within the period of (current time – life-span of a mobile transaction) to current time. Similar to the experiment setting in [16], 50% of the client cache is reserved for new versions of the data items and the remaining 50% are for old versions. The cache will use the least recent policy for cache replacement. Note that although in [17] it was found that if the previous versions of data items were put in the secondary storage, the performance could be better, this way of storing the old versions may not be practical in many mobile machines, i.e., palm and handheld PC, do not have secondary memory. The implementation of IR is similar to the invalidation scheme used in OUFO except that the generation of the invalidation report is fixed at the end of a broadcast cycle and all updates are performed at the end of a cycle.

We have implemented a simulator using CSIM-18, which is a simulation language based on the programming C language, to model the three protocols, OUFO, MV and IR. Extensive simulation experiments have been performed to investigate the impact of different factors, such as update rate, data access pattern of the update transactions, cache size and database size on the performance of the protocols.

### 6.1 Simulation Model

Basically, the model consists of three main entities and three processes as shown in Figure 3. The three main entities are the broadcast server, the air media and the mobile clients. Two of the three processes are at the broadcast server: the broadcast process and the server update process. The last main process is the client process at each mobile client for simulating the mobile transactions.

#### Broadcast Server

The database is located at the broadcast server. It is assumed that all the items in the database have the same size and each item has a unique identifier and an update time-stamp indicating its last update time.

#### Air Media

Air media is a collection of air channels. In our experiments, we assume that both scheduled broadcast items and re-broadcast items share the same broadcast channel.

#### Mobile Clients

There are a large number of mobile clients in the system, which generates read-only transactions. Each client will generate one transaction at a time and will generate the next mobile transaction after an exponentially distributed think time upon the completion of a transaction.

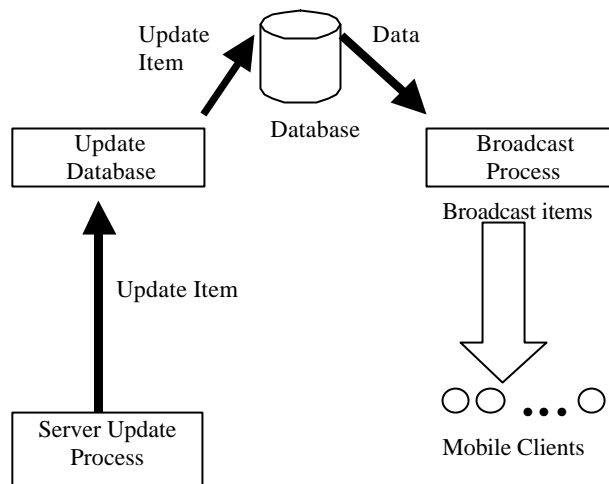


Figure 3. Simulation model

#### Broadcast Process

The broadcast process defines the broadcast schedule, e.g., the selection of data items from the database for broadcast. We assume a flat broadcast disk model to simplify the model and analysis. Thus, a broadcast cycle includes all the data items in the database.

#### Server Update Process

The server update process generates update transactions. The data items to be updated by an update transaction are assumed to follow the Zipf distribution which is commonly used in other studies in the area [12, 16, 17]. The server update process resolves data conflicts among the update transactions by using 2PL.

#### Mobile Client Process

The mobile client process generates mobile transactions. It is assumed that each mobile transaction is defined with a deadline which is set to be its arrival time plus the life-span of the transaction. It is assumed that the usefulness of completing a mobile transaction will be significantly affected after its deadline and will become totally useless after its deadline. Thus, a transaction will be aborted after its deadline. Each transaction consists of a sequence of operations and each operation requires access to one data item. It is assumed that the data access pattern of the operations follows the Zipf distribution. Once generated, the mobile transaction will try to access the required data item of its first operation from the cache. If it does not find the data item at the cache, it will listen to the broadcast cycle continuously for the data item. After getting its required data item, the transaction will access the CPU for computation. Then, it will process its next operation. When a mobile transaction has completed all its operations, it commits.

