On Indexed Data Broadcast*

Sanjeev Khanna †   Shiyu Zhou‡

Abstract

We consider the problem of efficient information retrieval in asymmetric communication environments where multiple clients with limited resources retrieve information from a powerful server who periodically broadcasts its information repository over a communication medium. The cost of a retrieving client consists of two components: (a) access time, defined as the total amount of time spent by a client in retrieving the information of interest; and (b) tuning time, defined as the time spent by the client in actively listening to the communication medium, measuring a certain efficiency in resource usage. A probability distribution is associated with the data items in the broadcast, representing the likelihood of a data item's being requested at any point of time. The problem of indexed data broadcast is to schedule the data items interleaved with certain indexing information in the broadcast so as to minimize simultaneously the mean access time and the mean tuning time.

Prior work on this problem thus far has focused only on some special cases. In this paper we study the indexed data broadcast problem in its full generality and design a broadcast scheme that achieves a mean access time of at most $(1.5+\epsilon)$ times the optimal and a mean tuning time bounded by $O(\log n)$.

*A preliminary version of this paper appears in the *Proceedings of the 30th Annual Symp. on Theory of Computing*, 1998.

†Department of Fundamental Mathematics Research, Bell Laboratories, 700 Mountain Avenue, Murray Hill, NJ (`sanjeev@research.bell-labs.com`).

‡Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104-6389 (`shiyu@cis.upenn.edu`). The work was mainly done while the author was at Bell Laboratories, Lucent Technologies.

1 Introduction

We study the problem of efficient information retrieval in asymmetric communication environments. A communication environment is said to be *asymmetric* if the available/needed communication capacity from the information source to the information recipient is much larger than the communication capacity that is available/needed in the reverse direction. A representative example is the communication environment in which multiple mobile clients retrieve information from a server base-station over a wireless channel. The "pull-based" model used in the traditional client-server information retrieval systems, where clients retrieve data by making individual requests to the server, is poorly-suited for such environments. An alternative model is the so called "push-based" model, whereby the server broadcasts (pushes) its information repository onto the communication medium and multiple clients simultaneously retrieve the specific information of individual interest. This latter model has been extensively studied in the information systems community (sometimes under the name of *broadcast disks*) and is the model of choice for many asymmetric settings [1, 5, 13].

While the broadcast approach is effective in disseminating massive amounts of information to multiple clients, an individual client looking for certain data items may be required to actively listen to the medium for indefinitely long periods of time. From a client's perspective, the cost of this information retrieval process can be viewed as being composed of two distinct components: (a) the total time elapsed from the moment a client requesting a data item tunes in to the medium to the time when the required data item is received, and (b) the total time spent by the client *actively* listening to the communication medium. We refer to the first component as the *access time* and the second as the *tuning time*. The distinction between these two lies in the fact that the clients (such as the laptop computers in the context of wireless mobile computing) are assumed to be able to switch between the resource-consuming *active mode* and the resource-conserving *doze mode*. Since listening to the medium requires a client to be in the active mode, the server may provide certain *indexing information* in the broadcast so as to enable the clients to lapse into the doze mode during periods when no relevant information is being broadcast. (For example, the server may broadcast periodically as the indexing information a list of the keys of the data items to be broadcast in the coming period so that a client, upon receiving the list, can check the list to see whether or not the key of the requested item is in the list. If it is, then the client can keep listening and retrieve the item in the coming period. Otherwise, the client can first doze off and then wake up after this period when new indexing information comes in.) Therefore, so to speak, the tuning time forms a measure of the efficient utilization of certain important resources in the process of information retrieval (such as the limited power supply of mobile laptop computers).

The objective of this paper is to study the design of efficient broadcast schemes (i.e., protocols between the server and the clients) in which the server broadcasts data items interleaved with indexing information so as to minimize both the average access time and the average tuning time.

1.1 The Model and the Problem

We consider the single channel broadcast model in which a server (information provider) periodically broadcasts various *data items* (sequences of bits) over a fixed channel and the clients (information receivers) tune in to the channel and extract the data of interest.

Let n be the total number of data items to be broadcast. We specify that each item j is uniquely identified by its key (index), denoted $key(j)$, which is a distinct number between 1 and n assigned by the server. The clients can search for data items in the retrieval process only by making key comparisons.

Remark: The assignment of the keys to the data items is done by the server during the broadcast scheduling. We shall assume here that this assignment is also known to the clients. Although the validity of this assumption is not important for the theoretical results of this paper, it is critical for a practical implementation of the scheme under a scenario where the schedule may be dynamically varying. We assure a careful reader that this assumption is in fact easily supported by natural ways of implementing our scheme. However, since the implementation details are beyond the scope of the present paper, we will not further address this issue here.

A *broadcast* consists of a sequence of item buckets each of uniform size L bits. We assume that each bucket can hold up to $\log n$ keys. Since the key of any data item takes $\log n$ bits, we have $L \geq \log^2 n$. The assumption is reasonable since, for instance, buckets of 1KB can be used for broadcasts of 2^{90} data items.

The set of buckets of a broadcast is classified into two classes: *data buckets* and *index buckets*, where data buckets are used to contain data items and index buckets are used to contain certain indexing information such as a list of the keys of a set of data items. For ease of exposition, we assume without loss of generality that each data bucket of a broadcast contains exactly one complete data item. (Thus a data item is simply a sequence of L bits.)

We consider exclusively the broadcast that consists of repetitions of a *broadcast cycle*. We will often refer to a broadcast cycle as just a *broadcast* for convenience. The time required to broadcast/receive a bucket is assumed to be one *unit of time*, and we will measure both the access time and tuning time in terms of the time units.

A probability distribution $\vec{p} = (p_1, p_2, \ldots, p_n)$ is associated with the data items in the broadcast, where the probability p_j associated with data item j represents the "likelihood" of the item's being requested by the clients (independently) at any point of time (or the "popularity" of the item). The distribution is assumed to be known to the server prior to the broadcast: It either may be known as statistical data or can adaptively change from cycle to cycle in the broadcast. But it is fixed for any particular cycle and the scheduling of one cycle depends solely on the distribution.

