An Efficient Scheme for Processing Wireless Read-only Transactions in Data Broadcast

SangKeun Lee and Masaru Kitsuregawa
Institute of Industrial Science, The University of Tokyo, Japan
{lsk, kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract

This paper addresses the issue of ensuring consistency and currency of data items requested by wireless read-only transactions in data broadcast. To handle an inherent property in wireless data broadcast that data can only be accessed strictly sequential by users, a predeclaration-based query optimization is explored and a practical method, called PwA (Predeclaration with Autoprefetching)*, is devised for efficient wireless read-only transactions processing. The performance of the proposed method is also evaluated by an analytical study. Evaluation results show that the predeclaration technique we introduce reduces response time significantly and adapts to dynamic changes in workload.*

1 Introduction

In wireless computing, the stationary server machines are sometimes provided with a relatively high-bandwidth channel which supports broadcast delivery to all mobile clients located inside the geographical region it covers. This facility provides the infrastructure for a form of data delivery called *push-based* delivery. Push-based delivery is important for a wide range of applications that involve dissemination of information to a large number of clients. Dissemination-based applications include information feeds such as stock quotes and sport tickets, electronic newsletters, mailing lists, traffic management systems. In such applications, the server repetitively broadcast data to a client population without a specific request. Clients monitor the broadcast channel and retrieve the data items they need as they arrive on the broadcast channel.

In wireless broadcast environment, if there is a mobile client waiting for a data item, the client will get the data item from the air while it is being broadcast by the server. Thus, the cost for data dissemination is independent of client number since a data broadcast can satisfy multiple clients waiting for the same data item, resulting in a much more efficient way of using the bandwidth. It is therefore quite suitable for disseminating substantial amount of information and data to a large number of clients where bandwidth efficiency is a major concern.

An important consideration in data broadcast is to provide consistent data values to wireless transactions. In data broadcast, transactions do not need to inform the server or set any locks at the server before they access data items. They can get data items from the air while the data items are being broadcast. If updates at the server are done concurrently, however, transactions may observe inconsistent data values. This paper addresses such a consistency problem in wireless data broadcast. Another important consideration in data broadcast is to provide current, i.e. up-to-date, data values to wireless transactions. Most advanced applications in a dissemination-based environment do need to read current data items. Thus, the major challenge in this paper is *how to provide consistent and current data items to wireless transactions while speeding up their processing.*

1.1 Related Work

Recently, several approaches to consistent and current data access despite updates in wireless data broadcast have been proposed in the literature [SNS+99, PC99a, PC99b, LAC99]. A control information matrix and a serialization graph testing are used for concurrency checking in [SNS+99] and [PC99a] respectively. The work in [SNS+99] involves the development of a weaker correctness criterion called *update consistency* and the outline of mechanisms to ensure both the mutual consistency of data items and the currency of data items read by clients. In serialization graph testing [PC99a], at each broadcast cycle, the server broadcasts any updates of the graph which includes the committed transactions at the server. Upon receipt of the updates, the client integrates them into its local copy of the graph and then checks if any cycle is created. The major problem with these approaches is, however, that the large overhead is involved in maintaining control information for concurrency control and conflict detection at the server.

A simple invalidation method is presented in [PC99b], where an invalidation report is broadcast at pre-specified points (e.g. at the beginning of each broadcast cycle) during the broadcast. The invalidation report includes a list with the data items that have been updated since the previous invalidation report was broadcast. The client tunes in at the pre-specified points to read the invalidation reports. The transaction is aborted if any data item belonging to the readset appears in the invalidation report. To increase the number of read-only transactions that are successfully processed despite updates at the server, multiversion schemes are also employed in [PC99b]. The basic idea underlying multiversioning is to temporarily retain old versions of data items in a broadcast, so that the number of aborted transactions is reduced. Multiversioning with invalidation method, a variation of the multiversioning method, is also suggested in [PC99b]. In this method, after being invalidated for the first time, multiversioning method is applied. However, these multiversion schemes increase broadcast cycle length, so they impose a serious performance deterioration especially in an update-intensive environment. Update-first with order (UFO) algorithm proposed in [LAC99] checks data conflicts among broadcast transactions and up-

date transactions instead of detecting conflicts among mobile transactions and update transactions. By rebroadcasting conflicting data items, this algorithm ensures that the serialization order of a broadcast transaction is preceded by an conflicting update transaction. However, this method is vulnerable to performance degradation in case of heavy updates.

1.2 Contribution of the Paper

In this paper, we investigate a simple but robust read-only transaction processing scheme that facilitates predeclaration combined with local caching. In a traditional pull-based (i.e. client-initiated) data delivery, predeclaration technique has often been used to avoid deadlocks in locking protocols [BHG87]. In the push-based data delivery, however, predeclaration in transaction processing has a novel property that each read-only transaction can be processed successfully with a *bounded* worst-case response time. Only assuming that the server broadcasts transactionally consistent data values in each broadcast cycle, the consistency of read-only transactions is easily guaranteed. In particular, predeclaration combined with invalidation-based cache consistency maintenance can process read-only transactions successfully even when their processing is across more than one broadcast cycle. In this manner, read-only transactions are processed successfully without increasing broadcast cycle length or being considerably affected by the rate of updates at the server. The unique contributions of this paper can be summarized as follows:

- Wireless read-only transactions can be processed successfully with a bounded worst-case response time. Even in an extreme case where all data items in a database are updated in each broadcast cycle, the worst-case response time is only doubled from a broadcast cycle.
- The worst-case response time of a wireless read-only transaction is totally immune to local cache hit ratio, the number of data items it reads, and the mismatch between the needs of the transaction and the broadcast program generated by the server.
- Each wireless read-only transaction reads current data values that correspond to the database state at the completion of data acquisition of its readset, which is usually the broadcast cycle at which it commits.

