Efficient Processing of Wireless Read-only Transactions in Data Broadcast

SangKeun Lee
Next Generation Handsets Lab.
LG Electronics Inc.
yalphy@lge.com

Masaru Kitsuregawa
Institute of Industrial Science
The University of Tokyo
kitsure@tkl.iis.u-tokyo.ac.jp

Chong-Sun Hwang
Dept. of Computer Science and Engineering
Korea University
hwang@disys.korea.ac.kr

Abstract

In this paper, we address the issue of ensuring consistency of multiple data items requested in a certain order by read-only transactions in wireless data broadcast. To handle the inherent property in a data broadcast environment that data can only be accessed strictly sequential by users, we explore a predeclaration-based query optimization and devise two practical transaction processing methods in the context of local caching. We also evaluate the performance of the proposed methods 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 a wireless broadcast environment, mechanisms to efficiently transmit information to mobile clients are of significant interest. For instance, such mechanisms could be used by a satellite or a base station to communicate information of common interest to mobile clients. Broadcast-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 trading and sport tickets, electronic newsletters, mailing lists, banking and traffic management systems. In such applications, if there is a 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 to a large number of clients where bandwidth efficiency is a major concern.

Providing consistent data values to transactions is one of main issues in designing mechanisms for wireless data broadcast [3, 7, 11]. In wireless 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. In the previous work, some weak consistency criterion has often been adopted (e.g. *update consistency* in [10]) to properly handle consistency problem caused by updates at the server. This kind of approaches is based on the belief that serializability would be ”expensive” to achieve for asymmetric communication environments. This paper, however, has still stuck to serializability for the following reasons:

- We observed that serializability is *not* expensive to achieve in the proposed concurrency control techniques. This is in contrast to the argument that any potential protocol for ensuring serializability would be very expensive in broadcast environments, thus leading to poor performance [10]. Performance evaluation presented in Section 5 supports our argument.
- It is natural to consider serializability ”expensive”, if the client consistency requirement is not so strict *and* weak consistency requirement is proven to be much more efficient to achieve than serializability. However, it is not easy to meet the latter condition without sacrificing currency requirement. This is important since we believe most advanced applications in a dissemination-based environment do need to read current data items as possible as they can.

- While the protocols based on the relaxed consistency requirements are useful in some applications, serializability may still be necessary to guarantee the correctness of some other applications such as mobile stock trading where a buy/sell trade will be triggered to exploit the temporary pricing relationships among stocks. From the trader's perspective, the inability of maintaining serializability may lead to important financial consequences. For instance, if the users who submitted multiple read-only transactions to communicate and compare their query results, they may be confused [6].

The major challenge in this paper is how to provide consistent data items to wireless transactions while speeding up their processing. For this end, a predeclaration-based query optimization is explored in conjunction with local caching technique. In a traditional pull-based (i.e. client-initiated) data delivery, predeclaration technique has often been used to avoid deadlocks in locking protocols [5]. 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.

The unique contributions of our work are two-fold. First, to the best of our knowledge, our work is the first approach to query optimization for reducing transaction response time significantly in wireless data broadcast. This is based on the philosophy that a client should take a more active role in maintaining its transactions consistency in an asymmetric communication environment. Second, our work is able to balance between average response time (i.e. *overall* system performance) and worst-case response time (i.e. *individual* performance), which is one important challenge in wireless transaction management [11]. In the general case, the performance gap between average and worst-case response time in our approach is only half of a broadcast cycle.

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 two predeclaration-based transaction processing methods to maintain consistency of transactions. Section 4 and 5 contain an analysis and its results respectively. Our conclusion is finally presented in Section 6.

2. Basic Design Principles

With the system model similar to [8], let us consider the two different broadcast organizations illustrated in Figure 1, where the server broadcasts a set of data items $\{d_1, d_2, d_3, d_4, d_5, d_6, d_7\}$ in one broadcast according to a broadcast program (d_1 is the relatively 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 the uniform broadcast program, (b) is the nonuniform broadcast program (refer to [1] for detail).

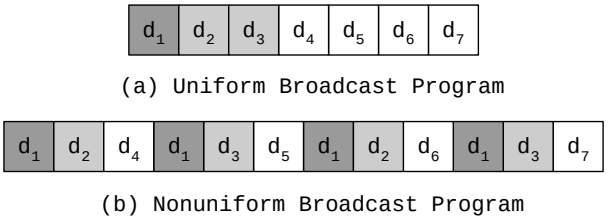


Figure 1. Broadcast Organization

Suppose that, in each broadcast organization, a client transaction program starts its execution at the middle of the broadcast cycle:

IF ($d_3 \leq 3$) THEN read(d_1) ELSE read(d_2)

Now, we can pose the following two questions, including

- What is the response time (in terms of number of data items) taken to execute the transaction?
- What happens to transactional consistency if data items are being updated at the server?