Mean Access Time / Tuning Time: The *access time* of a request is defined to be the time elapsed from the moment the request is made (i.e., the moment when a client looking for a data item tunes in to the channel) to the time when the requested item is received by the client. The *mean access time* of a broadcast is the expected access time of a request (randomly chosen according to the distribution p) averaged over all possible moments of making the request.

That is, if we denote by N the total number of buckets in a broadcast cycle and by $W(t, j)$ the access time of a request for item j that is made at time t in a broadcast cycle, then the mean access time of the broadcast is

$$\mathrm{ACC} = \frac{1}{N} \sum_{t=1}^{N} \sum_{j=1}^{n} p_j W(t, j).$$

The *tuning time* of a request is the amount of time spent on listening to the channel from the moment when the request is made to the time when the requested item is received, where by *listening* we mean that the client is in the resource-consuming active mode. Observe that the access time is given by the sum of the time spent in the doze mode and the tuning time. The *mean tuning time* of a broadcast is defined analogously: if we denote by $T(t, j)$ the tuning time of a request for item j that is made at time t in a broadcast cycle, then the mean tuning time of the broadcast is

$$\mathrm{TUNE} = \frac{1}{N} \sum_{t=1}^{N} \sum_{j=1}^{n} p_j T(t, j).$$

Indexed Data Broadcast: An *indexed data broadcast scheme* consists of two protocols: a *server protocol* followed by the server to schedule the broadcast consisting of data items (in data buckets) interleaved with indexing information (in index buckets), and a *client protocol* followed by the clients to retrieve information in the broadcast (by switching between the active mode and the doze mode depending on the relevance of the information received). The problem, which we refer to as the *indexed data broadcast* problem, is to design an indexed data broadcast scheme that minimizes both the mean access time and the mean tuning time.

1.2 Related Work and Our Results

There are some obvious similarities between the problem of information retrieval in the aforementioned push-based broadcast model and the problem of classical database search. For example, any search in either problem is accomplished by key comparisons. However, we point out here a fundamental difference between these two problems. To retrieve an information item in the classical database search, the search may always begin at a certain well-defined location of choice in a data structure, say, the root of a balanced search tree for instance. A lot of work has been done to minimize the access time in this setting (see, for instance, [15, 10, 18, 16]). In contrast, however, in the broadcast model, the client begins its search based only on the information that is being broadcast at the moment it tunes in. This constitutes a unique aspect of the indexed data broadcast problem. One of the contributions of our work is to introduce new ideas to deal with this aspect of the problem.
Related Work: Using data broadcast as an information dissemination mechanism has been studied in both the network model [11, 6] and the wireless model [8] since the mid 80s. Much of the previous work has focused mainly on the problem of minimizing the mean access time in the model where broadcast consists solely of data items. In [21] and [4], an optimal condition on the mean access time for this case was shown. Anily *et al.* gave a factor of two

approximation algorithm for mean access time minimization [3]. Recently, Bar-Noy *et al.* [4] showed that the mean access time minimization problem is NP-complete even in this simpler model [1]. They proposed an approximation scheme that achieves a 9/8-approximation ratio. It is not known whether the problem of minimizing the mean access time is Max SNP-hard. It is also not known whether the techniques of Bar-Noy *et al.* could be used to minimize both the mean access time and the mean tuning time simultaneously. Some other related work can be found in [2, 7, 9, 12, 19, 20].

The problem of indexed data broadcast was first formalized by Imielinski *et al.* in [13]. They considered the simplest case where the distribution over data items is uniform and studied some easy broadcast schemes that achieve $(1+\epsilon)$-approximation to mean access time and $\lceil \log n \rceil$ mean tuning time, where n is the number of data items. Note that information-theoretic reasons imply that mean tuning time must be $\Omega(\log n)$ in the uniform distribution case. Indexed data broadcast in the *multi-channel* model has also been considered. Using $O(\log n)$ channels, an indexing scheme that achieves $O(\log n)$ mean tuning time was presented in [17].

Our Results: To our knowledge, there has been no prior work giving guaranteed good performance for the problem of indexed data broadcast with arbitrary distributions over data items. In this paper, we study for the first time the problem of indexed data broadcast in its full generality, in which the distribution over data items is arbitrary, and prove the following main result.

Theorem 1 *For each $\epsilon > 0$, there is an indexed data broadcast scheme that achieves mean access time at most $(1.5+\epsilon)$ times the optimal plus an additive $O(\log n)$ term, and mean tuning time upper bounded by $O(\log n/(\epsilon \log \epsilon^{-1}))$.*

We remark that for almost all the distributions of interest, the mean access time needed is super-logarithmic. (For example, for any distribution which has a linear number of probabilities of value $\Omega(\frac{1}{n})$, the optimal mean access time is $\Omega(n)$ as we will see by Lemma 2.) Therefore the $O(\log n)$ additive term in the mean access time approximation is essentially negligible. We further remark that the constants hidden in the complexities of the scheme are small and the scheme itself is fairly easy to implement.

1.3 Organization of the Paper

The remainder of this paper is organized as follows. In Section 2, we present in detail the construction of the broadcast scheme and establish some useful properties. In Section 3, we analyze the performance of our scheme and prove the main result of the paper, namely, Theorem 1.

[1]The NP-hardness proof of [4] requires that the sum of the p_i's is strictly less than 1. It is not known if the case where this sum is required to be 1 is NP-hard.

2 The Indexed Data Broadcast Scheme

In this section, we present the construction of our indexed data broadcast scheme and examine some of its basic properties. As mentioned in Section 1.1, an indexed data broadcast scheme consists of two protocols: a server protocol followed by the server to schedule the broadcast and a client protocol followed by the clients to retrieve information in the broadcast. Since the broadcast is scheduled by the server, it is typical that once the server protocol is completed, the client protocol would follow accordingly.

