データベースシステム　122－4
(2000. 7. 26)

無線データ放送における読取り専用トランザクションの支援方法

李尚根 喜連川優
東京大学生産技術研究所
E-mail:{lsk,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract

無線のデータ放送はユーザーがそれらの多量数から同時に独立したデータを検索することを可能にするが、データは、ユーザーによって厳格にシークエンシャルにアクセスされなければならない。この論文では、無線のデータ放送における読取り専用のトランザクションによって要求されたデータアイテムの一貫性を保つことの問題について述べている。Predeclaration と autoprefetching の結合せがデータ放送におけるトランザクション処理のために非常に有利であるということが論証される。特に、我々の研究は、読取り専用のトランザクションが放送周波の長さを増幅させたりサーバーの更新の割合の影響を受けることなく首尾よく処理することを可能にする。我々は、読取り専用のトランザクション処理への方法として PwA (Predeclaration with Autoprefetching) について述べる。そしてまた、分析的研究により我々の手法の性能を評価する。

Supporting Read-only Transactions in Wireless Data Broadcast

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

Abstract

While wireless data broadcast allows users to retrieve data simultaneously independent of the number of them, data can only be accessed strictly sequential by users. This paper addresses the issue of maintaining consistency of data items requested by mobile read-only transactions in wireless data broadcast. It is demonstrated that the combination of predeclaration with autoprefetching is highly advantageous for transaction processing in data broadcast. In particular, our approach allows read-only transactions to process successfully without increasing the broadcast cycle length or being considerably affected by the rate of updates at the server. We describe *PwA* (Predeclaration with Autoprefetching) method for read-only transactions processing, and also evaluate the performance of our method by an analytical study.

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 the 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 mobile clients where bandwidth efficiency is a major concern. An important consideration in data broadcast is to provide consistent data values to mobile 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. However, if updates at the server are done concurrently, the transactions may observe inconsistent data values. This paper addresses such a consistency problem in wireless data broadcast.

Recently, several approaches to consistent data access despite updates in wireless data broadcast have been proposed in the literature [LAC99, PC99a, PC99b, SNS+99]. Update-first with order (UFO) algorithm proposed in [LAC99] checks data conflicts among broadcast transactions and update transactions instead of detecting conflicts among mobile transactions and update transactions. By re-broadcasting conflicting data items, this algorithm ensures that the serialization order of a broadcast transaction is preceded by an conflicting update transaction. Since mobile transactions read data items from broadcast transactions, their serialization orders are always preceded by broadcast transactions. However, this method is vulnerable to performance degradation in the case of heavy updates.

A control matrix is used for concurrency checking in [SNS+99], and serialization graph testing method is proposed in [PC99a]. The major problem with these approaches is that the large overhead is involved in maintaining control information for concurrency control and conflict detection. Furthermore, maintaining control matrix or serialization graph involves complicated processing at the server. A simple invalidation-only method is also presented in [PC99a]. In this method, however, read-only transactions processing is significantly affected by the rate of updates at the server. To increase the number of read-only transactions that are successfully processed despite updates at the server, multiversion schemes are employed in [PC99b]. However, multiversion schemes increase the broadcast cycle length, so they impose a serious performance problem.

In this paper, we investigate a read-only transaction processing scheme that facilitates predeclaration combined with local caching. In a traditional pull-based 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 very useful 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 (i.e., serializable) data values in each broadcast cycle [1], 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 way, read-only transactions are processed successfully without increasing broadcast cycle length or being considerably affected by the rate of updates at the server. The strength of our method is based on the assumption that a mobile client is equipped with enough local storage capacity to hold all data items needed to execute its 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.

The remainder of this paper is organized as follows. Section 2 introduces basic preliminaries. Sections 3 and 4 describe our PwA method for processing mobile read-only transactions and an analytical study, respectively. We conclude in Section 5.

2 Preliminaries

The server periodically broadcasts data items 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 mobile client is a nonvolatile memory such as a hard disk. At any given time, it is assumed that

[1] This kind of assumption was also made in [PC99a, PC99b].

there exists a single read-only transaction in a mobile client. Clients are assumed to have only predictable and willing disconnections (e.g. go into a doze mode to conserve batter power). 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 (our work is also applicable to the case where some form of directory information is broadcast along with data items without loss of generality).

2.1 Predeclaration and Its Usefulness

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 broadcast, while all data items are broadcast, some will appear more often than those that are less frequently broadcast. For example, two different bcast organizations are 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 bcast 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). Program (a) is a uniform broadcast program, while (b) is a nonuniform broadcast program.