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

2.1. 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 broadcast in Figure 1. Since both d_1 and d_2 precede d_3 in the broadcast content with respect to the client and access to data is strictly sequential, the transaction has to read d_3 first and wait to read the value of d_1 or d_2. Thus, the response time of the transaction is 11.5 (in case d_3 and d_1 are accessed) or 12.5 (in case d_3 and d_2 are accessed). If, however, all data items that will be accessed *potentially* by the transaction, i.e., $\{d_1, d_2, d_3\}$, are predeclared in advance, a client can hold all necessary data items with a reduced response time of 6.5. This is also true to the case of the nonuniform broadcast content in Figure 1. The response time of the transaction is 7 (in case d_3 and d_1 are accessed) or 8 (in case d_3 and d_2 are accessed), while the response time is 5 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.

2.2. Transactional Consistency in presence of Updates

One way to ensure the consistency of read-only transactions is to abort transactions that read data values that correspond to different database states [9, 8]. 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 they read consistent data items only. To achieve this, the server can broadcast multiple versions of data items so that clients could read appropriate versions [8]. However, this kind of multiversion schemes increases a broadcast cycle length, thereby resulting in the increased transaction response time.

A better way, which is adopted in this paper, is to make sure that each broadcast cycle represents a consistent snapshot of the database (this requirement was also assumed in the work [9, 8]). Thus, a read-only transaction that reads all its items within a single broadcast cycle can be successfully executed without any concurrency control overhead at all. In reality, however, most transactions will be started at some point within a broadcast cycle, thus may have to read data items from different broadcast contents. In such a situation, there is no guarantee that the values they read are consistent. This is also true to the case of the predeclaration-based data access scheme. To cope with this, we require that data acquisition process be separated from actual data reads by transactions, so only a subset of consistent snapshot of the database can be acquired from a single broadcast content (in a synchronous manner) or two broadcast contents (in an asynchronous manner).

3. Proposed Methods

Now, two predeclaration-based transaction processing methods are devised in the context of local caching technique: PA (Predeclaration with Autoprefetching) and PA^2 (PA/Asynchronous). The central idea is to employ predeclaration of readset in order to minimize the number of different broadcast cycles from which transactions read data. The information about the readset of a transaction is assumed to be available at the beginning of transaction processing by using preprocessor on a client to analyze its transaction before being submitted to the client system (this will consume certain processing power on the client side).

3.1. Caching and Invalidation Bit Pattern

Clients can cache data items of interest locally to reduce access latency. Caching reduces the latency of transactions since transactions find data of interest in their local cache and thus need to access the broadcast channel for a smaller number of times. In this paper, clients use their available hard disks as local caches and caching technique is employed in the context of transaction processing. We therefore need to guarantee that transaction semantics should not be violated as a result of the creation and destruction of cached data based on the runtime demands of clients.

In the presence of updates on the server, items in cache may become stale. In wireless data broadcast, clients access data from their local caches, while updates to data values are collected at a server site. In order to keep the clients' caches consistent with the updated data values, the client-cached copies of modified items must be invalidated or updated. Among various approaches to communicating updates to the clients, it has been shown in the work [2, 4] that the client cache coherency can be effectively maintained exploiting a periodic *invalidation report* which is a list of the items that have been updated recently. Broadcasting identifiers of updated items, however, may consume much portion of broadcast channel, which is a scarce resource, especially if a large portion of items in the database is updated. Furthermore, in the context of serializability consistency model, consistency must be preserved across reads of multiple data items.

For practical transaction processing methods in conjunction with local caching, in our work, the server is required to broadcast an *invalidation bit pattern* which is followed by a broadcast content. In an invalidation bit pattern, each bit corresponds to a single data item in the database (we assume 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 broadcast cycle but not installed into the previous broadcast content. The remaining bits are set to 0s. This way, compared to invalidation reports, the size of invalidation information broadcast by the server can be significantly reduced, especially if a large portion of items in the database is updated. Moreover, transactional cache consistency can be easily maintained if a serializable broadcast content is on the air in each broadcast cycle. Thus, we presume that the following always hold:

Server Requirement: In each broadcast cycle, the server broadcasts an invalidation bit pattern which is followed by serializable data values written by committed transactions.

At the beginning of each broadcast cycle, a client tunes in and reads the invalidation bit pattern broadcast by the server. For any data item d_i in its local cache, 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 broadcast content and puts it into local cache. Cache management in our scheme is therefore an invalidation combined with a form of autoprefetching [2].

Invalidated data items remain in cache to be autoprefetched later. In particular, at the next appearance of the invalidated data item in the broadcast content, the client fetches its new value and replaces the old one.

3.2. Methods PA and PA^2

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*. For all methods, each client processes 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 the broadcast content(s) or its local cache. During this phase, a client additionally maintains a set $Acquire(T)$ of all data items that it has acquired so far, and (3)*Delivery phase:* it delivers data items to its transaction according to the order in which the transaction requires data.