In what follows, we discuss the constructions of these two protocols in detail. For our discussion, we assume that $p_j \geq \frac{1}{n^{10}}$ for all $1 \leq j \leq n$; this assumption is without loss of generality as we indicate next. If it were not so, then we increase the p_j's that are less than $\frac{1}{n^{10}}$ to $\frac{1}{n^{10}}$. This causes an increase of at most $\frac{1}{n^9}$ in the total probability. We then perturb the largest probability in the distribution by this tiny amount to make the distribution well-defined, i.e., $p_j \geq \frac{1}{n^{10}}$ and $\sum_{j=1}^{n} p_j = 1$. It is not difficult to verify that this adjustment (essentially) does not affect the evaluation of the mean access time and the mean tuning time. We remark here that the threshold of $\frac{1}{n^{10}}$ is not important and it can be replaced by any reasonably small polynomial.

We denote by $[n]$ the set of integers $\{1, 2, \ldots, n\}$ and all logarithms are to the base 2.

2.1 The Server Protocol

Our construction of the server protocol can be broadly divided into three components:

1. the scheduling of the data items;
2. the design of the indexing mechanism; and
3. the scheduling of the broadcast that contains both the data and the indices.

The scheduling of the data items aims to minimize the mean access time while designing the indexing mechanism seeks to minimize the mean tuning time. The objective of scheduling the broadcast is to interleave the data items as scheduled in (1), and the indices as designed in (2), so as to minimize both the mean access time and the mean tuning time. We discuss next the construction of each of these components and examine some of its properties.

2.1.1 Scheduling of the Data Items

The problem of scheduling only the data items to minimize the mean access time has been studied extensively over the past few years. It was shown in [14] that in order to achieve the minimum mean access time, the broadcast should be arranged in a way that the instances (appearances) of each data item are *equally spaced.* The following lemma of [21] gives a quantitative characterization of the optimal scheduling of data items, and serves as the starting point in the construction of our schedule:

Lemma 2 *Let* $d_j^* = (\sum_{i=1}^n \sqrt{p_i})/\sqrt{p_j}$*. Then the mean access time is at least*

$$\mathrm{ACC}^* = \frac{1}{2}\sum_{j=1}^{n} p_j(d_j^* + 1) = \frac{1}{2} + \frac{1}{2}(\sum_{i=1}^{n}\sqrt{p_i})^2.$$

In fact, ACC^* *is achievable if all* d_j^**'s are integral and it is possible to place exactly* $d_j^* - 1$ *data items between every two consecutive appearances of item* j *in the broadcast (i.e., equal spacing is possible with respect to* d_j^**'s).*

One serious limitation of the above result is that equal spacing may not be achievable by any schedule, e.g., some d_j^* may not be integral. In fact, even if all d_j^*'s are integral, the problem of minimizing the mean access time has been shown to be NP-complete by Bar-Noy *et al.* [4] who also presented a 9/8-approximation algorithm for this problem. However, it is not known whether the resulting sequence of data items can be used for simultaneously minimizing the mean access time and the mean tuning time.

To overcome the above-mentioned problems, we first "shift" the optimal distance sequence d_j^* by some "small" amount so that the resulting sequence of distances, which we call a feasible sequence, becomes "nicer" in the sense that all distances in the sequence are integral, equal spacing is achievable, and moreover, the increase in the mean access time is small.

Definition 3 *A sequence of distances* $d_1, d_2, \ldots, d_n$ *is said to be* feasible *if it satisfies the following two requirements:*

- *each* d_j *is an integral power of 2; and*
- $\sum_{j=1}^n \frac{1}{d_j} \leq 1$.

In what follows, we first describe the procedure Shifting that shifts the optimal distance sequence d_j^* to a feasible sequence. Next we exhibit the procedure to schedule data items according to the feasible sequence of distances that results from Shifting. Then we describe the procedure to assign keys to the data items. We examine some properties achieved by these procedures at the same time.

Throughout the rest of the paper, we assume without loss of generality that

$$p_1 \geq p_2 \geq \ldots \geq p_n. \tag{1}$$

Then following Lemma 2, we have $d_j^* = (\sum_{i=1}^n \sqrt{p_i})/\sqrt{p_j}$ and therefore

$$d_1^* \leq d_2^* \leq \ldots \leq d_n^*.$$

Shifting: For integral $i \geq 1$, let $S_i = \{j \in [n] \mid 2^{i-1} < d_j^* \leq 2^i\}$ and $S_i' = \{j \in S_i \mid 2^{i-1} < d_j^* \leq (4/3)2^{i-1}\}$ (thus $S_i' \subset S_i$). Let K be the largest integer such that S_K is non-empty. Since elementary calculus shows that $\sum_{j=1}^n \sqrt{p_j} \leq \sqrt{n}$, and we assumed that $p_j \geq \frac{1}{n^{10}}$ for all j, it follows that $d_j^* \leq n^6$ and therefore $K \leq \lceil 6 \log n \rceil$. The shifting procedure is as follows:

Procedure **Shifting**

Input: optimal distance sequence $(d_1^*, d_2^*, \ldots, d_n^*)$, where $d_1^* \le d_2^* \le \ldots \le d_n^*$.

For each $1 \le i \le K$,

1. for each $j \in S_i$, if $j \in S_i \setminus S_i'$ then set $d_j' = 2^i$. Otherwise,
2. suppose $j_1 < j_2 < \ldots < j_p$ are the indices (subscripts j of d_j^*) in S_i'. Set $d_{j_l}' = 2^i$ if l is odd and $d_{j_l}' = 2^{i-1}$ otherwise.

Let $\vec{d} = (d_1, d_2, \ldots, d_n)$ be the sorted sequence of distances d_j';
Output: $\vec{d}$

In other words, for each $S_i, 1 \le i \le K$, if for an index $j \in S_i$, d_j^* is "big", i.e., more than $(4/3)2^{i-1}$, then we shift it to the next power of 2. For the set of d_j^*'s where $j \in S_i'$, i.e., d_j^*'s are "small", we shift them up and down alternately to the closest powers of 2. Finally we sort the shifted sequence.

Let us show that the output sequence is feasible. For this we need some definitions. Let X_1 be the set of indices j such that d_j' is shifted in Step (1). In Step (2), we pair up the set of indices in S_i' into pairs (j_{2k-1}, j_{2k}) for $k = 1, \ldots, \lfloor \frac{p}{2} \rfloor$, with one possible singleton exception j_p for the case that p is odd (j_p is the largest index in S_i'). Let X_2 be the set of indices that are paired up in Step (2) and let X_3 be the set of indices that have no matched pair in Step (2). It is clear that (X_1, X_2, X_3) is a partition of $[n]$.