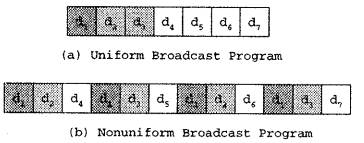

Figure 1: Bcast Organization using Uniform vs. Nonuniform Broadcast Programs

To first show that the order in which a transaction reads data affects the response time of the transaction, consider a client transaction program starting its execution at the beginning of the bcycle in Figure 1.(a):

IF ($d_6 \leq 3$) THEN read(d_1) ELSE read(d_2).

Since both d_1 and d_2 precede d_6 in the bcast and access to data is strictly sequential, the transaction has to read d_6 first and wait for the next cycle to read the value of d_1 or d_2. Waiting for the next cycle is also necessary in the case of Figure 1.(b) if the transaction has to read d_2 after reading d_6. If, however, all data items that will be accessed by a transaction are predeclared in advance, a mobile client can hold all necessary data items within a single bcycle.

Another point, however, is not as obvious 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 mobile client will have time to get ready to read before the data its transaction needs reaches itself.

2.2 Transactional Consistency and Transaction Length

The server broadcasts the content of a database. A database consists of a finite set of data items. We assume that the server broadcasts only transactionally consistent data values in each bcycle, and *serializability* [BHG87] is adopted as a correctness criterion.

While data items are being broadcast, update transactions are executed at the server that update the values of the data items broadcast. In particular, we assume that the values of data items that are broadcast during each bcycle correspond to the state of the database at the beginning of the cycle. Thus, a read-only transaction that reads all its data 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 in which the readset of a transaction is predeclared in advance.

3 Supporting Read-only Transactions

In this section, we show that the serializability of mobile read-only transactions can be maintained in the environment where data 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. 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. Each mobile 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 following section, handling such a situation is addressed.

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. In particular, 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) and a bit is set to 1 if its corresponding data item has been updated at the server during the previous bcycle. Bits are set to 0s if their corresponding data items have not been updated at the server during the previous bcycle. 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 bcast, 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, 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.

During its delivery phase, however, the timing when the client fetches a new value for an invalidated item again should be slightly modified. This is because a transaction may not read consistent data if the client replaces an item belonging to $Acq\_RS(T)$ with a new value during delivery phase. One way of avoiding this undesirable situation is that the client does not replace an invalid item belonging to $Acq\_RS(T)$ with a new value until its current transaction is completed. Since all data items necessary for the transaction are already in local cache, however, the delayed period is negligible.

3.3 *PwA* method for Processing Read-only Transactions

In this section, we describe *PwA* (Predeclaration with Autoprefetching) method. A mobile client processes its read-only transaction T_i according to the following *PwA* method:

1. *Preparation phase*
   (a) The client gets $Pre\_RS(T_i)$ and assigns an empty set to $Acq\_RS(T_i)$.
   (b) The client executes Step 2.
2. *Acquisition phase*
   (a) For each $d_i \in Pre\_RS(T_i)$, if it resides in local cache and is not marked as "invalid", the client puts d_i into $Acq\_RS(T_i)$.
   (b) The client tunes in and reads each $d_j \in Pre\_RS(T_i) - Acq\_RS(T_i)$ from the bcast(s) according to the order in which they appear on the air. Whenever each d_j is read, it is put into local cache and added to $Acq\_RS(T_i)$.
   (c) During Step 2-(a) or 2-(b), the client listens to an invalidation bit pattern, if any, broadcast by the server. If $d_i \in Acq\_RS(T_i)$ is invalidated, the client excludes d_i from $Acq\_RS(T_i)$. Data item d_i is added to $Acq\_RS(T_i)$ again when it is autoprefetched from the bcast.
   (d) If $Acq\_RS(T_i)$ is equal to $Pre\_RS(T_i)$, then the client executes Step 3.
3. *Delivery phase*
   (a) The client delivers data items to T_i according to the order in which T_i requires them.
   (b) During Step 3-(a), the client listens to an invalidation bit pattern, if any, broadcast by the server. If $d_i \in Acq\_RS(T_i)$ is invalidated, the client marks d_i as "invalid" in local cache. However, the client does *not* exclude d_i from $Acq\_RS(T_i)$. Also, the client does *not* fetch d_i from the bcast again until Step 3-(c) is completed.

(c) If a commit operation is issued from T_i, the client commits it successfully.

Obviously, PwA method never aborts read-only transactions without resorting to multiversion schemes which increase the broadcast cycle length considerably. Furthermore, PwA method is at minimum vulnerable to the rate of updates at the server. In particular, in the absence of unwilling disconnections, all data items needed for a read-only transaction can be fetched from *at most* two different bcasts. This leads to a considerable reduction of transaction response time compared to other schemes.