With the server requirement shown in Section 3.1, the execution of each read-only transaction is clearly serializable if a client can fetch all data items within a single broadcast cycle. Since, however, a transaction is expected to start at some point within a broadcast cycle, its acquisition phase may therefore be across more than one broadcast cycle. To remedy this problem, in method PA, a client starts the acquisition phase synchronously, i.e. at the beginning of the next broadcast cycle. Since all data items for its transaction are already notified, the client will complete the acquisition phase within a single broadcast cycle. More specifically, a client processes its transaction T_i as follows:

1. On receiving $Begin(T_i)$ {
   get $Pre\_RS(T_i)$ by using preprocessor;
   $Acquire(T_i) = \emptyset$;
   wait for the next broadcast cycle to begin;
   }
2. Tune in and listen to an invalidation bit pattern;
   For every item d_i in local cache {
   if (a corresponding invalidation bit is set to 1)
   { mark d_i as "invalid"; }
   }
   For every "valid" item d_i in local cache {
   if ($d_i \in Pre\_RS(T_i)$) { $Acquire(T_i) \Leftarrow d_i$; }
   }
   While ($Pre\_RS(T_i) \neq Acquire(T_i)$) {
   for any ("invalid" item d_k in local cache)
   or any ($d_j \in Pre\_RS(T_i) - Acquire(T_i)$) {
   tune in and read d_k or d_j when available from the broadcast content;
   if (d_k was fetched)
   { overwrite the value in local cache; }
   if (d_j was fetched)
   { put d_j into local cache; $Acquire(T_i) \Leftarrow d_j$; }
   }
   }
3. Deliver data items to T_i according to the order in which T_i requires, and then commit T_i.

Theorem 1. *PA generates serializable execution of read-only transactions if, in each broadcast cycle, the server broadcasts an invalidation bit pattern which is followed by serializable data values.*

Proof. It is straightforward from the fact that the data set read by each transaction is a subset of a single broadcast.

The main advantage of PA is that it achieves a considerable reduction of transaction response time in an update-intensive environment without sacrificing serializability. In particular, each transaction can be successfully committed within two broadcast cycles even in an extreme case where all data items in a database are updated during a broadcast cycle. The disadvantage of PA, however, is that its synchronous approach may incur unnecessary response time latency to short transactions in a sporadically updated database. For example, if most of data items reside in local cache and all missed items can be retrieved from the current broadcast content, then a transaction would be completed within a single broadcast cycle in which it is initiated.

To get over the disadvantage of method PA, a client can take an asynchronous way, i.e. it fetches data items immediately without waiting for the next broadcast cycle to begin. Notice that, unlike a synchronous approach, the acquisition phase may be across two different broadcast contents in this case. This method is referred to as PA^2. It goes as follows:

1. On receiving $Begin(T_i)$ {
   get $Pre\_RS(T_i)$ by using preprocessor;
   $Acquire(T_i) = \emptyset$;
   }
2. For every "valid" item d_i in local cache {
   if ($d_i \in Pre\_RS(T_i)$) { $Acquire(T_i) \Leftarrow d_i$; }
   }
   While ($Pre\_RS(T_i) \neq Acquire(T_i)$) {
   for any ("invalid" item d_k in local cache)
   or any ($d_j \in Pre\_RS(T_i) - Acquire(T_i)$) {
   tune in and read d_k or d_j when available from the broadcast content;
   if (d_k was fetched)
   { overwrite the value in local cache; }
   if (d_j was fetched)
   { put d_j into local cache; $Acquire(T_i) \Leftarrow d_j$; }
   }
   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 (a corresponding invalidation bit is set to 1)

{ mark d_i as "invalid";
$Acquire(T_i) = Acquire(T_i) - \{d_i\}$;
}
}
}
}
3. Deliver data items to T_i according to the order in which T_i requires, and then commit T_i.

Theorem 2. *PA^2 generates serializable execution of read-only transactions if, in each broadcast cycle, the server broadcasts an invalidation bit pattern which is followed by serializable data values.*

Proof. Let $bcycle_i$ be the broadcast cycle in which some transaction T_1 completes its acquisition phase and DS_i be the serializable database state that corresponds to the broadcast cycle $bcycle_i$. 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.

Note that both PA and PA^2 impose minimal overhead on the server. The only overhead on the server side is to broadcast both serializable data values and an invalidation bit pattern at each broadcast cycle.

4. Analysis

In this section, we develop analytical models to compare predeclaration-based transaction processing methods with other two methods which are slightly modified versions from ones proposed by [8]. We below describe the procedure sketch of the two.

Invalidation with Autoprefetching (IA) : This method invalidates (i.e. aborts) any transaction that reads data values that correspond to different database states in order to ensure the serializability of transactions. To achieve this, the server broadcasts an invalidation bit pattern, which has been described in Section 3.1. Each client caches items of interest locally with each item as a unit of caching. The invalidation with a form of autoprefetching is employed as cache management policy. In addition, each client maintains a set $RS(T)$ for each active transaction T, which includes all data items T has read so far. The client tunes in at the beginning of each broadcast to read the invalidation bit pattern. A transaction T is aborted and restarted if any item $d_i \in RS(T)$ is invalidated, i.e. if d_i is updated.