Lemma 4 *The output sequence $\vec{d}$ of Shifting is feasible.*

Proof: Each d_j is a power of 2 by definition. So it suffices to show that $\sum_{j=1}^n \frac{1}{d_j} \le 1$. We will show that for each $i = 1, 2, 3$, $\sum_{j \in X_i} \frac{1}{d_j'} \le \sum_{j \in X_i} \frac{1}{d_j^*}$. Since $\sum_{i=1}^n \frac{1}{d_j'} = \sum_{j=1}^n \frac{1}{d_j}$ (sequence of d_j is just the sorted sequence of d_j') and $\sum_{j=1}^n \frac{1}{d_j^*} = 1$ (by Lemma 2), this will clearly complete the proof.

For $i = 1$ and 3, the statement is immediate since each such d_j' is obtained by shifting d_j^* to the next integral power of 2. For $i = 2$, we notice that X_2 consists of disjoint matched pairs and for each such pair $i, i+1 \in X_2$ (i is odd), if $i, i+1 \in S_k'$ for some $1 \le k \le K$ then, by the description of Step (2), we have $\frac{1}{d_i'} + \frac{1}{d_{i+1}'} = \frac{1}{2^{k-1}} + \frac{1}{2^k}$. But this is at most $\frac{1}{d_i^*} + \frac{1}{d_{i+1}^*}$ since $d_i^*, d_{i+1}^* \le (4/3)2^{k-1}$ by the definition of S_k'. □

Data Scheduling: Suppose $\vec{d} = (d_1, d_2, \ldots, d_n)$, where $d_1 \le d_2 \le \ldots \le d_n$, is the feasible sequence of distances output by procedure Shifting. For $1 \le j \le n$, let $\pi(i)$ be such that the distance in $\vec{d}$ corresponding to data item i is $d_{\pi(i)}$. Procedure Shifting first maps the optimal distance d_i^* corresponding to data item i to d_i' and then to $d_{\pi(i)}$ (by sorting) in the output sequence. By definition, $d_{\pi(i)} \le 2d_i^*$. It is clear that π is a permutation on $[n]$ and d_j is the distance in $\vec{d}$ that corresponds to data item $\pi^{-1}(j)$. Let us now describe the scheduling procedure.

Procedure **ScheduleData**

Input: $\vec{d} = (d_1, d_2, \ldots, d_n)$, where $d_1 \le d_2 \le \ldots \le d_n$ and $\vec{d}$ is feasible.

1. Initialize an array $Q[1..N_0]$ of N_0 empty data buckets, where $N_0 = d_n$;
2. For $j = 1$ to n, if the first available (empty) bucket in array Q is at position t, then we assign data item $\pi^{-1}(j)$ to the set of buckets at positions $t + ld_j$, $l = 0, 1, \ldots, \lfloor \frac{N_0 - t}{d_j} \rfloor$. (Such an empty bucket always exists in Q since $\vec{d}$ is feasible and thus $\sum_{j=1}^{n} \frac{1}{d_j} \le 1$.)

Output: Q.

Lemma 5 *In the schedule Q output by procedure ScheduleData, each data item is exactly equally spaced.*

Proof: It is clear from the description of the procedure that, to prove the lemma, all we need is to show that the above procedure is well-defined, i.e., no two data items are assigned to the same bucket by the procedure.

Suppose this is not the case. Then let $\pi^{-1}(j)$ be the first data item whose allocation causes a collision with an earlier allocated data item $\pi^{-1}(i)$ for some $i < j$. Let $t_i < t_j$ be the starting positions of data item $\pi^{-1}(i)$ and data item $\pi^{-1}(j)$, respectively. Then there exist l_i and l_j such that $t_i + l_i d_i = t_j + l_j d_j$. On the other hand, since $i < j$ we know that $d_i \le d_j$; moreover, since each d_k is an integral power of 2 we know that d_j is a multiple of d_i. Therefore, it must be the case that $t_j - t_i$ is a multiple of d_i as well. However, following Step 2 in the procedure, all such positions as t_j in Q (positions of the form $t_i + ld_i$) must have already been occupied by data item $\pi^{-1}(i)$ before item $\pi^{-1}(j)$ is scheduled. This contradicts the assumption that t_j is the starting position of data item $\pi^{-1}(j)$. $\square$

Remark: If there are empty buckets in Q when the scheduling is complete, we can delete them.

Next we examine the mean access time achieved by the scheduling.

Lemma 6 *Let us denote by* ACC_0 *the mean access time of the broadcast scheduled by ScheduleData($\vec{d}$), where $\vec{d} = (d_1, d_2, \ldots, d_n)$ is feasible. Then*

$$\mathrm{ACC}_0 = \frac{1}{2}\sum_{j=1}^{n} p_j(d_{\pi(j)} + 1) \le (1.5 + o(1))\mathrm{ACC}^* + O(\log n).$$

To establish the lemma, we begin with the following proposition.

Proposition 1 *Let $i, i+1$ (i odd) be two indices in X_2 that are paired up. Then*

$$\frac{3}{2}(p_i d_i^* + p_{i+1} d_{i+1}^*) \ge p_i d_{\pi(i)} + p_{i+1} d_{\pi(i+1)}.$$

Proof: Following the description of procedure Shifting and the definition of X_2, we have that $i, i+1 \in S'_k$ for some $k \in [K]$, and that $d_{\pi(i)} = 2^k$ and $d_{\pi(i+1)} = 2^{k-1}$. Moreover, by assumption, we have $d_i^* \le d_{i+1}^*$ and thus $p_i \ge p_{i+1}$. Denote p_i/p_{i+1} by α. Then $d_{i+1}^* = \sqrt{\alpha} d_i^*$ (by Lemma 2) and therefore, $1 \le \alpha \le 16/9$ since $2^{k-1} \le d_i^* \le d_{i+1}^* \le (4/3)2^{k-1}$.