## 6.2 Model Parameters and Performance Measures

### 6.2.1 Model Parameters

Table 1 lists the model parameters and their baseline values.

Parameter	Baseline value
Database size	1000 data items
No. of mobile clients	100
Broadcast rate	20 data items / sec
Cache size	50 data items
Cache replacement scheme	LRU
Data access distribution (both mobile and update transactions)	Zipf distribution
Degree of skew in access distribution	1.0
Offset between the data access distribution of mobile and update transactions	10%
No. of operations in a mobile transaction	1 to 4
No. of write operations in an update transaction	1 to 2
Invalidation report generation period (for OUFO and IR only)	50 sec
Report duration	1000 sec
Life-span of a mobile transaction	200 sec
Mean think time of a mobile client	10 sec
Mean inter-arrival time of update transactions	0.1 to 4 sec

**Table 1. Model parameters and their baseline values**

### 6.2.2 Performance Measures

The primary performance measures are *miss rate*, *mean response time* and *stale access rate*. Miss rate is defined as the number of transactions, which miss their deadlines (aborted) divided by the total number of transaction generated. It indicates the capability of meeting the timing requirements of the transactions. Stale access rate is defined as the number stale data access, e.g., reading a data value which is different from that of the latest updated value, divided by the total number of data accesses. It is an important measure for the currency of the data items observed by the mobile transactions.

In addition to the primary measures, we also measure other statistics such as *the cache hit rate*, *broadcast overhead*, and *restart rate* to investigate the performance characteristics of the protocols. Broadcast overhead captures the amount of broadcast bandwidth consumed by the protocols. Broadcast overhead in OUFO is the percentage of bandwidth used for re-broadcasting data items due to data conflicts and the overhead for broadcasting the invalidation report. In MV, the broadcast overhead is the percentage of broadcast bandwidth used for broadcasting all versions of data items, except the newest version. Restart rate is defined as the number of transaction restarts over the total number of transactions completed. In OUFO and IR, a mobile transaction may be restarted by checking with an invalidation report. In OUFO, a mobile transaction may also be restarted due to re-broadcast.

## 6.3 Performance Results and Discussion

### 6.3.1 Impact of Update Workload

Figures 4 to 9 show the impact of update interval on the performance of the three protocols, OUFO, MV and IR, when different skew coefficients (0.5 and 1.0) are used for the Zipf distribution which is adopted for both update transactions and mobile transactions. An offset of 10% is set between the Zipf distributions for the mobile transactions and the update transactions so that they do not have exactly the same set of hot data items. Note that increasing the skew coefficient will decrease the number of hot data items. As can be seen in Figure 4 and Figure 5, the mean response time and miss rate of the three protocols decrease gradually with an increase in update interval. This is consistent with our expectation. Increasing the update interval reduces the update workload and the data conflict probability between the update and mobile transactions will be lower. The consequence will be lighter re-broadcast workload in OUFO; lower multi-version broadcast overhead in MV; and smaller number of restarts in IR. These can be observed in Figure 6 and Figure 9.

As depicted in Figure 4 and Figure 5, the performance (mean response time and miss rate) of OUFO is much better than that of MV and IR. The better performance of OUFO as compared with MV is mainly due to a higher cache hit rate (as shown in Figure 7) although the broadcast overhead of OUFO is higher than that of MV as shown in Figure 6 especially when the update workload is heavy. In MV, multiple versions of a data item may be maintained at the cache. This will significantly reduce the number of data items at the cache, leading to a smaller cache hit rate. The broadcast of multiple versions of data items also increases the broadcast overhead. Another important advantage of OUFO over MV is that it can provide a higher data currency to the mobile transactions as shown in Figure 8. The stale access rate of OUFO remains zero while the stale access rate of MV is up to 25% when the workload is heavy and the skew coefficient equals 0.5. This indicates that many mobile transactions observe "old" values under MV. The drop in stale access rate when skew coefficient = 1.0 is due to the low update frequency of the data items.