Multiversion with Autoprefetching (MA) : In method MA, the server maintains and broadcasts multiple versions for each item, instead of broadcasting the last committed value only. Versions correspond to different values at the beginning of each broadcast cycle and version numbers to the corresponding broadcast cycle. On each client side, T reads the most current version for its first read operation, that is the version with the largest version number v_0. For subsequent reads, T reads versions with the largest version numbers smaller than or equal to v_0. If such a version exists, T proceeds, else T is aborted. In addition, each client maintains local cache and adopts the invalidation with a form of autoprefetching for handling cache coherence. To support multiversioning, items in cache also have version numbers. For reading items from the cache, the same tests regarding their version numbers are performed as when reading items from the broadcast. To ensure that items in cache are current, the server broadcast an invalidation bit pattern described in Section 3.1.

From now, we will derive the basic equation that describes the expected average response time which is measured in the number of data items broadcast by the server. We begin by stating some assumptions of our model:

- There are D equal size date items in the database.
- Each broadcast cycle represents the state of the database at the beginning of the cycle.
- Each broadcast is preceded by an invalidation bit pattern.
- Updates occur following an exponential distribution, at an update rate of μ per item.
- Each mobile client will repeatedly query a subset of D with a high degree of locality. This subset is thus a "hot spot" for the client. Each item in the hot spot will be queried at the client at the rate r.

In the following analysis, we preclude the possibility of client's disconnections for the sake of simplicity. We also ignore the size of invalidation bit patterns in computing the expected average response time of all methods. This is because, one bit is assigned to each data item in an invalidation bit pattern, thus an invalidation bit pattern is comprised of only D bits whose size corresponds to only a few number of data items.

Note also that, in wireless data broadcast, the performance of a single client read-only transaction for a given broadcast program is independent of the presence of other clients transactions. As a result, we will analyze the environment by considering only a single client. In particular, we mainly focus on the nonuniform broadcast, since the

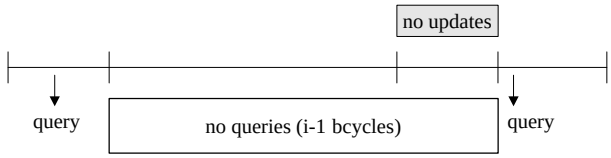


Figure 2. Scenario for Cache Hit Ratio

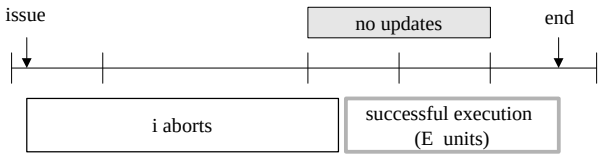


Figure 3. Scenario for Method IA

analysis for the uniform broadcast can be naturally derived from the methodology described here.

4.1. Size of Broadcast and Average Hit Ratio

4.1.1 Size of Broadcast

In the nonuniform broadcast, 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 [1], 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 $|NU|$ be the number of items in a single broadcast cycle for the nonuniform broadcast. Thus, the number of data items in a broadcast cycle is,

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

4.1.2 Average Hit Ratio

In order to compute the cache hit ratio, we assume that a query has occurred at a particular instant of time and compute the conditional probability of the value in the cache being valid. Note that the probability q_0 that no queries in a broadcast cycle is $e^{-r|NU|}$, and the probability u_0 that no updates in a broadcast cycle is $e^{-\mu|NU|}$. Let the two queries occur at i broadcast cycles of each other. Figure 2 shows the scenario that last query happened i broadcast cycles before the current one. Since the scheme of invalidation with autoprefetching is employed for cache management, all we need for the second query to be a hit is no updates during the one broadcast cycle immediately preceding the current one. Thus, the equation for the hit ratio for P-series and IA becomes,

$$h = (1 - q_0) \sum_{i=1}^{\infty} q_0^{i-1} u_0 = u_o \tag{2}$$

Notice that the cache hit ratio for MA, which will be shown in Section 4.2.2, is different from the above since the size of broadcast is increased by accommodating old versions.

