Energy-Efficient Tree-Based Indexing Schemes for Information Retrieval in Wireless Data Broadcast*

Jiaofei Zhong[1], Weili Wu[1], Yan Shi[1], and Xiaofeng Gao[2]

[1] The University of Texas at Dallas, Department of Computer Science, 800 West Campbell Road, Richardson, TX 75080-3021
{fayzhong,weiliwu,yanshi}@utdallas.edu

[2] Georgia Gwinnett College, 1000 University Center Lane, Lawrenceville, GA 30043
xgao@ggc.edu

Abstract. Mobile computing can be equipped with wireless devices to allow users retrieving information from anywhere at anytime. Recently, wireless data broadcast becomes a popular data dissemination method, especially for broadcasting public information to a large number of mobile subscribers at the same time. *Access Latency* and *Tuning Time* are two main criteria to evaluate the performance of a data broadcast system. Indexing can dramatically reduce tuning time by guiding clients to turn into doze mode while waiting for the data to arrive. *B^+-Tree Distributed Index Scheme (BTD)* is a popular index scheme for wireless data broadcast, which has been extended by many research works. Among traditional index structures, alphabetic Huffman tree is another important tree-based index technique with the advantage of taking data's access frequency into consideration. In this paper, we are trying to answer one question: can alphabetic Huffman tree replace B^+-tree and provide better performance? To answer this question, we propose a novel *Huffman-Tree based Distributed Index Scheme (HTD)* and compare its performance with *BTD* based on a uniform communication environment. The performances of *HTD* and *BTD* are analyzed both theoretically and empirically. With the analysis result, we conclude that *HTD* outperforms *BTD* and can replace *BTD* in most existing wireless data broadcast system.

1 Introduction

Wireless Data Broadcast has attracted great attention recently in wireless computing area because of its scalability and flexibility to broadcast public information to a large number of mobile subscribers. In a typical data broadcast system, base stations broadcast a set of data periodically within its region. Mobile clients within the region could tune into broadcast channel, search for the data it needs, wait till target data items are broadcasted and download them. Considering that mobile devices has limited battery power and restricted lifetime, *access latency* and *tuning time* becomes two main criteria to evaluate the performance of a data broadcast system. According to the architectural enhancements, each mobile device has two modes: *active mode* and *doze mode*. The energy consumed

* This work is supported by NSF grant CCF-0829993 and CCF-0514796.

in active mode can be up to 100 times higher than that in doze mode. Based on this, access latency is defined to denote the whole time interval from the moment when a client initiates a query, till the moment it finishes downloading the data item, which evaluates the query time efficiency of a system; while tuning time is defined as the sum of time when a client keeps "active" during the process, which evaluates the energy efficiency of the system.

Researchers apply index technologies to reduce tuning time for a data broadcast system. An index is a specific data structure storing the location information of data items. Due to the nature of data broadcast scheme, indices in data broadcast system store the "time offset" of target data items. Once a client gets this offset, it is aware of the waiting time for the target data item to arrive on the broadcast channel. The client turns into doze mode to save some energy and tunes back to the broadcast channel right before the data item appears. Different index technologies have different searching efficiency. If we insert indices between data items, then the whole size of a program will increase, resulting a longer access latency. Therefore, discussing about an index technology, researchers will always consider the balance between tuning time and access latency.

B^+-Tree Distributed Index Scheme (BTD) is a popular index scheme for wireless data broadcast. Many other research works [3,4,5,18] have extended *BTD* with respect to different system configurations. Since the idea of distributed index can be generally adopted on tree-based search index methods, we naturally wonder what impact the choice of different search tree structures will have on the performance of broadcast system. Alphabetic Huffman tree is another primary tree-based index techniques. Compared with B^+-tree, it can not only guide searching of target data item, but also take into consideration the access frequencies of different data items. The higher access frequency a data item has, the closer it is to the root in an alphabetic Huffman tree. This can be a very beneficial feature for wireless data broadcast because it may reduce the time needed to search for more frequently requested data items and consequently reduce the average access latency. The purpose of this paper is to construct an Alphabetical *Huffman-tree based Distributed Index Scheme (HTD)* and evaluate the performance of distributed index schemes *BTD* and *HTD*.