The strength of our scheme is based on the assumption that a mobile client is equipped with enough local storage capacity to hold all data items needed to execute a single transaction. This assumption is valid due to the fact that even though there exists no hope to increase battery life, recent advances in the hardware indicate that processing power, main memory and local storage capacity will be increased.

1.3 Organization of the Paper

The remainder of this paper is organized as follows. Section 2 introduces the basic design principles to process wireless read-only transactions in data broadcast. Section 3 describes our PwA method used to maintain consistency and currency of transactions. Sections 4 and 5 contain an analytical study and its results, respectively. Finally, Section 6 presents our conclusions and future work.

2 Basic Design Principles

2.1 Preliminaries and Problem Statement

The server periodically broadcasts data items over a single channel to a large client population. Each period of broadcast is called a broadcast cycle or *bcycle*, while the content of the broadcast is called a *bcast*. Each client listens to the broadcast and fetches data as they arrive. This way data can be accessed concurrently by any number of clients without any performance degradation. However, access to data is strictly sequential since clients need to wait for the data of interest to appear on the channel.

Clients access data from the bcast in a read-only mode, and maintain their local caches. We assume that the cache at a client is a nonvolatile memory such as a hard disk and a client does not have a mechanism for sending messages to the server. At any given time, it is assumed that there exists a single read-only transaction in a client. Clients do not need to continuously listen to the bcast. They tune-in to read specific data items. To do so, clients must have some prior knowledge of the structure of the bcast that they can utilize to determine when the item of interest appears on the channel. In this paper, we assume that the location of each data item in the broadcast channel remains fixed and clients have sufficient storage capacity, thus an index for the data of interest may be maintained locally at each client [1].

To disseminate data via broadcasting, the server constructs a broadcast program and periodically transmits data according to the program. In a *uniform* broadcast program all data items are broadcast once in a bcycle regardless of their access frequencies. On the contrary, a *nonuniform* broadcast program favors data with higher access frequencies. Hence, in a bcycle of a nonuniform bcast, while all data items are broadcast, some will appear more often than those that are less frequently broadcast. Consider the two different broadcast organizations illustrated in Figure 1, where the server broadcasts a set of data times $\{d_1, d_2, d_3, d_4, d_5, d_6, d_7\}$ in one broadcast according to a broadcast program (d_1 is the most frequently accessed item, d_2 and d_3 are less frequently accessed ones, and d_4, d_5, d_6 and d_7 are least frequently accessed ones). While program (a) is a uniform broadcast program, (b) is a nonuniform broadcast program (refer to [AAF+95] for detail). Suppose that, in each broadcast organization, a client transaction program starts its execution at the middle of the bcycle:

$$\mathsf{IF}\ (d_2 \leq 3)\ \mathsf{THEN}\ \mathsf{read}(d_1)\ \mathsf{ELSE}\ \mathsf{read}(d_6)$$

Now, we can pose the following three questions, including

- What is the response time (in terms of the number of data items) taken to execute the transaction?

[1] Our work is also applicable to the case where some form of directory information is broadcast along with data items without loss of generality.

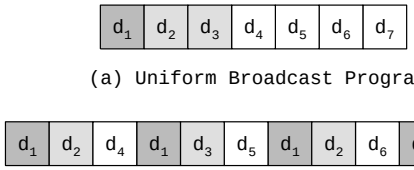

Figure 1: Broadcast Organizations

- What happens to transactional consistency if data items are being updated at the server?
- To what extent the transaction can read current data values?

In the following sections, we answer the questions raised above and their rationale.

2.2 Predeclaration and Its Usefulness

To first show that the order in which a transaction reads data affects the response time of the transaction, consider the client transaction program at the uniform bcast in Figure 1. Since both d_1 and d_6 precede d_2 in the bcast with respect to the client and access to data is strictly sequential, the transaction has to read d_2 first and wait to read the value of d_6 or d_1. Thus, the response time of the transaction is 11.5 (in case d_2 and d_1 are accessed) or 9.5 (in case d_2 and d_6 are accessed). If, however, all data items that will be accessed *potentially* by a transaction, i.e. $\{d_1, d_2, d_6\}$, are predeclared in advance, a client can hold all necessary data items with a reduced response time of 5.5. This is also true, although not so apparent, to the case of the nonuniform bcast in Figure 1. The response time of the transaction is 4 (in case d_2 and d_1 are accessed) or 3 (in case d_2 and d_6 are accessed), while the response time is 3 if predeclaration is used. Thus the use of predeclaration allows the necessary items to be retrieved in the order they are broadcast, rather than in the order the requests are issued.

Another point is that the response time of a transaction can be affected by a transaction's local processing delay. This is true even when the order in which a transaction reads data is consistent with the order in which data items are broadcast by the server. The reason is that after reading one data item, there will be a slight delay before a transaction is ready to read the next. If the next data the transaction requires is broadcast in the immediately next position in the current bcast, that data will have already passed by the time the transaction is ready for it. Thus the transaction would have to wait another bcast before that data came by again. If, however, all data items that will be accessed by a transaction are predeclared in advance, a client will have time to get ready to read before the data its transaction needs reaches itself. For these reasons, predeclaration is adopted in our method and the remaining design issues are discussed in the context of predeclaration based transaction processing.

2.3 Transactional Consistency in presence of Updates

The server broadcasts the whole content of a database which consists of a finite set of data items. We adopt *serializability* [BHG87], instead of a weaker notion in [SNS+99], as a correctness criterion. While data items are being broadcast, update transactions are executed at the server that update the values of data items broadcast. Only assumption at the server's part is that the values of data items that are broadcast during each bcycle are those produced by committed transactions. Furthermore, we make sure that each bcycle represents a consistent snapshot of the database [2]. Thus, a read-only transaction that reads all its items within a single bcycle can be successfully executed without any concurrency control overhead at all. In reality, however, most transactions will be started at some point within a bcycle, thus may have to read data items from different bcasts. In such a situation, there is no guarantee that the values they read are transactionally consistent. This is also true to the case of predeclaration based transaction processing.