4.2. Expected Average Response Time

Let a_s and a_t be the average response time for accessing a single data item and for accessing multiple data items, respectively, in a given transaction for the nonuniform broadcast. For the nonuniform broadcast, a_s 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 [12]. 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,

$$\omega(d_i) = \frac{|NU|}{2f_i} \tag{3}$$

, where f_i is the frequency of d_i. The expected a_s 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,

$$a_s = \sum_{d_i \in D} p(d_i)\omega(d_i) \tag{4}$$

4.2.1 Method IA

With the use of *IA*, a client retrieves data items in a one-at-a-time fashion, i.e. only after retrieving one item from a broadcast another request is issued. Let us first consider the case where there is no updates at the server so the execution of transaction is always committed successfully. The average response time for a transaction accessing m data items without local cache can be computed as

$m \sum_{d_i \in D} p(d_i)\omega(d_i)$. Then, the average response time of IA can be calculated by,

$$E = m(1-h) \sum_{d_i \in D} p(d_i)\omega(d_i) \quad (5)$$

If data items are updated at the server, however, a transaction may be aborted and restarted several times before it commits successfully. Figure 3 shows the scenario that its abort occurs i times before the successful execution. The probability c that one execution of transaction with IA leads to a successful commitment is smaller than or equal to $e^{-\mu|NU|m\lfloor\frac{E}{|NU|}\rfloor}$. This is because, in order for a client to commit its transaction successfully, m data items should not be updated during at least $|NU|\lfloor\frac{E}{|NU|}\rfloor$ units (note that it receives at least $\lfloor\frac{E}{|NU|}\rfloor$ invalidation bit patterns). Thus, the equation for response time (i.e. the difference between end and $issue$) is,

$$a_t(IA) = \sum_{i=0}^{\infty} E(1-c)^i = \frac{E}{c} \quad (6)$$

4.2.2 Method MA

In [8], three approaches have been proposed to maintain old versions of data items in the nonuniform broadcast: clustering, overflow bucket pool and new disks. With any approach to broadcast organization, keeping multiple versions in the nonuniform broadcast leads to the overall increased length of broadcast cycle, 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 selected broadcast organization carrying old versions of data items, we assume that there is some optimal broadcast organization in which the inter-arrival time of a data item is same. If the average number of data items that have updated during $|NU|$ is $N_c = D(1 - e^{-\mu|NU|})$ 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 broadcast is at least kN_c. For each data item $d_i \in D$, hence, the expected delay of d_i is,

$$\omega_c(d_i) = \frac{|NU| + kN_c}{2f_i} \quad (7)$$

, where f_i is the frequency of d_i on the broadcast accommodating old versions of data items together with up-to-date data items.

The expected average response time for any data request is calculated as $\sum_{d_i \in D} p(d_i)\omega_c(d_i)$. Therefore, the average response time for a transaction accessing m data items without local cache can be computed as $m \sum_{d_i \in D} p(d_i)\omega_c(d_i)$. Thus, the average response time of MA can be calculated by (we assume that each transaction is successfully committed),

$$a_t(MA) = m(1-h^v) \sum_{d_i \in D} p(d_i)\omega_c(d_i) \quad (8)$$

, where $h^v = e^{-\mu(|NU|+kN_c)}$ as the size of a broadcast is increased by kN_c (see the scenario for cache hit ratio in Figure 2).

4.2.3 Methods PA and PA^2

In our methods, a transaction processing is divided into 3 phases: preparation, acquisition and delivery phase. 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,

$$a_t(PA, PA^2) = PT + AT + DT \quad (9)$$

In a synchronous method PA, PT will be, on average, half of one broadcast cycle and DT is trivial, thus Expression (9) can be reduced to,

$$a_t(PA) \approx \frac{1}{2}|NU| + AT \quad (10)$$

For PA, the retrieval time for the first item from the broadcast is a_s itself. The retrieval time for the second item from the broadcast is $\frac{a_s}{|NU|}(= \delta)$ of the remaining broadcast size, and the retrieval time for the next item from the broadcast is in turn δ of the remaining broadcast size, and so on. Thus, the expected AT for a transaction with m_p predeclared items is,

$$AT(PA) = \sum_{i=1}^{m_p(1-h)} \delta(1-\delta)^{i-1}|NU| \quad (11)$$

The expected average response time of PA is therefore computed as,

$$a_t(PA) \approx \frac{1}{2}|NU| + \sum_{i=1}^{m_p(1-h)} \delta(1-\delta)^{i-1}|NU| \quad (12)$$

In an asynchronous method PA^2, both PT and DT are trivial, thus Expression (9) can be reduced to,

$$a_t(PA^2) \approx AT \quad (13)$$

For PA^2, AT involves retrieving some items in the broadcast cycle at which a transaction is issued and then downloading the remaining items (and possibly those items previously retrieved again) in the next broadcast cycle. We can compute the expected average response time of PA^2

Parameter	Value(s)
D	Varying (1000)
μ	Varying (5×10^{-4} per unit)
n	3
λ_1,λ_2, λ_3	4, 2, 1
$\|P_1\|$, $\|P_2\|$, $\|P_3\|$	Varying (50, 150, 800)
k (MA only)	2
m	Varying (10)
m_p	$\frac{3}{2}m$
f_{P_1}, f_{P_2}, f_{P_3}	Varying (0.7, 0.2, 0.1)
Cache Invalidation	Invalidation with Autoprefetching