Now to establish the proposition, it suffices to show that $\frac{3}{2}(\alpha d_i^* + \sqrt{\alpha} d_i^*) \ge \alpha 2^k + 2^{k-1}$. Since $d_i^* \ge 2^{k-1}$, the preceding inequality is implied by $\frac{3}{2}(\alpha + \sqrt{\alpha}) \ge 2\alpha + 1$, which in turn can be easily verified to hold for $1 \le \alpha \le 4$. The proposition follows. $\Box$

Proof of Lemma 6: Since $\vec{d}$ is feasible, ScheduleData on $\vec{d}$ gives an equal-spacing schedule by Lemma 5 and so by Lemma 2, we have:

$$\mathrm{ACC}_0 = \frac{1}{2}\sum_{j=1}^{n} p_j (d_{\pi(j)} + 1) = \frac{1}{2} + \frac{1}{2}\sum_{i=1}^{3}\sum_{j \in X_i} p_j d_{\pi(j)}.$$

To complete the proof, it is sufficient to show that, for $i = 1, 2$, $\sum_{j \in X_i} p_j d_{\pi(j)}$ is upper bounded by $\frac{3}{2}\sum_{j \in X_i} p_j d_j^*$; and for $i = 3$, it is either $o(\mathrm{ACC}^*)$ or $O(\log n)$.

For the case $i = 1$, this follows from the fact that for each $j \in X_1$, we have $d_{\pi(j)} \le \frac{3}{2} d_j^*$ by the definition of X_1 and Step (1) of procedure Shifting. For the case $i = 2$, this follows from Proposition 1 since X_2 solely consists of disjoint matched pairs as in the proposition.

For the case $i = 3$, we first notice that by the pairing up procedure, each S'_i may contribute at most 1 element to X_3. Therefore we have $|X_3| \le K \le \lceil 6 \log n \rceil$. Now,

$$\begin{aligned} \sum_{j \in X_3} p_j d_{\pi(j)} &\le 2 \sum_{j \in X_3} p_j d_j^* \\ &= 2\left(\sum_{i=1}^{n} \sqrt{p_i}\right) \sum_{j \in X_3} \sqrt{p_j} \\ &\le 2(\sqrt{2\mathrm{ACC}^*})(\sqrt{\lceil 6 \log n \rceil}). \end{aligned}$$

So if $\mathrm{ACC}^* = \omega(\log n)$, the contribution of the above summation to the mean access time is $o(ACC^*)$. Otherwise, $\mathrm{ACC}^* = O(\log n)$ and it contributes an additive term of $O(\log n)$. $\Box$

Key Assignments: Our key assignment to the data items i is defined to be

$$key(i) = \pi(i)$$

(equivalently, $key(\pi^{-1}(j)) = j$). That is, the keys of the data items i are ordered according to their corresponding distances $d_{\pi(i)}$ in $\vec{d}$.

Let us examine a property of this key assignment that will be useful for our subsequent analysis. First we need a definition.

Definition 7 *An* interval-partition *of a (multi)subset S of $[n]$ is a partition of S into disjoint intervals of integers. Such a partition is said to be* minimal *if there are no two intervals in the partition whose union is also an interval.*

It is readily seen that the minimal interval-partition of a set S is unique. We define the number of intervals in the minimal interval-partition of S to be the *interval partition number* of S. For example, the minimal interval-partition of $S = \{7, 6, 9, 3, 5, 1, 6, 2, 7\}$ is $([1, 3], [5, 7], [9, 9])$. So the interval partition number of S is 3.

Lemma 8 *Suppose $d_1 \le d_2 \le \ldots \le d_n$ is a feasible sequence of distances. Let J be an arbitrary subsequence of consecutive data items in Q resulting from ScheduleData$(d_1, d_2, \ldots, d_n)$, and let $T_J \subseteq [n]$ be the set of keys assigned to the data items in J. Then the interval partition number of T_J is at most $2K$, where $K = \log d_n$.*

To prove the lemma, we begin with the following observation.

Proposition 2 *For each $j \in [n]$, the starting position of data item $\pi^{-1}(j)$ in Q is at most d_j.*

Proof: Consider the interval I formed by the first d_j buckets in the array Q. For each $i < j$, the number of appearances of data item $\pi^{-1}(i)$ in I is at most d_j/d_i. Therefore, at the time when we start allocating data item $\pi^{-1}(j)$, there can be at most $d_j \sum_{i=1}^{j-1} \frac{1}{d_i}$ data items that have already been allocated in I. But this is less than d_j by the assumption that $\vec{d}$ is feasible and therefore there must be an empty bucket in I which can be used as the starting position for data item $\pi^{-1}(j)$. $\square$

Proof of Lemma 8: Let us partition the set $[n]$ of keys of n data items into K intervals $I_1, \ldots, I_K$ such that, for each $1 \le i \le K$ and each $j \in I_i$ we have $d_j = 2^i$. First we notice that the set of keys in each I_i forms an interval following our key assignment (the keys of the data items are ordered according to their corresponding distances in $\vec{d}$). We will show that after Step 2 in the scheduling, for each $1 \le i \le K$, the data items with keys in I_i appear in Q in a *round-robin* fashion. That is, suppose $I_i = \{k, k+1, \ldots, k+l\}$, then the corresponding data items of these keys in I_i appear in Q in the order of $\pi^{-1}(k), \pi^{-1}(k+1), \ldots, \pi^{-1}(k+l), \pi^{-1}(k), \pi^{-1}(k+1), \ldots, \pi^{-1}(k+l)$, etc. Then it is straightforward to see, by our key assignment, that each I_i can contribute at most 2 intervals in the minimal interval-partition of T_J. Since we have K such I_i's, these can contribute for a total of at most $2K$ intervals in the minimal interval-partition of T_J. This will complete the proof of the lemma.