One way to ensure the consistency of read-only transactions is to abort transactions that read data values that correspond to different database states. However, this kind of abort-based methods leads to intolerable transaction abort rate in case of intensive updates at the server. Another way is to control read-only transactions such that it reads consistent data items only. To achieve this, the server can broadcast multiple versions of data items so that clients could read an appropriate version. However, this kind of multiversion schemes increases a bcycle length, thereby resulting in the increased transaction response time. A better way, which is adopted in this paper, is to separate the processing of acquiring data items from that of delivering data items to transactions. This separation allows clients to retrieve the new values of invalidated data items again from the next bcast prior to transactions reading data items actually. Under this scheme, clients can deliver only consistent data items to their transactions, and so, there is no need to abort read-only transactions. Moreover, the server does not have to broadcast old versions of data items. Even if some items among acquired data items are invalidated, clients can get new data values from the next bcast.

2.4 Currency of Read-only Transactions

In terms of currency of reads, UFO algorithm [LAC99] among others can provide most up-to-date data values to transactions by allowing updates to be interleaved with data broadcast. If there is any data conflict between update transaction and broadcast transaction and the conflicting data items are already broadcast, the server re-broadcasts the conflicting items. Thus every time a data item is being broadcast, it will be the most current version. The major drawback of this

[2] This assumption was also made in the work [PC99a, PC99b]. One obvious way to satisfy this assumption is to make each bcycle represent the state of the database at the beginning of the cycle. However, if all updates of an update transaction are made solely to those items not yet transmitted over the broadcast channel at the moment, the updated values would be installed into the current bcast, thereby improving data currency without violating consistency of items within a bcycle.

approach is, however, that the large number of rebroadcast is needed if the probability of data updates is high. Invalidation method [PC99b] can also provide relatively up-to-date data values to transactions. In the invalidation method, if an invalidation report is broadcast at the beginning of each bcycle, a transaction reads the most current values as of the beginning of the bcycle at which it commits. The way to make a transaction read the most current values as of the time of its commitment is that each client reads the next invalidation report that will appear in the bcast before committing its transaction. However, the main limitation of this kind of approaches is that they suffer from the extremely low transaction throughput in an update-intensive environment. Multiversioning method [PC99b] sacrifices currency for high commitment probability of transactions, where a transaction reads the database state that corresponds to the currency point at which the transaction starts its processing. The currency of a transaction in multiversioning with invalidation [PC99b] lies at the point in which its first invalidation occurred. In contrast, with the use of both predeclaration and separation of data acquisition from delivery, a wireless read-only transaction reads the most current values as of the end of data acquisition processing, which usually corresponds to the bcycle at which it commits.

3 Supporting Wireless Read-only Transactions

In this section, we show that the use of predeclaration combined with local caching can effectively maintain the serializability of wireless read-only transactions in the environment where data items are being updated and disseminated from the server.

3.1 Predeclaration of ReadSet

The basic principle of our method is to employ predeclaration of readset in order to minimize the number of different bcycles from which transactions read data. It is particularly assumed that the information about the readset of a transaction is available at the beginning of transaction processing. For example, it is possible by using preprocessor, such as a compiler, on a client to analyze its transaction before being submitted to the client system.

We define the predeclared readset of a transaction T, denoted by $Pre\_RS(T)$, to be a set of data items that T reads *potentially*. Note that additional reads may be included to the predeclared readset due to control statements such as IF-THEN-ELSE and SWITCH statements in a transaction program. In particular, each client processes its read-only transaction, T, in three phases: (1)*Preparation phase:* it gets $Pre\_RS(T)$, (2)*Acquisition phase:* it acquires all data items belonging to $Pre\_RS(T)$ from its local cache or the bcast(s), and (3)*Delivery phase:* it delivers data items to its transaction according to the order in which the transaction requires data. The execution of read-only transactions is clearly serializable if it can fetch all data items within a single bcycle. In reality, however, a transaction is expected to be started at some point within a bcycle, thus its acquisition phase may be across more than one bcycle. In the next section, handling such a situation is addressed in the context of local caching.

3.2 Caching and Invalidation Bit Patterns

In our method, caching technique is employed in the context of transaction processing, so transaction semantics are not violated as a result of the creation and destruction of cached data based on the runtime demands of clients. To this end, the maintenance of cache consistency is based on *invalidation bit patterns* broadcast by the server. In an invalidation bit pattern, each bit corresponds to a single data item in the database (recall that the location of each data item in the broadcast channel remains fixed). A bit is set to 1 if its corresponding data item has been updated during the previous bcycle but not installed into the previous bcast. The remaining bits are set to 0s. Each bcast is preceded by an invalidation bit pattern.

During its acquisition phase, in addition to $Pre\_RS(T)$, a client keeps a set $Acq\_RS(T)$ of all data items that it has acquired from its local cache or the broadcast channel so far. Clearly, $Acq\_RS(T)$ is a subset of $Pre\_RS(T)$. At the beginning of each bcycle, the client tunes in and reads the invalidation bit pattern broadcast by the server. If any data item $d_i \in Acq\_RS(T)$ was updated but not installed during the previous bcast, that is if a bit corresponding to d_i is 1 in the invalidation bit pattern, the client marks d_i as "invalid" and gets d_i again from the current bcast and puts it into local cache. Cache management in our scheme is therefore an invalidation combined with a form of autoprefetching [AFZ96b]. Invalidated data items remain in cache to be autoprefetched later. In particular, at the next appearance of the invalidated data item in the bcast, the client fetches its new value and replaces the old one.