Table 1. Parameter Settings

by using the expression for PA if the situation is divided into two: one is that those items achieved from the broadcast in the broadcast cycle at which a transaction is issued are not invalidated during acquisition phase, and the other is that some of those items are invalidated during acquisition phase. In the first case, since a client will, on average, retrieve $\frac{|NU|}{2a_s}$ items at the broadcast cycle at which a transaction is issued and the probability that the items are not invalidated is ${u_0}^{\frac{|NU|}{2a_s}}$, the expected AT of PA^2 is equal to the multiplication of Expression (11) and ${u_0}^{\frac{|NU|}{2a_s}}$. In the second case, the expected AT is equal to the multiplication of Expression (12) and $(1 - {u_0}^{\frac{|NU|}{2a_s}})$. Thus, the expected average response time of PA^2 is the sum of the two multiplications,

$$a_t(PA^2) \approx \frac{1}{2}|NU|(1-{u_0}^{\frac{|NU|}{2a_s}}) + \sum_{i=1}^{m_p(1-h)} \delta(1-\delta)^{i-1}|NU| \quad (14)$$

Note that, for both methods, the upper-bound of average response time is $\frac{3}{2}|NU|$. Further, the worst-case response time is bounded by $2|NU|$ irrespective of the number of data items or cache hit ratio.

For the uniform broadcast, the expected average response time for all methods can be naturally derived from the methodology described in this section. All we need to compute the average response time is therefore to substitute the size of broadcast with D, and a_s with $\frac{1}{2}D$ in all equation.

5. Analytical Results

We show some analytical results in this section. In the results, one time unit corresponds to the physical time taken to broadcast a single item on the server side. Table 1 summarizes the parameter settings for the server (the top half) and a client (the bottom half), where values in parenthesis are 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.

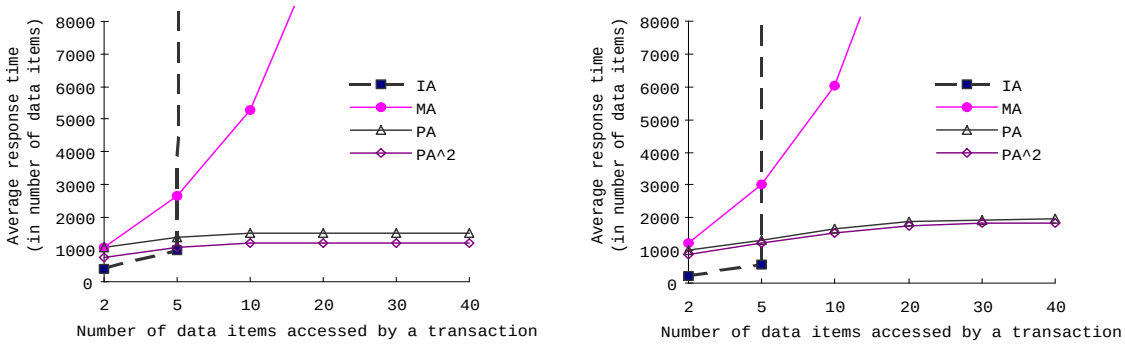
5.1. Effect of Transaction Size

In this analytical result, we show the effect of transaction size on various methods when μ is set to 5×10^{-4} per unit [1]. Figure 4 shows the performance behavior as the number of data items accessed by a transaction is increased in the uniform and the nonuniform broadcast content respectively. We see that for large data items (greater than 5 in our analysis), the response time of IA is increased rapidly. This is because a large value m decreases the probability of a transaction's commitment. As a result, a transaction suffers from many restarts until it commits. MA avoids this problem by making a client access old versions on each broadcast content, thereby increasing the chance of a transaction's commitment. We can observe that the performance of MA is less sensitive to the number of items than IA. However, the increased size of broadcast content affects the response time negatively in both broadcast contents. This explains why MA is inferior to IA for small data items. With our methods, as a transaction can access data items in the order they are broadcast, the average response time is almost independent of transaction size. As a result, our methods outperform MA, which in turn outperforms IA, when the number of items is large. For example, when m is 10, PA^2 yields the response time reduced by a factor of 3 on MA in both broadcast contents. Among ours, PA^2 exhibits only a marginal performance improvement over PA. This is because a high update rate makes the deployment of asynchronous approach useless.

Another interesting examination is on which broadcast organization provides better response time to each method. Over a single broadcast channel, we intuitively expected that the nonuniform broadcast would be better than the uniform one in both IA and MA methods even when data items are updated at high rate, because more resources are allocated to the data items that are accessed more often. However, we have got a rather unexpected result: MA shows similar response time behavior in both broadcasts for all range of transaction size (we have observed this by changing the access pattern and/or data partition parameters in various ways). With our default parameter settings, MA even favors the uniform broadcast over the nonuniform one. For instance, the response time in the uniform broadcast is reduced by about 15% from that in the nonuniform one, when m is 10.

Although this counter-intuitive result may be limited due to the constraint of our analysis (e.g. the sub-optimal broad-

[1] This value indicates, when D is set to 1000, that about 40% and 50% of database items are updated in the uniform and nonuniform broadcast content during a broadcast cycle.