Although the performance of IR is similar to MV when the skew coefficient equals 1.0, IR is much better than MV when the skew coefficient equals 0.5. This is mainly due to a much higher cache hit rate and lower broadcast overhead, although its restart rate is higher than MV. The stale access rate of IR is also much lower as shown in Figure 8 since the cache only maintains the most update versions of the data items. Out-dated versions are invalidated by invalidation report periodically.

Figure 4 and Figure 5 also show that the performance of OUFO, IR and MV is better when the skew coefficient is larger. The skew coefficient affects the number of hot data items. Reducing the skew coefficient (e.g., from 1.0 to 0.5) increases the number of hot items. This reduces the cache hit rate as shown in Figure 7.

Figure 10 and Figure 11 show the performance of the protocols when the offset of the Zipf functions for the update and mobile transactions is set to zero, e.g., the two types of transactions have the same set of hot data items. Consistent with the previous results, the performance (mean response time and miss rate) of OUFO is much better than MV and IR as shown in the figures. However, the performance of IR becomes much worse than MV due to large number of restarts.

Figure 12 and Figure 13 show the results when the length of a mobile transaction is increased from 1 to 4 operations to 4 to 8 operations. Consistent with the results in previous figures, OUFO gives the best performance. Increasing the length of the transactions, the overhead of the protocols and the restart probability of a transaction under OUFO and IR will be higher. Although the number of restart will be larger in the OUFO, the restart cost is low as a restarted transaction can find its required data items from the cache.

### 6.3.2 Impact of Cache Size

Figures 14 to 16 show the performance of the protocols at different client cache sizes. As expected, increasing the cache size improves the performance of the protocols as shown in Figure 14 and Figure 15. It is mainly due to a higher cache hit rate as shown in Figure 16.

Consistent with the results in the previous section, the performance of OUFO is much better than MV and IR as shown in Figure 14 and Figure 15, e.g., smaller mean response time and lower miss rate. The better performance of OUFO is again due to higher cache hit rate (Figure 16) and lower broadcast overhead. The difference in their performance is greater when the update workload is heavier. The poor mean response time of IR is mainly due to the restart nature of the protocol.

### 6.3.3 Database Size

Increasing the database size increases the length a broadcast cycle as we are using a flat broadcast disk (where a broadcast cycle includes all the data items in the database.) At the same time, the number of hot data item will also be larger due to a larger database. Thus, the conflict probability will be lower. Figures 17 to 19 show the results when the database size is increased from 1000 to 2000 data items. It can be observed that the performance of the protocols degrades when a larger database is used. It is due to (1) a longer broadcast cycle so that the mobile transactions have to wait longer for their data items if they cannot find their required data items in the cache and; (2) a lower cache hit rate as the number of hot data items is larger (Figure 19). The performance of OUFO remains consistently better than MV and IR in mean response time, miss rate and stale access rate even if the database size is larger.

## 7. 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 largely ignored. If the broadcast of data items and the execution of update transactions (which are important for maintaining the validity of the data items at the database) are uncontrolled, the mobile transactions may observe inconsistent data values. In this paper, we study the issue of concurrency control between update transactions and mobile transactions, which consist of ordered read-only operations. Our proposed protocol is called *Ordered Update First with Order (OUFO)*. Although the protocol is simple, it can effectively maintain the schedule between update and mobile transactions to be serializable and at the time maximize the currency of the data items observed by the mobile transactions. Unlike the multi-

version broadcast, another efficient protocol for broadcasting consistent data items to mobile transactions, OUFO can be applied to different broadcast algorithm and its impact on the mechanism in the mobile clients is minimal, especially in cache data management. This means that OUFO can be implemented easily in most mobile broadcast systems.

In order to investigate the performance characteristics of the OUFO protocol, a simulation model of a mobile computing system with data broadcast is implemented. Experiments are performed to investigate the impact of different system parameters on its performance as compared with the multi-version broadcast and multi-version cache method and the invalidation method. The results show that OUFO significantly outperforms the other two efficient methods in terms of mean response time of mobile transactions and miss rate. Another important advantage is that the data items under OUFO are generally much more current than that in multi-version broadcast method.