By Proposition 2, we know that for each $j \in I_i$, the starting position of data item $\pi^{-1}(j)$ is at most $d_j = 2^i$. Since d_j is the distance between consecutive appearances of data item $\pi^{-1}(j)$ in Q after Step 2, each such item $\pi^{-1}(j)$ appears exactly once in the first 2^i buckets in Q. Moreover, since the starting position of each data item is always chosen to be the first available bucket and it proceeds in the order of the keys, the data items $\pi^{-1}(j)$ corresponding to the keys $j \in I_i$ appear in Q in order. Now Lemma 5 guarantees that the same pattern of these items appears in the next 2^i buckets and so on, which forms a round-robin. $\square$

2.1.2 The Indexing Mechanism

We use a balanced q-ary broadcast tree (q is to be determined) defined as follows: A *balanced q-ary broadcast tree* is a complete q-ary tree whose leaves are the data buckets in Q (we assume without loss of generality that the length of Q is an integral power of q). The internal nodes of the tree contain the indexing information, in which each internal node v consists of two *domains*:

- a storage scheme that stores the ranges of the keys associated with the data items contained in the leaves of the subtree rooted at v;
- a pointer to the first internal node w visited after v in a depth-first traversal of the broadcast tree excluding the nodes in the subtree of v. (The pointer is stored as a time offset indicating how long from now w would be broadcast.)

Remark: In the case that v has no succeeding nodes except those in its subtree, its pointer would point to the root of the broadcast tree for the next broadcast cycle.

In the storage scheme defined by the first domain of an internal tree node v, we use the data structure of a balanced tree (for example) to store the intervals in the minimal interval partition of the set of keys associated with the data items contained in the leaves of the subtree rooted at v. Then, in spite of the fact that the total number of distinct keys associated with the leaves contained in a subtree may be as large as $\Theta(n)$, Lemma 8 guarantees that the interval partition number of such a set of keys is at most $2K$. Since to specify any interval $[a, b]$ needs only to store its two endpoints a and b, and each key can be specified using $\log n$ bits, this takes a total of $O(K \log n)$ space.

An internal tree node consists of a sequence of index buckets that contains the above storage scheme of the first domain plus the pointer defined by the second domain, which is contained in the last bucket of the sequence. The total amount of storage needed by any internal node is thus bounded by $O(K \log n)$. Since each bucket in a broadcast is of uniform size L, we have:

Proposition 3 *Each internal node of the broadcast tree can be stored in $r = O((K \log n)/L)$ buckets.*

By assumption $K \leq \lceil 6 \log n \rceil$ and $L \geq \log^2 n$; thus we have $r = O(1)$.

We refer to the broadcast of a leaf of the tree as a *data burst* and the broadcast of an internal node of the tree an *index burst*. While a data burst involves the broadcast of merely one (data) bucket, an index burst typically spans several (index) buckets in a broadcast. However, the number of buckets in any index burst is upper bounded by $r = O(1)$ following the above proposition. We specify that all the buckets contained in any single index burst are broadcast consecutively in order.

2.1.3 Scheduling the Broadcast

A broadcast cycle, denoted B, is generated by a pre-order traversal of the broadcast tree. A broadcast schedule is simply an infinite sequence of the broadcast cycles.

2.2 The Client Protocol

2.2.1 Description of the Protocol

We now describe the client protocol used by the clients to retrieve a specific data item. By assumption (see the remark in Section 1.1), the key of the requested item is known to the client. In what follows, we assume that each bucket in a broadcast contains a (1-bit) "flag" indicating whether it is a data bucket or an index bucket and that each index bucket contains a flag indicating whether or not it is the first bucket of an index burst.

When a client tunes in to the broadcast, three possible scenarios arise:

1. The client is in the midst of receiving a sequence of data bursts. In this case, the client compares its search key j against the keys of the data items being broadcast. If the client finds a match, it simply downloads the data bucket and tunes off. Otherwise, it stays tuned in for the next burst.

2. The client finds the first bucket of an index burst. In this case, the client checks if the key j belongs to one of the key ranges specified in the index burst and then proceeds as follows:

   (a) If the key j indeed belongs to one of the ranges, the client stays tuned in for the next burst. Otherwise, if the next burst is a data burst, the client proceeds as in Step (1); else it repeats Step (2).

   (b) If the key j does not belong to any range specified in the current index burst, the client records from the terminal index bucket the time offset t (the pointer) for the broadcast of the next index burst and dozes off until then. (The time t indicates the earliest time from now that the server may broadcast the indexing information of an item not in the subtree.) The client repeats Step (2) upon waking up.

3. The client is in the midst of receiving an index burst. It waits until the beginning of the next burst. If the burst is a data burst then it proceeds as in Step (1); otherwise (it finds the first bucket of an index burst) it proceeds as in Step (2).

2.2.2 Basic Properties of the Protocol

Since the factor contributed by a single burst to the access time and the tuning time is negligible, we assume from here on that the client always tunes in to the broadcast at the beginning of a burst. We start with some definitions.

Let $R = R(B)$ denote the q-ary broadcast tree for the broadcast cycle B and let N_B denote the length of a broadcast cycle B, i.e., the total number of buckets in B. Suppose that data item j is requested by a client at time t in B. Let $v = v(t)$ be the node of R that is broadcast at time t and let $w = w(t, j)$ be the leaf of R (data bucket) containing data item j that is nearest to v in the future broadcast after time t. Denote by $p = p(t, j)$ the least common ancestor of v and w in R. Let c_v and c_w be the children of p on the path P_v from p to v and the path P_w from p to w, respectively. For each node $u \neq c_v$ on the path P_v, we denote by $SR(u)$ the set of siblings that lie to the right of u; and we set $SR(c_v)$ to be the set of children of p that lie between c_v and c_w (excluding c_v and c_w). Define $A(t,j)$ to be the union of $SR(u)$ over all the nodes u on P_v. Similarly, for each node $u \neq c_w$ on the path P_w, we denote by $SL(u)$ the set of siblings that lie to the left of u; and we define $D(t,j)$ to be the union of $SL(u)$ over all the nodes $u \neq c_w$ on P_w. Set $E(t,j)$ to be the set of nodes on the path P_w. Finally, let $V(t,j)$ be the (disjoint) union of $A(t,j), D(t,j)$ and $E(t,j)$.

We say that the client *probes* a node v in R if the client is in the active mode at the time when node v is broadcast.