(a) Uniform Bcast (b) Nonuniform Bcast

Figure 4. Effect of Transaction Size

cast frequencies of items in each partition or the sub-optimal number of items in each partition), we believe this trend will be applied to most cases for the following reasons: First, the uniform broadcast provides a higher cache hit ratio than the nonuniform broadcast. Second, the efficiency of data access under the nonuniform broadcast, which is based on access frequency on items, is mitigated by the inclusion of old versions of items (recall that each old version of an updated item is broadcast once in each broadcast cycle irrespective of its frequency). For IA, however, the second factor is not true although the first factor is still true. Since access optimization for a single item still dominates, the nonuniform broadcast gives IA better response time than the uniform one. Interestingly, our methods work better in the uniform broadcast than in the nonuniform broadcast with the exception of small data items. The reason is that the dominating factor to response time of predeclaration-based methods is the length of a single broadcast cycle when the update rate is high or large data items are retrieved.

5.2. Effect of Update Rate

Figure 5 shows the effect of update rate on the performance of various methods when m is set to 10. A higher update rate means a lower cache hit ratio, and also a higher probability of cache invalidation. This explains why the response time of IA deteriorates so rapidly. In particular, if $\mu > 2 \times 10^{-4}$ per unit[2], IA results in unacceptable performance. MA also degenerates as update rate increases. This is because a higher update rate leads to more number of updated items in the database, resulting in a longer broadcast content size. Unlike IA, however, with MA, a transaction can proceed and commit by reading appropriate old versions of items which are on the air. This difference of commitment probability is the main reason why MA beats IA for high update rate (in our analysis, when $\mu > 2 \times 10^{-4}$ per unit). For low update rate, there is a high probability that a transaction commit successfully even with IA. Thus, IA shows better response time than MA since the former retrieves each item more quickly than the latter.

With our methods, the response time is not affected by update rate significantly. As expected, Figure 5 shows that ours are superior to MA and IA when update rate is high. Only when a small portion of items in the database is updated during a broadcast cycle, the response time of ours is worse than that of MA and IA. Consistent with our expectation, however, PA^2 exhibits the average response time comparable to other methods in a rarely changing database where a transaction can be processed within a single broadcast cycle with the use of other methods. From the figure, we observe that the uniform broadcast is preferable for a high update rate, while the nonuniform broadcast works better for a low update rate.

5.3. Effect of Access Pattern

This analytical result 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 the server's averaged broadcast over the needs of a large client population [1].

To model such a mismatch between the needs of a client and the server's broadcast program, we have used three ac-

[2]This value indicates, when D is set to 1000, that more than 20% and 23% of database items are updated in the uniform and nonuniform broadcast content during a broadcast cycle.

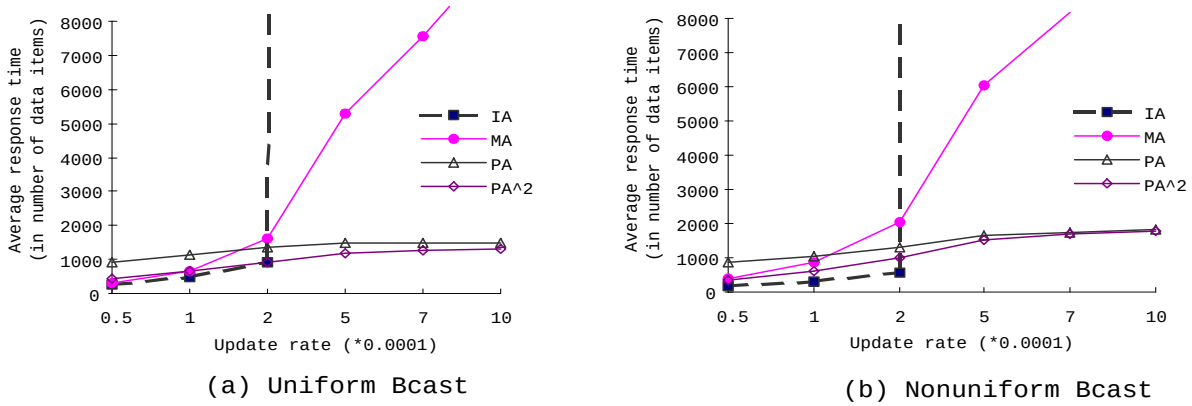


Figure 5. Effect of Update Rate

cess patterns under default data partitions: AP1 = (0.1, 0.2, 0.7), AP2 = (0.5, 0.3, 0.2) and AP3 = (0.7, 0.2, 0.1). There, AP1 is relatively the least matched access pattern, AP2 is less matched one, and AP3 is the relatively most matched one. We then observed the performance behavior of MA and our methods in the nonuniform broadcast content when a client follows different access patterns under the default update rate (from this section, we drop IA from performance presentation because of its poor performance and show the performance of PA and PA^2 in terms of upper-bound of expected average response time because of the similarity to the average response time). We could observe that the response time of MA gets worse by 20% as the mismatch becomes increasingly large from AP3 to AP1. This susceptibility to a broadcast mismatch is to be expected, as the client can not gain the benefit of the nonuniform broadcast content approach. In the case of our methods, however, the upper-bound response time is totally immune to the disagreement.