## 8. ACKNOWLEDGEMENT

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 a grant from CityU (Project No. 7001069).

## 9. REFERENCES

- [1] Acharya, S., Alonso, R., Franklin, M., Zdonik, S., "Broadcast Disks: Data Management for Asymmetric Communications Environments", in *Proceedings of ACM SIGMOD*, 1995.
- [2] Acharya, S., Franklin, M., Zdonik, S., "Balancing Push and Pull for Data Broadcast", in *Proceedings of ACM SIGMOD*, Tucson, Arizona, May 1997.
- [3] Bemstein, P.A., Hadzilacos, V., Goodman, N., *Concurrency Control and Recovery in Database System*, Addison-Wesley Publishing Company, 1987.
- [4] 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*, pp. 124-133, 1997.
- [5] Fernandez, J., Ramamritham, K., "Adaptive Dissemination of Data in Real-Time Asymmetric Communication Environments", in *Proceedings of Euromicro Conference on Real-Time Systems*, June 1998.
- [6] Franklin, M. and Zdonik, S., "Data In Your Face: Push Technology in Prospective", in *Proceedings of 1998 ACM SIGMOD Conference*, Seattle, 1998.
- [7] Hu, Q., Lee, D.L., and Lee, W.C., "A Comparison of Indexing Methods for Data Broadcast on the Air", in *Proceedings of 12th International Conference on Information Networking*, 1998.
- [8] Hameed, S. and Vaidya, N.H., "Efficient Algorithms for Scheduling Single and Multiple Channel Data Broadcast", Technical Report 97-002, Department of Computer Science, Texas, A&M University, Feb. 1997.

- [9] Imielinski, T. and Badrinath, B.R., "Mobile Wireless Computing: Challenges in Data Management," *Communications of the ACM*, vol. 37, no. 10, Oct. 1994.
- [10] Imielinski, T., Viswanathan, S. and Badrinath, B.R., "Data on Air: Organization and Access", *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 3, pp. 353-372.
- [11] Kayan, E., Ulusoy O., "An Evaluation of Real-Time Transaction Management Issues in Mobile Database Systems", *The Computer Journal*, vol.42, no.6, 1999.
- [12] 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, Feb. 1999.
- [13] Leong, H.V. and Si, A, "Data broadcasting strategies over multiple unreliable wireless channels", in *Proceedings of the 4th International Conference on Information and Knowledge Management*, pages 96-104, November 1995.
- [14] Leong, H.V. and Si, A., "Database Caching over the Air-  
*The Computer Journal*, vol. 40, no. 7, pages 401-415, 1997.
- [15] 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.
- [16] Pitoura, E. and Chrysanthis, P.K., "Scalable Processing of Read-Only Transactions in Broadcast Push", in *Proceedings of International Conference on Distributed Systems*, May 1999.
- [17] Pitoura, E. and Chrysanthis, P.K., "Exploiting Versions for Handling Updates in Broadcast Disks", in *Proceedings of Very Large Data Base Conference*, Sept. 1999
- [18] 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.
- [19] Srinivasan, R., Liang C. and Ramamritham, K., "Maintaining Temporal Coherency of Virtual Data Warehouses", in *Proceedings of Real-time Systems Symposium*, Spain, 1998.
- [20] Shanmugasundaram, J., Nithrakashyap, A., Sivasankaran, R. and Ramamritham, K., "Efficient Concurrency Control for Broadcast Environments", in *Proceedings of ACM SIGMOD International Conference on Management of Data*, Philadelphia, June 1-3, 1999.
- [21] Ulusoy, O., "Real-Time Data Management for Mobile  
*Proceedings of International Workshop on Issues and Applications of Database Technology (IADT'98)*, Berlin, Germany, July 1998.
- [22] Wolfson, O., Sistla, P., Chamberlain, S., and Tesha, Y., "Updating and Querying Databases that Track Mobile  
*Distributed and Parallel Databases Journal*, vol. 7, no. 3, 1999.
- [23] Xuan, P., O. Gonzalez, J. Fernandez & 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.