3.3 Handling Disconnections

In case a client misses an invalidation bit pattern due to a disconnection during its acquisition phase, it merely invalidates all data items in its local cache after waking up from a disconnection. And, at the next appearance of the invalidated data items in the following bcast the client fetches its new value and replaces the old one. In this simple method, some data items in local cache may lead to "*false invalidation*", while in fact they are valid. That is, discarding the entire cache because of a disconnection may be costly in terms of energy-efficiency as most of benefits of caching are lost, especially if most of the cached data items are still valid [WYC96]. In fact, re-acquiring data items from scratch in point-to-point communication is costly since the disconnected client should send request messages, which consumes much energy than receiving ones, for invalidated data items. In the previous research [BI94, LHY99, PC99a], it has been shown that periodic retransmission of invalidation information can increase clients tolerance to intermittent connectivity. For instance, an invalidation bit pattern on data items updated during last ω (> 1) bcycles may be broadcast to allow a disconnected client to resynchronize. In this case, however, the client should wait for the next invalidation bit pattern to arrive in order to decide to retain or discard data items in its cache.

In the push-based data delivery, in contrast, the simple method using autoprefetching has the advantage of handling disconnections effectively by just fetching data items, which are on the air anyway. Furthermore, a read-only transaction can be shielded from disconnections if a client delivers data items to its transaction only after all data items belonging to $Pre\_RS(T_i)$ are acquired. For these reasons, we employ the simple method to handle disconnections in processing read-only transactions.

3.4 PwA method

Here we describe the read-only transactions processing method, called the PwA (Predeclaration with Autoprefetching) method. A client processes its read-only transaction T_i according to the following PwA method:

1. On receiving $Begin(T_i)$ {
   get $Pre\_RS(T_i)$; $Acq\_RS(T_i) = \emptyset$;
   }
2. For every "valid" item d_i in local cache {
   if ($d_i \in Pre\_RS(T_i)$) { $Acq\_RS(T_i) \Leftarrow d_i$; }
   }
   While ($Pre\_RS(T_i) \neq Acq\_RS(T_i)$) {
   for any ("invalid" item d_k in cache)
   or any ($d_j \in Pre\_RS(T_i) - Acq\_RS(T_i)$) {
   tune in and read d_k or d_j from the bcast;
   if (d_k was fetched) { overwrite the value; }
   if (d_j was fetched) {
   put d_j into local cache; $Acq\_RS(T_i) \Leftarrow d_j$;
   }
   }
   if (it is time to receive an invalidation bit pattern) {
   tune in and listen to an invalidation bit pattern;
   for every item d_i in local cache {
   if (the invalidation bit is set to 1) {
   mark d_i as "invalid";
   $Acq\_RS(T_i) = Acq\_RS(T_i) - \{d_i\}$; }
   }
   }
   if (miss an invalidation) {
   mark all data items in local cache "invalid";
   $Acq\_RS(T_i) = \emptyset$;
   }
   }
3. Deliver data items to T_i according to the order in which T_i requires, and then commit T_i.

The following **Theorem 1** shows the correctness of PwA method.

Theorem 1. *PwA method generates serializable execution of read-only transactions if the server broadcasts only serializable data values in each bcycle.*

Proof. Let $bcycle_i$ be the bcycle in which a transaction T_1 completes its acquisition phase and DS_i be the serializable database state that corresponds to the bcycle $bcycle_i$. We show that the values read by T_1 correspond to the database state DS_i by using a contradiction. Let us assume that the value of data item d_1 read by T_1 differs from the value of d_1 at DS_i. Then, an invalidation bit pattern should have been broadcast at the beginning of $bcycle_i$ and thus d_1 should have been invalidated. □

So far, we have described how PwA method works and its correctness. Now, we identify some useful properties of PwA method. In particular, PwA method has three highly desirable properties for transaction processing in wireless data broadcast:

1. It achieves a considerable reduction of transaction response time in an update-intensive environment. In particular, in the absence of disconnections, all data items needed for a read-only transaction can be fetched from *at most* two different bcasts even in an extreme case where all data items in a database is updated during a bcycle.

2. It commits each transaction successfully without sacrificing currency of data values.

3. It imposes minimal overhead on the server. The only overhead on the server side is to broadcast an invalidation bit pattern at the beginning of a bcycle.

4 Analytical Study

In this section, we analyze the performance of transaction processing methods in terms of expected average response time. The average response time will be measured in the number of data items. We compare PwA method with other methods proposed by [PC99b]. In particular, invalidation based methods only are considered for comparison purpose in no update environment, whereas multiversion based methods only are considered for comparison purpose in an updated database. This is due to the following reasons:

- Without updates at the server, invalidation based methods are superior to multiversion based ones which increase a bcycle length unnecessarily.

- With intensive updates at the server, invalidation based methods are intolerable to transaction aborts. Since PwA method never aborts read-only transactions, multiversion based methods which can commit transactions with high probability are more appropriate than invalidation based ones for comparison purpose.

In the uniform bcast, all D data items are broadcast periodically. In the nonuniform bcast, the D data items are split into n partitions, where each partition comprises data items with similar access frequencies. Partitions with larger access frequencies will be broadcast more often than those with lower access frequencies. Let partition i be broadcast λ_i times ($1 \leq i \leq n$). Moreover let $\lambda_i > \lambda_j$ for $0 < i < j$ and $\lambda_n = 1$. Let λ be LCM (least common multiple) of λ_i for all i. In [AAF+95], the ith partition, P_i ($1 \leq i \leq n$), is further split into c_i chunks ($c_i = \lambda/\lambda_i$). The data broadcast is then organized by a broadcast program that interleaves the chunks of the various partitions.

The broadcast program can also be viewed as a sequence of equal sized segments such that P_1 appears in all segments. Since P_1 is broadcast λ_1 times, there are

λ_1 segments and each segment contains $\sum_{i=1}^{n} \lambda_i |P_i|/\lambda_1$ items, where $|P_i|$ denotes the number of data items in partition i.