For fair comparison, we build up a uniform environment with same communication model and data set. We assume each data item can have different size and different access probability, such that our mathematical model can be more practical and more accurate. Since system performance in skew broadcast heavily relies on data schedule algorithm/design, and we just want to compare the performance of indices, flat broadcast is adopted in this paper. We choose single channel data broadcast model to eliminate the impact of index and data allocation algorithms on the performances so that the difference in performances can be purely from different tree-based index structures. In order to perform the evaluation, we first develop a novel *Huffman-Tree based Distributed index scheme (HTD)*. We also adjust the packet design of *BTD* to fit our communication model. Based on the uniform system setup, a detailed theoretical analysis is performed on the expected access latency and tuning time of *BTD* and *HTD*.

Finally, we simulate the broadcast environment and provide mass numerical simulation. The theoretical and empirical analysis proves the superiority of Alphabetic Huffman tree based distributed index. We are the first work to construct a Huffman-tree based Distributed Index Scheme for wireless data broadcast problem.

The rest of this paper is organized as follows: in Section 2 we study previous literatures for wireless data broadcast problem, including various index technologies in different communication environments. In section 3 we illustrate our system model, and discuss broadcast environment, data type, and data schedule in detail. In Section 4 and 5 we construct and evaluate distributed index and Huffman tree correspondingly. Next, in Section 6 we illuminate the process of simulation and discuss index performance based on our numerical experiments. Finally, Section 7 gives conclusion and the plan of our next stage work.

2 Related Works

In wireless data broadcast area, the main research topics always focus on how to design index structures and how to allocate data on channels. Their purpose is to reduce access latency and tuning time, in order to improve the energy efficiency of the system. Many research works deal with data scheduling problem so as to decrease access latency. Acharya et al. [1] proposed "broadcast disk", which allocates data with similar access frequencies onto different disks and broadcast data of these disks repeatedly according to their frequencies, in order to cope with nonuniform access distribution. Vaidya et al. [16] discussed optimization issue with respect to the average access latency when data access distribution is nonuniform. Vlajic et al. [17] presented an optimized data broadcast strategy in hierarchical cellular organization system. However, none of these works implements indexing technique. Moreover, without doze mode, the tuning time is as long as access latency, which leads to high power consumption of mobile devices.

There are also many works converting traditional disk-based indexing approaches to air indexing by converting physical address into time offset. One paper [12] discussed a signature based approach for information filtering in wireless data broadcast. Another work by Xu et al. [19] gave an idea of exponential index that shares links in different search trees and allows clients to start searching at any index node. However, their approach may not perform well under nonuniform access probabilities. Yao et al. [22] proposed *MHash* to facilitate skewed data access probabilities and reduce access latency. Imielinski et al. [8] presented the flexible index and hash based index. Furthermore, they customized B$^+$-tree index and proposed (1, m) index as well as distributed index (BTD) [9]. BTD was extended by many other researchers to fit different system requirements. Hu et al. [5] designed a hybrid index scheme combining BTD and signature-based index. [18] proposed an index allocation method named TMBT for multi-channel data broadcast, which creates a virtual BTD for each data channel and multiplexes them on the index channel. Hsu et al. [4] modified *BTD* to deal with data with nonuniform access frequencies. In [3], Gao et al. built a complete multi-channel broadcast system to broadcast a data set with nonuniform access probability and data sizes, which used *BTD* as their index scheme.

Huffman-tree is a skewed index tree which takes into account the access probability of each data item, that the popular data has a shorter path from the root of the tree, thus it minimizes the average tuning time [2,14]. In [2], the proposed algorithms for constructing the skewed Huffman tree have a problem that the clients may fail to find the desired data item by traversing that Huffman tree. In [14], it discussed the construction of Huffman tree, which is similar to that of Huffman code, but the constructed Huffman tree has the same problem. There is another kind of Huffman tree, Alphabetic Huffman tree, proposed in [6], which serves as a binary search tree. It is further extended to k-ary search tree in [14], so that a tree node will fit in a wireless packet of any size by adjusting the fanout of the tree. However, all the above works discussed Huffman-tree on multi-channel environment. When it comes to the multi-channel data broadcast, how to allocate index and data will produce heavy impact on the performance of each index technique. A certain allocation method could be helpful to specific index structure, but at the same time it might reduce the efficiency of another index method. In this paper, we aim at comparing two commonly used index approaches under the same conditions, as well as minimizing both average access latency and average tuning time, so we adopt single channel data broadcast environment to avoid all kinds of influences introduced by a multiplicity of different multi-channel allocation methods. Unfortunately, there is no existing research applying alphabetic Huffman-tree onto single channel data broadcast systems.

3 System Symbols and Bucket Design

Now we present the system model of a wireless data broadcast communication environment for future comparison of tree-based distributed index schemes.