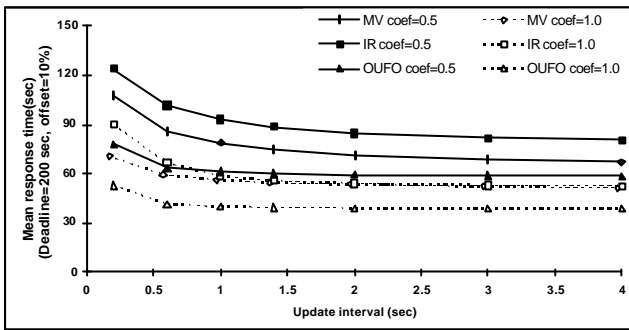


Figure 4. Mean response time Vs. skew coefficients

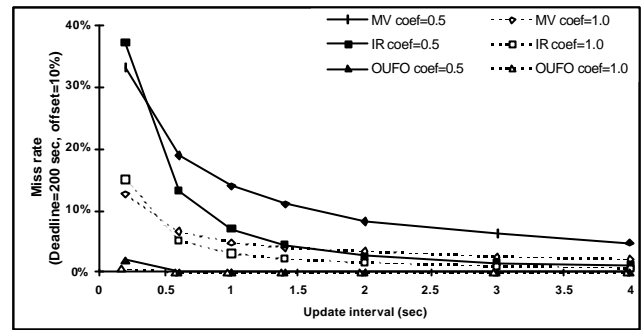


Figure 5. Miss rate Vs. skew coefficients

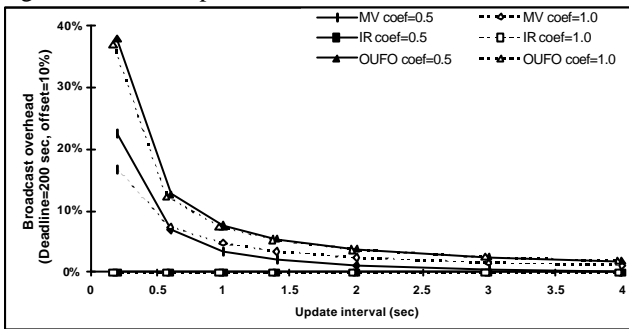


Figure 6. Broadcast overhead Vs. skew coefficients

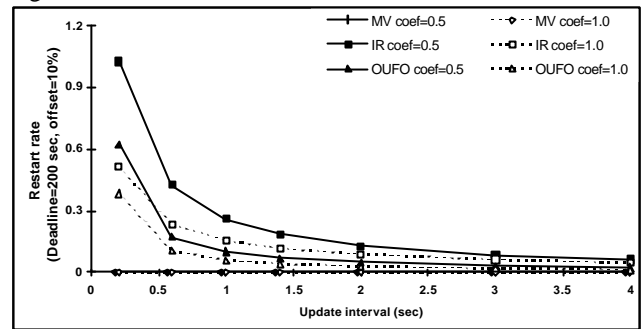


Figure 7. Cache hit rate Vs. skew coefficients

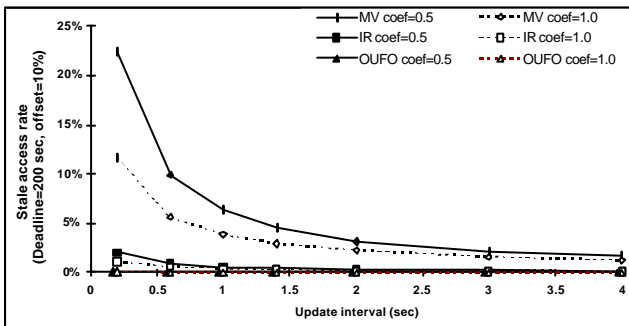


Figure 8. Stale access rate Vs. skew coefficients