Let $|U|$ and $|NU|$ be the number of items in a single bcycle for the uniform bcast and the nonuniform bcast, respectively. In the uniform bcast, any data item appears only once in a bcycle. Thus,

$$|U| = D \tag{1}$$

For the nonuniform bcast, the number of data items in a bcycle is,

$$|NU| = \sum_{i=1}^{n} \lambda_i |P_i| \tag{2}$$

Obviously, $|NU| > |U|$. Furthermore, let a_s^U(or a_s^{NU}) and a_t^U (or a_t^{NU}) be the average response time for accessing a single data item and the average response time for accessing multiple data items in a given transaction for the uniform (or nonuniform) bcast, respectively.

4.1 Uniform Bcast

For the uniform bcast, the average response time for a single data item (the time elapsed from the moment a client requests for a data item to the point when the desired one is downloaded by the client) will, on average, be half the time between successive broadcasts of the data items,

$$a_s^U = \frac{1}{2}|U| \tag{3}$$

4.1.1 Case of No Updates

Let us first consider the case where there is no updates at the server so the execution of read-only transactions is always committed successfully. With the use of *invalidation* (InV) method proposed in [PC99b], a client retrieves data items in a one-at-a-time fashion, i.e. only after retrieving one item from a bcast another request is issued. Thus the average response time for a transaction accessing m data items can be computed as [3],

$$a_t^U(InV) = \frac{m}{2}|U| \tag{4}$$

Note that the average response time of InV method will be improved if local caching is employed. For the sake of a fair comparison, we assume that cache management is an invalidation combined with a form of autoprefetching. We call such a method IwA. Let h be the cache hit ratio of client caching. Then the average response time can be calculated by,

$$a_t^U(IwA) = \frac{m(1-h)}{2}|U| \tag{5}$$

In PwA method, a transaction processing is divided into 3 phases: preparation, acquisition, and delivery. If the time required by a client for each phase is expressed as PT, AT, and DT respectively, the response time can be formulated by,

$$a_t^U(PwA) = PT + AT + DT \tag{6}$$

Because of the serial nature of the bcast and the short span of preparation and delivery phases, most of transaction response time is dominated by acquisition phase. Thus Expression (6) can be reduced to,

$$a_t^U(PwA) \approx AT \tag{7}$$

Since a client can acquire all data items necessary for its transaction within a single bcylce with the use of PwA method, the average response time is bounded by $|U|$ irrespective of the number of data items or cache hit ratio, i.e.

$$a_t^U(PwA) \leq |U| \tag{8}$$

4.1.2 Case of Updates

The inherent drawbacks behind both InV and IwA methods are that they are prone to starvation of read-only transactions by frequent updates at the server and they perform poorly when the number of data items a transaction requires is increased. That is, the performance of InV and IwA methods is very sensitive to both update rate and the number of data items necessary for a transaction.

In order to increase the number of read-only transactions that are successfully processed, broadcasting multiple versions of data items is proposed in [PC99b]. *Multiversioning* (MV) method can effectively increase the number of read-only transactions that are successfully committed. To process every read-only transaction successfully by using multiversioning, however, the server should maintain enough large number of old versions per data item. Keeping multiple versions in the uniform bcast leads to the increased length of bcycle, which is proportional to the number of additional versions per data item, thereby resulting in the increased average response time.

More specifically, if the average number of updated data items during a single bcycle (i.e. $|U|$) is N_c and the server maintains large k old versions per data item enough to process all read-only transactions successfully, the increase for old versions on the bcast is at least kN_c. Thus, the average response time of a transaction is (note that, like the case with InV method, a client retrieves data items in a one-at-a-time fashion),

$$a_t^U(MV) = \frac{m}{2}(|U| + kN_c) \tag{9}$$

The average response time of MV method will be reduced if local caching is employed. For the sake of a fair comparison, we assume that cache management is an invalidation combined with a form of autoprefetching. We call such a method $MVwA$. Let h_c be the client cache hit ratio in an update environment. Then the average response time can be calculated by,

$$a_t^U(MVwA) = \frac{m(1-h_c)}{2}(|U| + kN_c) \tag{10}$$

However, using part of the cache space to keep old versions seems to result in a very small increase in

[3] The worst-case response time of InV method is $m|U| - m$, in the case where a transaction issues requests for items, whose next arrival time on the bcast is the highest, one at a time in the reverse order they are broadcast.

concurrency of long running transactions, since the effective cache size is decreased [PC99b].

In PwA method, a client listens to at most one invalidation bit pattern during its transaction processing. If the time required by a client for processing invalidation bit pattern is expressed as IBP, the response time of PwA can be formulated by,

$$a_t^U(PwA) = PT + AT + DT + IBP \quad (11)$$

One bit is assigned to each data item in an invalidation bit pattern, thus IBP is comprised of only D bits of which size corresponds to very few number of data items. Since PT, DT, and IBP are all trivial, Expression (11) can be reduced to Expression (7).

Since a client can acquire all data items necessary for its transaction within two bcylces with the use of PwA method, the average response time is bounded by $2|U|$ irrespective of the number of data items or cache hit ratio, i.e.