Table 1. Symbol Description

Sym	Description	Sym	Description
D	Data set $D = \{d_1, \cdots, d_t\}$	D_i	Data block on $\mathbb{B}_i$
t	Number of data items	P_i	Probability for block $\mathbb{B}_i$
P	Probability set $P = \{p_1, \cdots, p_t\}$	$\triangle_i$	The i^{th} subtree at level l+1 on T
S	Length set $S = \{s_1, \cdots, s_t\}$	V_i	Distributed path for $\triangle_i$
T	An index tree	u_i	Length of index on $\mathbb{B}_i$ with average u
L	Level of T	v_i	Length of V_i on $\mathbb{B}_i$ with average v
k	Maximum branch number for T	x_i	Length of D_i on $\mathbb{B}_i$ with average x
l	Threshold to cut T	R	Total number of $\triangle_i$ on T
$\mathbb{B}$	One broadcast sequence on a channel	B_i^j	The j^{th} index at i^{th} level of T
$\mathbb{B}_i$	The i^{th} block on $\mathbb{B}$	d_i^j	The j^{th} bucket of data item d_i
$\lvert \cdot \rvert$	Cardinality of one set	$\lVert \cdot \rVert$	Length measured in data bucket unit

3.1 System Symbols

The Base Station broadcasts data set D on a single wireless broadcast channel, where the total number of data items is t, and $D = \{d_1, d_2, \cdots, d_t\}$. Without loss of generality, assume data items in D are arranged in a consecutively increasing order of their primary key values. The access probability for each data item d_i is p_i, where $\sum_{i=1}^{t} p_i = 1$, and P indicates the probability set of D. Data items may have different sizes due to various applications, Let s_i denote the size of d_i, and S denote the length set of D. Fig. 1 is an example data set with 16 data items, which will be continuously used as our data sample throughout the paper.

In order to reduce tuning time for mobile clients, some tree-based index strategies, for instance the B^+*-tree Index* scheme, are applied to the wireless data broadcasting system. We use T to denote the index tree for tree-based index strategies, and define k as the maximum number of branches for each node in T. L is the depth or height of T. In *distributed index* [9], T is "cut" at the l^{th} level. B_i^j denotes the j^{th} index at i^{th} level of T. Sec. 4 and 5 will give detailed design of two index strategies. Table 1 lists most of the symbols used in this paper. Some symbols will be introduced in later sections.

Data Key	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Probability	0.08	0.01	0.04	0.03	0.06	0.10	0.05	0.07	0.10	0.06	0.02	0.04	0.05	0.05	0.07	0.16
Data Size	4	2	3	1	4	2	4	2	3	1	1	3	4	2	1	3

Fig. 1. An Example of Data Set

3.2 Bucket and Pointer Design

A *bucket* is the minimum logical unit for data transmission in wireless data broadcast systems. Data buckets and index buckets have different structures and sizes. An index bucket contains a complete index node; while a data item can take several data buckets. Data item size s_i is measured by the number of data buckets it occupies. A bucket has two segments: **head** and **payload**. For both index bucket and data bucket, its **head** has the same elements:

bId: id of a bucket, in the format of (i, j, n). For a tree-based distributed index bucket, it indicates the n^{th} recurrence of index B_i^j. For a data bucket, it denotes the j^{th} bucket of d_i (denoted as d_i^j) with $n = s_i$.

bType: the type of this bucket. E.g., a tree-based distributed index strategy has three types of buckets, i.e. control index, search index and data.

bLength: the total length or size of this bucket.

bOffset: the offset to the next nearest control index.

The **payload** segments of a data bucket and an index bucket are different. In a data bucket, the payload stores data. A data item may take up several data buckets of same lengths. On the other hand, in an index bucket, the payload stores index information, such as pointers, which indicate the time offsets to some other index buckets. A pointer contains the following elements:

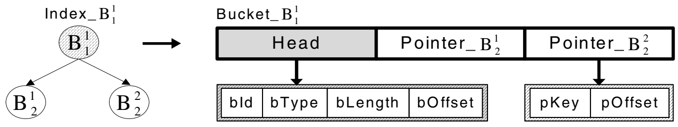

Fig. 2. An Example of Index Bucket Structure

pKey: the **bId** of the bucket it points to.
pOffset: time offset from current moment to the moment target bucket starts to broadcast.