Proposition 4 *If data item j is requested at time t in a broadcast cycle B then the set of nodes of $R(B)$ that the client probes is exactly $V(t,j)$.*

Proof: Recall the fact that the broadcast of B does a pre-order traversal on $R(B)$ and recall the pointer-jumping step, Step 2b, in the algorithm. It is then not difficult to see that the client first probes the nodes in $A(t,j)$ in a left-to-right and ascending fashion; next it reaches node c_w whose subtree contains the desired data bucket; then the client probes the nodes in $D(t,j) \cup E(t,j)$ in a left-to-right and descending fashion till it locates the leaf node w containing data item j. $\square$

Corollary 9 *Suppose that a client looking for data item j tunes in at time t in the broadcast. Then the client always succeeds in retrieving the first occurrence of data item j after time t in the broadcast.*

3 Performance Analysis

In this section we analyze the indexed data broadcast scheme as described in Section 2. We prove the main result, Theorem 1, the proof of which will be derived from the two lemmas stated below. Recall that the length of the data schedule Q (the number of the leaves in the broadcast tree) is N_0 and that r bounds the number of buckets needed to store the indexing information contained in an internal node. We denote by h the height of the q-ary broadcast tree. Then, by definition, $h = \log_q N_0$.

Lemma 10 *The mean access time* ACC *of the broadcast is at most*

$$(1 + \frac{2r}{q})\mathrm{ACC}_0 + \frac{hr+1}{2},$$

where ACC_0 *is as defined in Lemma 6.*

Lemma 11 *The mean tuning time* TUNE *of the broadcast is at most*

$$4qr \log_q \sum_{j=1}^{n} \sqrt{p_j} + (h+2q)r.$$

First let us see that together these lemmas imply our main result. By Proposition 3, we have that $r = O(1)$. Also by the definition of N_0, we have $N_0 = d_n \leq 2d_n^* \leq 2n^6$. Now we can choose the parameter q to be $\lceil 3r/\epsilon \rceil$ and thus $h = \log_q N_0 = O(\log n)$. So by Lemma 6 and Lemma 10, we have

$$\text{ACC} = (1.5 + \epsilon)\text{ACC}^* + O(\log n).$$

Also using the fact that $\sum_{j=1}^{n} \sqrt{p_j} \leq \sqrt{n}$ for any probability distribution, and Lemma 11, we get

$$\text{TUNE} = O(\log n/(\epsilon \log \epsilon^{-1})).$$

This completes the proof of the main theorem.

3.1 Proof of Lemma 10

Recall that $d_{\pi(j)}$ is the distance between any two consecutive appearances of data item j in the data schedule Q. Then $n_j = N_0/d_{\pi(j)}$ is the total number of appearances of data item j in broadcast cycle B. We denote by d_j^i, $i = 1, \ldots, n_j$, the distance between the ith and $(i+1)$st appearances of data item j in B, where $d_j^{n_j}$ is the distance from the last appearance of data item j in B to its first occurrence in the next cycle. It is clear that for any $j \in [n]$, $N_B = \sum_{i=1}^{n_j} d_j^i$.

Let $W(t, j)$, where $1 \leq t \leq N_B$ and $j \in [n]$, be the amount of time elapsed from the moment t (in a broadcast cycle) when the request for data item j is made to the time when data item j is received. By Corollary 9, if t is between the ith and $(i+1)$st appearances of data item j in B and the distance from t to the $(i+1)$st appearance of data item j is k, then $W(t, j) = k$. It is then straightforward to show that for any j

$$\sum_{t=1}^{N_B} W(t, j) = \sum_{i=1}^{n_j} \sum_{k=0}^{d_j^i} k.$$

So we have:

$$\begin{aligned}
\text{ACC} &= \frac{1}{N_B} \sum_{t=1}^{N_B} \sum_{j=1}^{n} p_j W(t, j) \\
&= \frac{1}{N_B} \sum_{j=1}^{n} p_j \sum_{i=1}^{n_j} \sum_{k=0}^{d_j^i} k \\
&= \frac{1}{2} \sum_{j=1}^{n} p_j \frac{\sum_{i=1}^{n_j} d_j^i (d_j^i + 1)}{\sum_{i=1}^{n_j} d_j^i} \\
&\leq \frac{1}{2} \sum_{j=1}^{n} p_j (\max_{i=1}^{n_j} (d_j^i + 1)).
\end{aligned}$$

However, for any $j \in [n]$ and $1 \leq i \leq n_j$, d_j^i is at most $(2d_{\pi(j)}/q + h)r + d_{\pi(j)}$, since within an interval of $d_{\pi(j)}$ data buckets in a broadcast, there can be at most $(2d_{\pi(j)}/q + h)$ index bursts each of length at most r (buckets). Thus we have

$$\begin{aligned}
\mathrm{ACC} &\le \frac{1}{2}\sum_{j=1}^{n} p_j((2d_{\pi(j)}/q + h)r + d_{\pi(j)} + 1) \\
&= \frac{1}{2}\sum_{j=1}^{n} p_j((1+\frac{2r}{q})d_{\pi(j)} + hr + 1) \\
&\le \frac{1}{2}(1+\frac{2r}{q})(\sum_{j=1}^{n} p_j(d_{\pi(j)}+1)) + \frac{hr+1}{2} \\
&= (1+\frac{2r}{q})\mathrm{ACC}_0 + \frac{hr+1}{2},
\end{aligned}$$

where the last equality is by Lemma 6. □

3.2 Proof of Lemma 11

Let $T(t,j)$, where $1 \le t \le N_B$ and $j \in [n]$, be the amount of time units spent by the client listening to the channel from the moment t (in a broadcast cycle) when the request for data item j is made to the time when data item j is received. Then

$$\begin{aligned}
\mathrm{TUNE} &= \frac{1}{N_B}\sum_{t=1}^{N_B}\sum_{j=1}^{n} p_j T(t,j) \\
&\le \sum_{j=1}^{n} p_j \max_t T(t,j).
\end{aligned}$$

So it suffices to give an upper bound on $\max_t T(t,j)$.