$$a_t^U(PwA) \leq 2|U| \quad (12)$$

4.2 Nonuniform Bcast

For the nonuniform bcast, the average response time for a single data item is optimal when the inter-arrival time between two consecutive occurrences of a data item is always the same, i.e. there is no variance in the inter-arrival time for each data item [VH99]. When the inter-arrival rate of a data item is fixed, the expected delay for a request arriving at a random time is one half of the gap between successive broadcasts of the data item. For each data item $d_i \in D$, thus, the expected delay of d_i is,

$$\omega(d_i) = \frac{|NU|}{2f_i} \quad (13)$$

, where f_i is the frequency of d_i. The expected average response time for any data request is calculated by multiplying the probability of access (denoted by $p(d_i)$) with the expected delay of each data item and summing the results,

$$a_s^{NU} = \sum_{d_i \in D} p(d_i)\omega(d_i). \quad (14)$$

4.2.1 Case of No Updates

With the use of InV method, assuming that there is no updates on the server side, the average response time for a transaction accessing m data items can be computed as,

$$a_t^{NU}(InV) = m \sum_{d_i \in D} p(d_i)\omega(d_i) \quad (15)$$

Also, the average response time of IwA method can by calculated by,

$$a_t^{NU}(IwV) = m(1-h) \sum_{d_i \in D} p(d_i)\omega(d_i) \quad (16)$$

If we apply the approximation in Expression (7), the average response time of PwA method is,

$$a_t^{NU}(PwA) \approx AT \quad (17)$$

Hence, the average response time for a transaction accessing m data items is bounded by $|NU|$,

$$a_t^{NU}(PwA) \leq |NU| \quad (18)$$

4.2.2 Case of Updates

In [PC99b], three approaches are proposed to maintain old versions of data items in the nonuniform bcast: clustering, overflow bucket pool, and new disks. With any approach to bcast organization, keeping multiple versions in the nonuniform bcast, as is the case with the uniform bcast, leads to the overall increased length of bcycle, which is proportional to the number of accommodated old versions per data item, thereby resulting in the increased average response time.

Although the inter-arrival time between two consecutive occurrences of a data item may be different on a bcast organization carrying old versions of data items, we assume that there is some optimal bcast organization in which the inter-arrival time of a data item is same. If the average number of data items that have updated during a single bcast (i.e. $|NU|$) is N_c and the server maintains large k old versions per data item enough to process all read-only transactions successfully, the increase for accommodating old versions on the bcast is at least kN_c. For each data item $d_i \in D$, hence, the expected delay of d_i,

$$\omega_c(d_i) = \frac{|NU| + kN_c}{2f_i} \quad (19)$$

, where f_i is the frequency of d_i on the bcast accommodating old versions of data items along with up-to-date data items. The expected average response time for any data request is calculated as,

$$a_s^{NU}(MV) = \sum_{d_i \in D} p(d_i)\omega_c(d_i) \quad (20)$$

Therefore, the average response time for a transaction accessing m data items can be computed as,

$$a_t^{NU}(MV) = m \sum_{d_i \in D} p(d_i)\omega_c(d_i) \quad (21)$$

Also, the average response time for $MVwA$ method can be calculated by,

$$a_t^{NU}(MVwA) = m(1-h_c) \sum_{d_i \in D} p(d_i)\omega_c(d_i) \quad (22)$$

As stated previously, however, using part of the cache space to keep old versions is not efficient because the effective cache size is decreased [PC99b]. In contrast, the average response time of a transaction in PwA method is bounded by $2|NU|$ if we apply Expression (17),

$$a_t^{NU}(PwA) \leq 2|NU| \quad (23)$$

5 Analytical Results

To further substantiate the analysis in last section, we show some analytical results in this section. In particular, the performance behavior of PwA methods will

be shown in terms of the *worst-case* expected response time, while the performance behavior of other methods will be shown in terms of the *average* expected response time. Table 1 and 2 summarize the parameter settings for the server and a client respectively, where values in parenthesis are the default ones. With respect to client's access frequency, the frequency of access of data items within a single partition is assumed to be uniformly distributed. It should be noted that the results described in this section are a small subset of the results that have been obtained. These results have been chosen because they demonstrate unique performance aspects of PwA method in wireless data broadcast.

Parameter	Value(s)
D	1000
n	3
λ_1, λ_2, λ_3	4, 2, 1
$\|P_1\|$, $\|P_2\|$, $\|P_3\|$	50, 150, 800

Table 1: Server Parameter Settings

Parameter	Value(s)
m	10-90 (20)
f_{P_1}, f_{P_2}, f_{P_3}	Varying (0.7, 0.2, 0.1)
Cache Replacement Policy	LRU
Cache Invalidation	Invalidation with Autoprefetch
h	10%-90% (90%)
h_c	10%-90% (90%)

Table 2: Client Parameter Settings

5.1 Asymptotic Analysis

First we analyze the performance behavior of several schemes in extreme cases. The analysis we want to present shows the behavior as the portion of updated data items during a bcycle tends to 0% ($N_c = 0$), and 100% ($N_c = D$).

5.1.1 Response Time Without Updates

In an environment where there is no updates at the server, $|U|$=1000 and $|NU|$=1300. Figure 2 shows the performance behavior of PwA and invalidation based methods as the number of data items for a transaction is increased (the y-axis is in logscale). In this graph, h is set to 90%. As shown in the figure, the worst-case response time of PwA method is totally immune to the number of data items, while the average response time of IwA method gets worse with increasing number of data items. Thus, if a transaction reads more than 20 or 50 items, PwA method provides an improvement over IwA method in a uniform or a nonuniform bcast respectively. When a transaction reads small or moderate number of data items, the worst-case response time of PwA method is higher than the average response time of IwA method in each bcast. In such a small- or moderate-scale case, however, the *average* response time of PwA is expected to show a similar shape to that of IwA method. This is because the procedure of PwA method is almost identical[4] to that of IwA method in a situation where a transaction can be processed within a single bcycle length with the use of IwA method.

[4]The only difference in retrieving data items is that additional reads may be necessary in PwA method.

With respect to the impact of different bcast organizations on transaction processing, in IwA method, the nonuniform bcast outperforms the uniform one by about 50% reduction of response time across all range of number of data items. In PwA method, however, the uniform bcast is superior to the nonuniform one with respect to the worst-cast response time. This is because the nonuniform bcast increases a bcycle length and a client has to spend more time to retrieve the less commonly requested data.

5.1.2 Response Time with Intolerable Updates

In the previous analysis, no data items is updated at the server. In this analysis, we examine the performance behavior of PwA and multiversion based methods in an "intolerable" environment where all data items in a database are updated during each bcycle. For this examination, we assume that $k = 4$ and $h_c = 90\%$. In the nonuniform bcast, in particular, the frequencies of access of different versions for a single data item are assumed to be uniformly distributed.