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 during which a transaction T_1 is committed and DS_i be the serializable database state that corresponds to the bcycle $bcycle_i$, i.e., the database state at the beginning of $bcycle_i$. 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. □

4 Analytical Study

In this section, we analyze the performance of transaction processing methods in terms of average response time. The average response time is measured in the number of data items. In particular, we study both PwA and previous methods in both a uniform and a nonuniform bcast environments.

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 LCD(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 $\sum_{i=1}^{n} \lambda_i |P_i|/\lambda_1$ objects, where $|P_i|$ denotes the number of data items in partition i.

Let $|U|$ and $|NU|$ be the number of objects 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, $|U|$ is the number of data items in the database. For the nonuniform bcast, the number of data items in a bcycle is $|NU| = \sum_{i=1}^{n} \lambda_i |P_i|$, where $|P_i|$ is the number of data items in partition i. 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, i.e. $a_s^U = \frac{1}{2}|U|$.

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], the average response time for a transaction accessing m data items can be computed as (the client retrieves data items in a one-at-a-time fashion),

$$a_t^U(InV) = \frac{m}{2}|U| \tag{1}$$

Note that the average response time of InV method will be reduced to some extent if local caching is employed. For comparison purpose, 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,

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

With the use of PwA method, the average response time is dominated by acquisition phase since both preparation and delivery phases are short enough to neglect, i.e.

$$a_t^U(PwA) \leq |U| \tag{3}$$

The inherent drawbacks behind both InV and IwA methods are that they are prone to starvation of read-only transactions by 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.

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,

$$a_t^U(MV) = \frac{m}{2}(|U| + kN_c) \quad (4)$$

Note that the average response time of MV method will be reduced to some extent if local caching is employed. For comparison purpose, we assume that cache management is an invalidation combined with a form of autoprefetching. We call such a method $MVwA$. Let h be the cache hit ratio of client caching. Then the average response time can be calculated by,

$$a_t^U(MVwA) = \frac{m(1-h)}{2}(|U| + kN_c) \quad (5)$$

However, using part of the cache space to keep old versions seems to result in a very small increase in concurrency of long running transactions, since the effective cache size is decreased [PC99b].

In contrast to these multiversion approaches, PwA approach is more immune to updates at the server and totally immune to the number of data items a transaction accesses at the cost of space complexity on the client side. In particular, the average response time of a transaction in PwA method is bounded by $2|U|$ without respect to the cache hit ratio, i.e.

$$a_t^U(PwA) \leq 2|U| \quad (6)$$

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, $\omega(d_i)$, is $\frac{|NU|}{2f_i}$, 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, i.e. $a_s^{NU} = \sum_{d_i \in D} p(d_i)\omega(d_i)$. 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 (the client retrieves data items in a one-at-a-time fashion),

$$a_t^{NU}(InV) = m \sum_{d_i \in D} p(d_i)\omega(d_i) \quad (7)$$

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

$$a_t^{NU}(IwV) = m(1-h) \sum_{d_i \in D} p(d_i)\omega(d_i) \quad (8)$$

With the use of PwA method, the average response time is dominated by acquisition phase, i.e.

$$a_t^{NU}(PwA) \leq |NU| \quad (9)$$

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 $d_i \in D$, thus, the expected delay of d_i, $\omega_c(d_i)$, is $\frac{|NU|+kN_c}{2f_i}$, 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 $a_s^{NU}(MV) = \sum_{d_i \in D} p(d_i)\omega_c(d_i)$, and the average response time for a transaction accessing m data items can be computed as,

$$a_t^{NU}(MV) = m \sum_{d_i \in D} p(d_i)\omega_c(d_i) \quad (10)$$

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

$$a_t^U(MVwA) = m(1-h) \sum_{d_i \in D} p(d_i)\omega_c(d_i) \quad (11)$$

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 $2|NU|$ without respect to the cache hit ratio, i.e.

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

4.3 Some Analytical Results

To further substantiate the previous analysis, we show some analytical results in this section. In particular, we compare the performance behaviors of several schemes for two extreme cases: one is for no updates at the server, and the other is for an update-intensive (half of the data items in a database are updated during a bcycle) environment. Since PwA method never aborts transactions, only invalidation based methods are considered for comparison purpose in no updates

environment, whereas only multiversion based methods are considered in an update-intensive one.

In our system model, the server broadcasts 1000 data items according to a broadcast program. Table 1 shows server parameter setting for a nonuniform broadcast program, and Table 2 shows client parameter setting, where the frequency of access of data items within a single partition is assumed to be uniformly distributed.