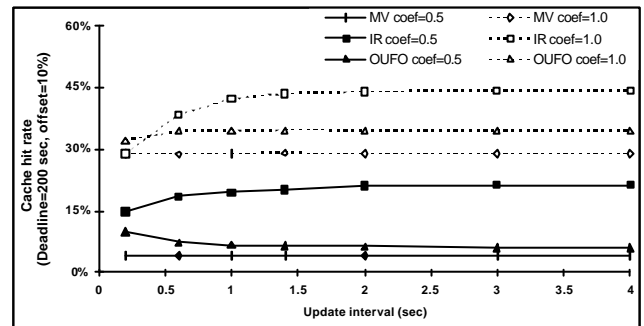


Figure 9. Restart rate Vs. skew coefficients

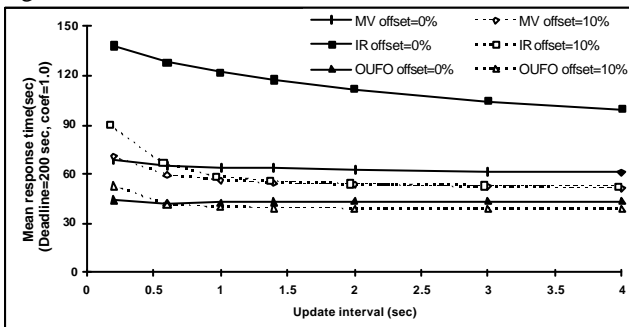


Figure 10. Mean response time Vs. offset

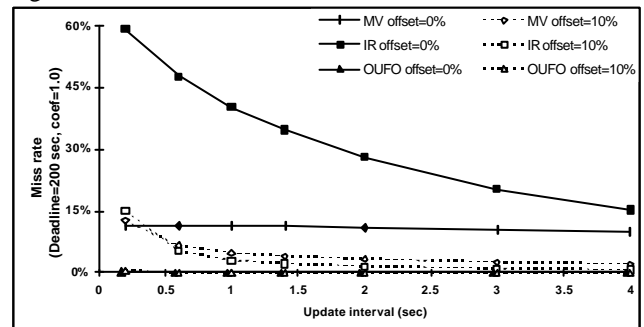


Figure 11. Miss rate Vs. offset

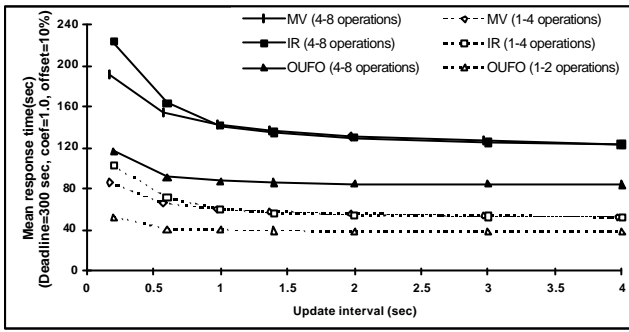


Figure 12. Mean response time Vs. length of mobile transactions

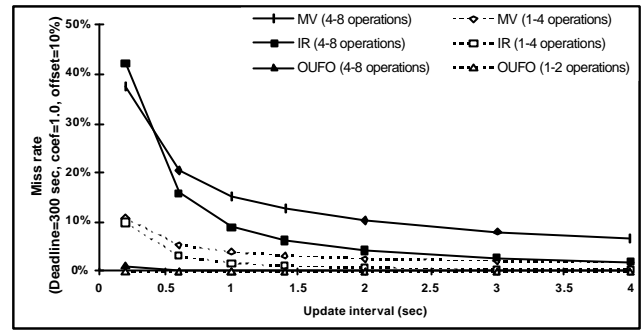


Figure 13. Miss rate Vs. length of mobile transactions

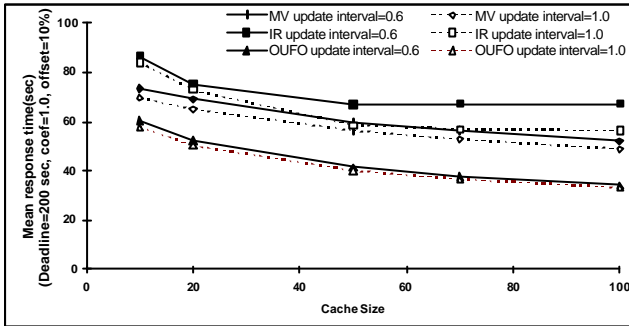