Figure 3 compares PwA and multiversion based methods in response time (y-axis is in logscale), where $MVwA$ method performs worse than PwA method across the entire range. The poor performance of $MVwA$ method is mainly due to the increased bcycle length; since a bcast accommodates 5 versions of data items where 4 old versions are appeared once within a bcycle, each bcycle in the uniform and the nonuniform bcasts is $5|U|$=5000 and $|NU| + 4(1000)$=5300 respectively. In contrast, in PwA method, each bcycle is $|U|$=1000 and $|NU|$=1300 respectively since only a single current version of data items appear on each bcast. In particular, PwA method has a bounded worst-case response time which is only doubled from a single bcycle length. Note that the worst-case response time is totally independent of client cache hit ratio. This means that PwA method works effectively especially in an intolerable update environment.

With respect to the impact of different bcast organizations on transaction processing, in $MVwA$ method, the uniform and nonuniform bcasts have almost the same average response times (the nonuniform bcast outperforms the uniform one by only 7% reduction of response time). In general, the similarity between two response times is more apparent for large k. In PwA method, however, the uniform bcast, which has a smaller bcycle length, is superior to the nonuniform one in terms of the worst-cast response time.

5.2 Some Examples

Here we show some additional comparison results for PwA and other methods in two environment: one is for no updates at the server, and the other is for an intensive update (half of the data items in a database are updated during a bcycle) environment.

5.2.1 Example 1: Response time with Varying Cache Hit Ratio

This example shows the sensitivity to cache hit ratio. Figures 4 and 5 present how the client performs in response to varying cache hit ratio in no update and intensive update environment respectively. As expected, performance of IwA and $MVwA$ methods suffer for each environment as cache hit ratio is decreased. As

we can see in Figure 4, PwA method behaves better than IwA method during the entire range of h in the uniform bcast. Only when h is greater than 70% in the nonuniform bcast, the average response time of IwA method is better than the worst-case response time of PwA method. As stated before, however, the *average* response time of PwA method is expected to be similar to that of IwA method in such a situation where a transaction can be processed within a single bcycle length with the use of IwA method.

Turning to the intensive update environment, PwA method outperforms $MVwA$ method for all range of cache hit ratio in the uniform bcast. Only if h_c is greater than 85% in the nonuniform bcast, the average response time of $MVwA$ method approaches close to the worst-case response time of PwA method. Furthermore, in reality, the hit ratio under $MVwA$ method will be a small value because of intensive updates and decreased effective cache size [PC99b]. This implies that PwA method has an improved performance over $MVwA$ method.

5.2.2 Example 2: Response time with Varying Access Pattern

This example shows the sensitivity to the disagreement between client access pattern and the server's broadcast program. In the nonuniform data broadcast, the server's broadcast may be sub-optimal for a particular client due to inaccurate access frequency of a client, dynamically changing access frequency of a client, and/or the server's averaged broadcast over the needs of a large client population [AAF+95]. To model such a mismatch between the needs of a client and the server's broadcast program, we use three access patterns, which are shown in Table 3, under the default data partitions. There, AP1 is the least matched access pattern, AP2 is less matched one, and AP3 is the most matched one.

Pattern	$(f_{P_1}, f_{P_2}, f_{P_3})$
AP1	(0.1, 0.2, 0.7)
AP2	(0.5, 0.3, 0.2)
AP3	(0.7, 0.2, 0.1)

Table 3: Different Client Access Patterns for Example 2

Figures 6 and 7 show the performance behavior of the mentioned methods in different access patterns. We can see that the average response times of IwA and $MVwA$ methods get worse as the mismatch becomes increasingly large from AP3 to AP1. Comparing these results with the results obtained in the previous example (see Figures 4 and 5), we see that the high degree of mismatch, i.e. when a client's access pattern is AP1, has the nonuniform bcast performance that is worse than the uniform bcast performance. This susceptibility to a broadcast mismatch is to be expected, as the client can not gain the benefits of the nonuniform bcast approach. In the case of PwA method, however, its worst-case response time is totally immune to the disagreement. If a transaction reads a moderate or large number of data items, PwA method dominates $MVwA$ and IwA methods for an entire range of access pattern. In no update environment, as stated early, PwA method is expected to have a comparable *average* response time to that of IwA method, although the latter dominates the former for a small or moderate number of data items (see Figure 6). Thus, we can conclude that PwA method is somewhat more tolerant to the degree of mismatch than other methods.

6 Conclusions

In this paper, we have proposed a simple but robust PwA (Predeclaration with Autoprefetching) method to speed up processing of wireless read-only transactions while keeping the serializability for the transactions in wireless data broadcast. Unlike other schemes, PwA method allows transactions to retrieve data items in the order they are broadcast rather than in the order the requests are issued. Wireless read-only transactions are therefore able to commit successfully with much reduction of response time. Through an analytical study, we have shown that PwA method is in favor of long transactions in a highly changing database environment. In particular, our method has a bounded worst-case performance behavior irrespective of the number of data items read by a transaction, client cache hit ratio, or client's data access pattern. Thus, it can adapt to dynamic changes in workload.

With respect to currency, which is another important performance metric, PwA method allows transactions to read current data values that correspond to the database state at the completion of data acquisition of its readset. Since the span of data delivery phase is usually short, however, most transactions read the values corresponding to the broadcast cycle at which they commit.

In the future, we intend to extend this work by considering multiple physical channels with which data items can be broadcast more efficiently than a single physical channel [LC00, PHO00, VH99].

References

[AAF+95] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast Disks: Data Management for Asymmetric Communication Environments. *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 199-210, 1995.

[AFZ96a] S. Acharya, M. Franklin, and S. Zdonic. Prefetching from a Broadcast Disk. *Proceedings of the 12th International Conference on Data Engineering*, pp. 276-285, 1996.