Following Proposition 4, it is easy to see that $T(t,j)$ is the number of time units spent on listening to (or equivalently, broadcasting) the nodes in $V(t,j)$. In fact, since the number of time units needed to broadcast any node in the tree is at most r, we have $T(t,j) \le r|V(t,j)|$. In what follows, we will first give an upper bound on $|V(t,j)|$. In fact, we will exhibit an upper bound $\kappa(j)$ on $|V(t,j)|$ that does not depend on t. Then it is clear that $\max_t T(t,j) \le r\max_t |V(t,j)| \le r\kappa(j)$.

Following the notation in Section 2.2, $|V(t,j)| = |A(t,j)| + |D(t,j)| + |E(t,j)|$. We will give an upper bound on each term in the summation.

First we observe that for every two nodes u_1 and u_2 in $A(t,j) \cup D(t,j)$, the set of leaves in the subtree rooted at u_1 and the set of leaves in the subtree rooted at u_2 are disjoint; moreover, data item j is not contained in any of these leaves. Let x be the node on the path P_v at the highest level in the tree such that $SR(x)$, the set of siblings that lie to the right of x, is non-empty. Let us use l to denote the level of x in the tree, level 0 containing the root. Then by the above observation, we have that $q^l < d_{\pi(j)}$. It is straightforward to show that $|A(t,j)| \le (q-1)(l+1)$, which is then at most $(q-1)(\log_q d_{\pi(j)} + 1)$. A similar argument shows that we can upper bound $|D(t,j)|$ by the same amount. Moreover, it is clear that $|E(t,j)| \le h$. So, overall, we can upper bound $|V(t,j)|$ by

$$h + 2(q-1)(\log_q d_{\pi(j)} + 1),$$

which we denote by $\kappa(j)$ and which does not depend on t as desired. Finally we have:

$$\begin{aligned}
\text{TUNE} &\leq \sum_{j=1}^{n} p_j \max_t T(t,j) \\
&\leq \sum_{j=1}^{n} p_j r \kappa(j) \\
&= \sum_{j=1}^{n} p_j r (h + 2(q-1)(\log_q d_{\pi(j)} + 1)) \\
&\leq \sum_{j=1}^{n} p_j r (h + 2(q-1)(\log_q 2d_j^* + 1)) \\
&\leq \sum_{j=1}^{n} p_j r (h + 2q(\log_q d_j^* + 1)) \\
&= 2qr(\log_q \sum_{j=1}^{n} \sqrt{p_j} + \sum_{j=1}^{n} p_j \log_q \frac{1}{\sqrt{p_j}}) + (h+2q)r \\
&\leq 4qr \log_q \sum_{j=1}^{n} \sqrt{p_j} + (h+2q)r
\end{aligned}$$

where the third inequality follows from the definition of $\pi(j)$ and the last inequality follows from the convexity of the logarithm function. $\square$

Acknowledgement

We would like to thank Badrinath, Ed Coffman and Peter Winkler for helpful discussions. We are also thankful to an anonymous referee for many valuable comments on an earlier version of this paper.

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments", *Proc. ACM SIGMOD Conf.*, May 1995.

[2] M. Ammar, J.Wong, "On the optimality of cyclic transmission in Teletext Systems", *IEEE Trans. Comm. 35 (11)*, 1987, pp. 1159-1170.

[3] S. Anily, C. Glass, R. Hassin, "The scheduling of maintenance service", *submitted for publication*, 1996.

[4] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber, "Minimizing Service and Operation Costs of Periodic Scheduling", *Proc. 9th ACM Symp. on Disc. Algorithms*, 1998.

[5] A. Bestavros and C. Cunha, "Server-Initiated Document Dissemination for the WWW", *IEEE Data Engineering Bulletin*, September 1996.

[6] T. Bowen et al., "The DATACYCLE architecture", *Comm. of the ACM, Vol 35, No. 12, December 1992, pp. 71-81.*

[7] M. Chan, F. Chin, "Schedulers for larger classes of pinwheel instances", *Algorithmica (9)*, 1996. pp. 425-462.

[8] D. Gifford et al., "The application of digital broadcast communication systems", *Stanford University Tech. Report, 1992.*

[9] C. Glass, "Feasibility of scheduling lot sizes of two frequencies on one machine", *European Journal of OR 75*, 1994. pp. 354-364.

[10] L. Guibas and R. Sedgewick, "A dichromatic framework for balanced trees", *IEEE FOCS*, 1978. pp. 8–21.

[11] G. Herman et al., "The datacycle architecture for very large high throughput database systems", *Proc. ACM SIGMOD conf.*, 1987, pp. 97-103.

[12] R. Holte, L. Rosier, I. Tulchinsky, D. Varvel, "Pinwheel scheduling with two distinct numbers", *Theoretical Computer Science* (100), 1992, pp. 105-135.

[13] T. Imielinski, S. Vishwanathan, and B. Badrinath. Energy Efficient Indexing on Air. *Proc. ACM SIGMOD Conf.*, May 1994.

[14] R. Jain and J. Werth, Airdisks and airRAID: modelling and scheduling periodic wireless data broadcast (extended abstract). *DIMACS Tech. Report 95-11, Rutgers University, May 1995.*

[15] D. Knuth, "The Art of computer programming", *Vol. 1, Addison-Wesley*, 1973.

[16] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees", *Comm. ACM 33(6)*, 1990, pp. 668–676.

[17] N. Shivakumar and S. Venkatasubramanian, Energy-efficient indexing for information dissemination in wireless systems. *ACM-Baltzer Journal of Mobile Networks and Nomadic Applications (MONET), December 1996.*

[18] D. Sleator and R. Tarjan, "A data structure for dynamic trees", *JCSS 26(3)*, 1983, pp. 362–391.

[19] S. Su, L. Tassiulas, "Broadcast scheduling for information distribution", *INFOCOM*, 1997.

[20] W. Wei, C. Liu, "On a periodic maintenance problem", *OR Letters 2*, 1983, pp. 90–93.

[21] N. Vaidya and S. Hameed, "Improved Algorithms for Scheduling Data Broadcast", *Technical Report 96-029*, Dept. of Computer Science, texas A&M University, 1996.