Figure 14. Mean response time Vs. cache size

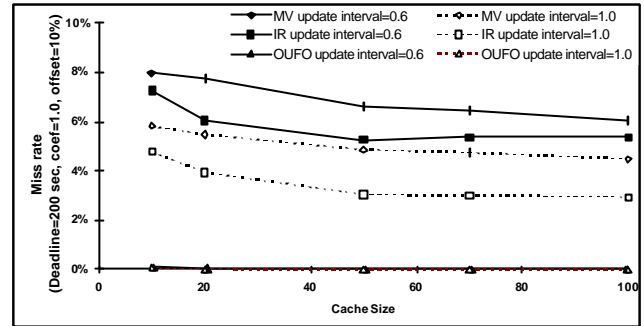


Figure 15. Miss rate Vs. cache size

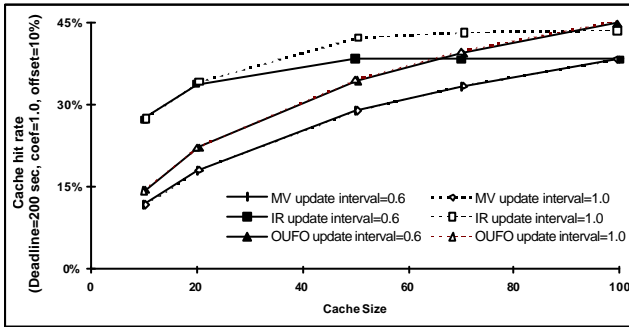


Figure 16. Cache hit rate Vs. cache size

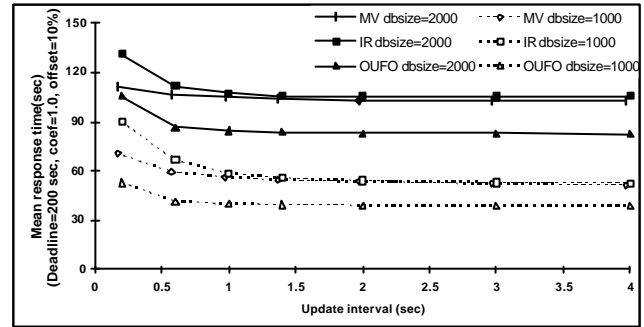


Figure 17. Mean response time Vs. database size

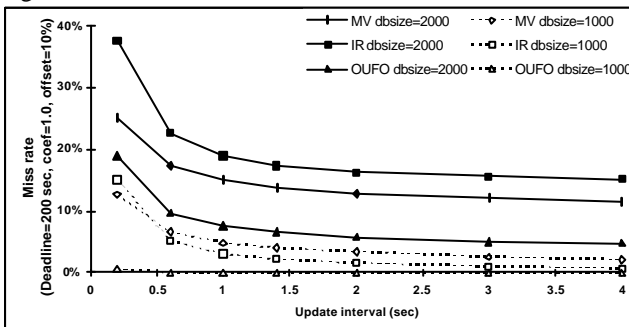


Figure 18. Miss rate Vs. database size

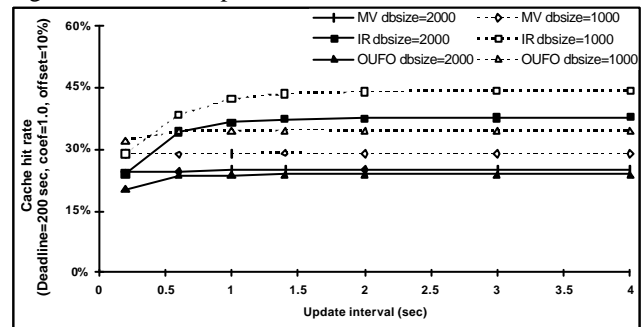


Figure 19. Cache hit rate Vs. database size