[AFZ96b] S. Acharya, M. Franklin, and S. Zdonik. Disseminating Updates on Broadcast Disks. *Proceedings of the 22nd International Conference on Very Large Data Bases*, pp. 354-365, 1996.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison Wesley, Reading, Massachusetts, 1987.

[BI94] D.Barbara, and T.Imielinski. Sleepers and Workaholics: Caching in Mobile Environments. *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 1-12, 1994.

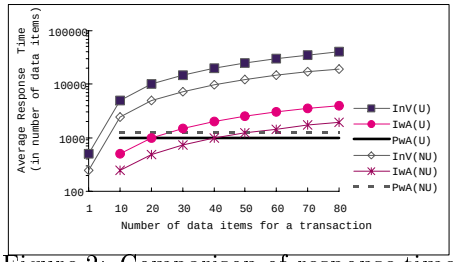

Figure 2: Comparison of response time in an extreme case

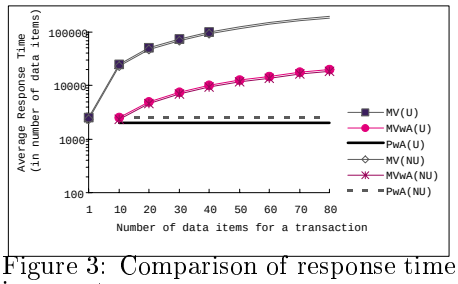

Figure 3: Comparison of response time in an extreme case

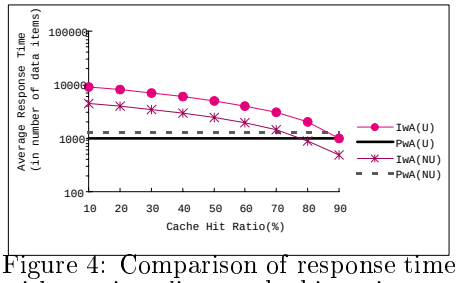

Figure 4: Comparison of response time with varying client cache hit ratio

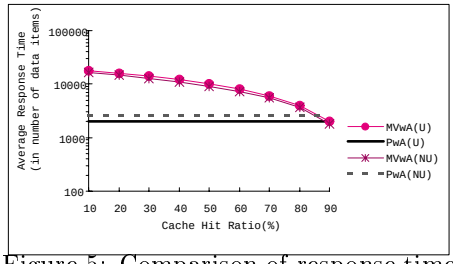

Figure 5: Comparison of response time with varying client cache hit ratio

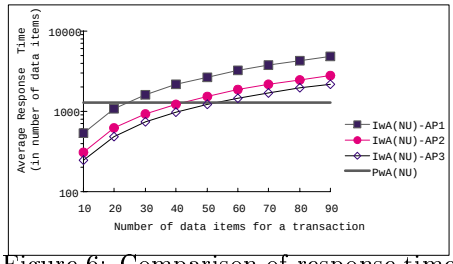

Figure 6: Comparison of response time with different access patterns

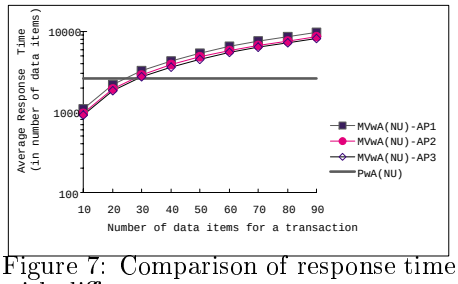

Figure 7: Comparison of response time with different access patterns

[JEH+97] J. Jing, A. Elmargarmid, S. Helal, and R. Alonso. Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments. *ACM/Baltzer Mobile Networks and Applications*, Vol. 2, No. 2, pp. 115-127, 1997.

[LAC99] K. Lam, M. Au, and E. Chan. Broadcast of Consistent Data to Read-Only Transactions from Mobile Clients. *Proceedings of the 2nd IEEE International Workshop on Mobile Computer Systems and Applications*, 1999.

[LC00] S. Lo and A. Chen Optimal Index and Data Allocation in Multiple Broadcast Channels. *Proceedings of the 16th IEEE International Conference on Data Engineering*, pp. 293-302, 2000.

[LHY99] S. Lee, and C. Hwang, and H. Yu. Supporting Transactional Cache Consistency in Mobile Database Systems. *Proceedings of the 1st ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pp. 6-13, 1999.

[PC99a] E. Pitoura and P. Chrysanthis. Scalable Processing of Read-only Transactions in Broadcast Push *Proceedings of the 19th International Conference on Distributed Computing Systems*, pp. 432-439, 1999.

[PC99b] E. Pitoura and P. Chrysanthis. Exploiting Versions for Handling Updates in Broadcast Disks. *Proceedings of the 25th International Conference on Very Large Data Bases*, pp. 114-125, 1999.

[PHO00] K. Prabhakara, K. Hua, and J. Oh. Multi-Level Multi-Channel Air Cache Designs for Broadcasting in a Moible Environment. *Proceedings of the 16th IEEE International Conference on Data Engineering*, pp. 167-176, 2000.

[SNS+99] J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran, and K. Ramamritham. Efficient Concurrency Control for Broadcast Environments. *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 85-96, 1999.

[TY98] Kian-Lee Tan and Jeffrey Xu Yu. Generating Broadcast Programs that Support Range Queries. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, No. 4, pp. 668-672, 1998.

[VH99] N. H. Vaidya and S. Hameed. Scheduling Data Broadcast in Asymmetric Communication Environments. *Wireless Networks*, Vol. 5, No. 3, pp 171-182, 1999.

[WYC96] Kun-Lung Wu, Philip S. Yu and Ming-Syan Chen. Energy-Efficient Caching for Wireless Mobile Computing. *Proceedings of the 12th International Conference on Data Engineering*, pp. 336-343, 1996.