5.4. Effect of Data Partition

This result shows the sensitivity to the variance of data partition controlled by the server's broadcast program. Note that, in the nonuniform broadcast, different data partitions give rise to different broadcast cycle lengths. We can expect that as the broadcast cycle length is increased, the response time of our methods becomes increasingly worse.

To show this we have used three data partitions under default client access pattern: DP1 = (50, 150, 800), DP2 = (200, 300, 500) and DP3 = (500, 300, 200). We then observed the performance behavior of MA and ours in the nonuniform broadcast when different data partitions are employed at the server. We could observe that while, in method MA, the response time under DP3 gets worse by about 150% compared to DP1, the upper-bound response time of ours under DP3 gets worse by about 115% compared to DP1. In case of our methods, since a client has to spend much more time to retrieve the less commonly requested data in the nonuniform broadcast with much longer broadcast cycle, the difference of performance degradation is reasonable. In general, however, the nonuniform broadcast is constructed such that the fastest partition level has relatively smaller items, while subsequent levels are progressively larger [1], thus the high sensitivity of ours due to different data partitions will not be common.

5.5. Effect of Database Size

In this section, we show the case where the number of distinct data items are increased from 1000 to 4000 (the relative size at each partition is maintained exactly same as the default case), as opposed to the database size with 1000 items considered before. It can be expected that increasing the database size degrades the performance of both MA and our methods. This is because a larger database size results in (1) a longer broadcast cycle length and (2) a lower cache hit ratio.

We summarize the response time behavior under default client access pattern in Table 2. It can be observed that the performance of all the methods degrades when the database size is increased. As the improvement factor shows, however, ours are much more scalable to the increased database size than MA (especially in the the uniform broadcast content). This can be explained as follows. The response time of ours is negatively impacted only by a longer broadcast cycle length, while the response time of MA is negatively impacted by both a longer broadcast cycle length and a lower cache hit ratio. It is shown that ours beat MA in all cases and, in general, the performance gap gets bigger

Database Size	Uniform			Nonuniform		
	MA	PA, PA^2	Factor	MA	PA, PA^2	Factor
1000	5279	1500	2.5	6040	1950	2.1
2000	20290	3000	5.8	13938	3900	2.6
3000	30138	4500	5.7	20887	5850	2.6
4000	54355	6000	8.1	27091	7800	2.5

Table 2. Effect of Database Size

with the increased database size. For instance, the improvement factor over MA changes from 2.5 to 8.1 in the uniform broadcast content.

6. Conclusion

In this paper, we have proposed simple, but yet robust predeclaration-based methods to speed up processing of wireless read-only transactions while keeping the serializability for transactions in wireless data broadcast. As it turned out, although there are certain processing and storage overhead for predeclaration-based transaction processing compared with traditional ones, the benefit our approach brings in terms of greatly reduced response time outweighs the overhead.

7. Acknowledgement

This work was partly supported by the JSPS (Japan Society for the Promotion of Science) postdoctoral fellowship for foreign researchers in Japan.

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communication environments. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 199–210, 1995.

[2] S. Acharya, M. Franklin, and S. Zdonik. Disseminating updates on broadcast disks. In *Proceedings of the 22nd International Conference on Very Large Data Bases*, pages 354–365, 1996.

[3] D. Barbara. Mobile computing and databases: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):108–117, 1999.

[4] D. Barbara and T. Imielinski. Sleepers and workaholics: Caching in mobile environments. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 1–12, 1994.

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

[6] H. Garcia-Molina and G. Wiederhold. Read-only transactions in a distributed database. *ACM Transactions on Database Systems*, 7(2):209–234, 1982.

[7] T. Imielinski and R. Badrinath. Wireless mobile computing: Challenges in data management. *Communications of the ACM*, 37(10):18–28, 1994.

[8] E. Pitoura and P. Chrysanthis. Exploiting versions for handling updates in broadcast disks. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 114–125, 1999.

[9] E. Pitoura and P. Chrysanthis. Scalable processing of read-only transactions in broadcast push. In *Proceedings of the 19th International Conference on Distributed Computing Systems*, pages 432–439, 1999.

[10] J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran, and K. Ramamritham. Efficient concurrency control for broadcast environments. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 85–96, 1999.

[11] K.-L. Tan and B. C. Ooi. *Data Dissemination in Wireless Computing Environments*. Kluwer Academic Publishers, 2000.

[12] N. H. Vaidya and S. Hameed. Scheduling data broadcast in asymmetric communication environments. *Wireless Networks*, 5(3):171–182, 1999.