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 Setting for a Nonuniform Broadcast Program

Parameter	Value(s)
m	Varying (10-80)
f_{P_1}, f_{P_2}, f_{P_3}	0.7, 0.2, 0.1
Cache Replacement Policy	LRU
h	90 %

Table 2: Client Parameter Setting

Experiment 1: Response Time without Updates. Since there is no updates at the server, $|U|$=1000 and $|NU|$=1300. Figure 2 compares the performance behaviors of PwA and invalidation based methods (the y-axis is in logscale), where the performance behavior lines of PwA methods correspond to the *worst-case* response time and the performance behavior curves of invalidation based methods correspond to the *average* response time. As shown in Figure 2, the worst-case response time of PwA method, i.e. a single bcycle length, is longer than the average response time of IwA method only when a transaction reads small or moderate number of data items in both a uniform and a nonuniform bcasts. Even in such a small- or moderate-scale case, however, the *average* response time of PwA will show a very similar performance behavior shape to that of IwA method. This is because the procedure of PwA method is almost identical 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. Another point is that, in invalidation based methods, a nonuniform bcast outperforms a uniform one by about 50% reduction of response time. In PwA method, however, a uniform bcast is always superior to a nonuniform one with respect to the worst-case response time. This is mainly due to the inherent property of PwA method that a client can more quickly acquire all data items from a bcast with a smaller bcycle length.

Experiment 2: Response Time with Intensive Updates. In this case, we assume that $k = 2$ and N_c=500. Thus, if a bcast accommodating multiple versions of data items is organized so that old versions of data items are appeared once within a bcycle, the bcycle lengths in the case of multiversion based methods are 2000 (uniform bcast) and 2300 (nonuniform bcast), whereas $|U|$=1000 and $|NU|$=1300 in PwA method. In a nonuniform bcast, in particular, half of the data items in each partition are assumed to be updated during a bcycle and the frequencies of access of different versions for a single data item are uniformly distributed. Figure 3 compares PwA and multiversion based methods in response time (y-axis is in logscale). As expected, multiversion based methods perform poorly as the number of data items is increased. In contrast, PwA method has a bounded worst-case performance behavior. In particular, the worst-case response time of PwA method is only doubled from a single bcycle length. PwA method in worst-case performs worse than $MVwA$ method in average case only when a transaction reads small number of data items in both a uniform and a nonuniform bcasts. Even such a small-scale case, however, the *average* response time of PwA will show a performance behavior comparable to that of $MVwA$ method. This is because, although the cache hit ratio for $MVwA$ method is assumed to be 90% in this experiment, in reality, the ratio results in a much smaller value because of intensive updates and decreased effective cache size. With respect to different bcast organizations, in multiversion based methods, a nonuniform bcast outperforms a uniform one by no more than 10% reduction of response time. In PwA method, however, a uniform bcast is always superior to a nonuniform one in terms of the worst-case response time. This is because a uniform bcast has a smaller bcycle length than a nonuniform one.

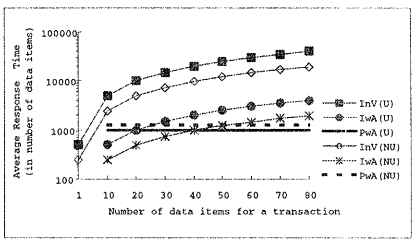

Figure 2: Comparison of Response Time (No Updates)

5 Conclusion

In this paper, we have proposed a simple but robust PwA (Predeclaration with Autoprefetching) method for processing read-only transactions in wireless data broadcast. Unlike other schemes, PwA method allows transactions to commit successfully without increasing the broadcast cycle length or being considerably affected by the rate of updates at the server. An analytical study conducted in this paper has demonstrated that

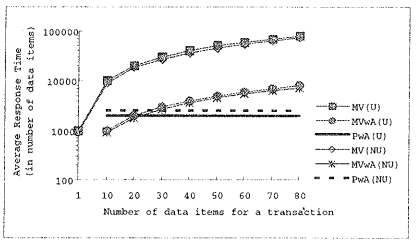

Figure 3: Comparison of Response Time (Intensive Updates)

the use of PwA method is highly effective for transaction management especially in an intensive-update environment. Although this work has assumed that the information about the readset of a transaction is available at the beginning of transaction processing, we believe that our idea can be applied to some real database applications by using preprocessor, such as a compiler, on a client to analyze its transaction before being submitted to the client system.

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.

[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.

[PC99a] E. Pitoura and P. Chrysanthis. Scalable Processing of Read-Only Transactions in Broadcast Push. *Proceedings of the 19th IEEE 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.

[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.

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