For tree-based index strategies, an index bucket may contain several pointers, each pointing to one of its children. The number of pointers depends on the design of the index tree. Fig. 2 is an example index bucket storing an index node B_1^1 of a binary index tree, which has a head segment (the block in shadow) to "label" index B_1^1 itself, and a payload segment (two white blocks) to store the pointers of B_1^1. Since B_1^1 has two children B_2^1, and B_2^2, its payload segment should have two pointers, recording the location of B_2^1, and B_2^2.

4 B$^+$-Tree Based Distributed Index

We adopt B$^+$-Tree based distributed index strategy *BTD* introduced in [9]. To fit our model, we reformulate *BTD* with the bucket design in Sec. 3.2.

In *BTD*, B$^+$-Tree index is streamed on broadcast channel in *depth-first* manner, and it is "cut" at level l. Nodes from level 1 to l are *replicated part*, while others are *non-replicated part* which can be viewed as a number of subtrees rooted at the indices at level $l+1$. Each index in the replicated part is a *control index* and has a *control table* to specify the search ranges of different subtrees.

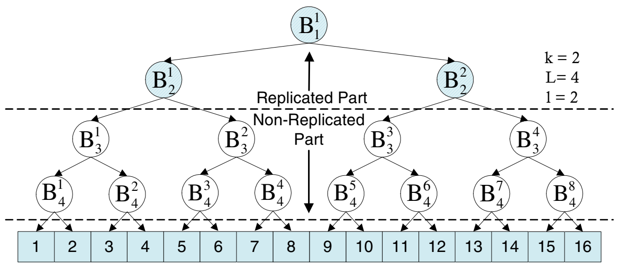

Fig. 3. An Example of B^+-tree cut at the 2^{nd} level

Fig. 3 is an example of a full binary B$^+$-Tree based distributed index structure with $k = 2$, $L = 4$, and $l = 2$. There are 16 data items in the data set D, represented by grey blocks at the bottom. Each index node B_i^j means the j^{th} index node on the i^{th} level of the tree. All the nodes above (including) the 2^{nd}

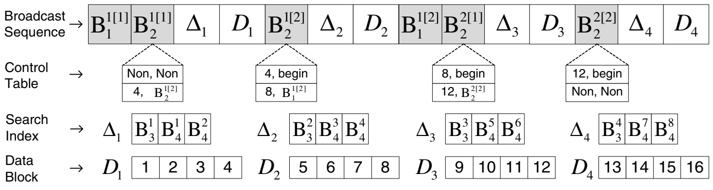

Fig. 4. An Example of $\mathbb{B}$ with Control Tables

level of the tree are control indices of the replicated part, while the other nodes below are search indices of the non-replicated part.

We traverse T according to distributed rules described in [9], and then append control table for each control index. During traversing process, data buckets and index buckets are interleaved on the same broadcast channel. $\mathbb{B}$ is a complete program streamed on a broadcast channel, including both index buckets and data buckets. Fig. 4 is an example of $\mathbb{B}$ with respect to the aforementioned index tree example in Fig. 3. $B_i^{j[1]}, \cdots, B_i^{j[k]}$ represent k appearances of B_i^j, where k is the same as branch number k in T.

Note that index bucket and data bucket may have different sizes. A data bucket is usually measured by KB. Each data bucket has size 1KB. However, the size of an index bucket is determined by the information stored in an index bucket. Therefore, we let $|\mathbb{B}|$ denote the cardinality of set $\mathbb{B}$, measured by the number of bucket, and $\|\mathbb{B}\|$ denote the total length of set $\mathbb{B}$, measured in the unit of one data bucket (KB). An index bucket may have different size from a data bucket, so we define "r" to indicate the ratio of data bucket size to index bucket size, i.e. $r = \textit{data bucket size/index bucket size}$. For instance, using the data set in Fig. 1 as an example, in Fig. 4 we have $|\mathbb{B}| = 34$, and $\|\mathbb{B}\| = 18/r + 40$, since there are totally 18 indices, 6 of which are control indices and the rest 12 are search indices. Additionally, *control tables* are used to specify the ranges of subtrees. For example, in the control table of $B_2^{1[2]}$, the first entry [4,begin] means that if the client is looking for a data item with key value ≤ 4, it needs to wait till the beginning of next broadcast cycle. The second entry [8, $B_1^{1[2]}$] implies if the client is looking for a data item with key value > 8, it should wait till $B_1^{1[2]}$ arrives. This control table indicates that the subtree immediately following it can only guide to data with key values in the range (4,8].

In *BTD*, index and data are interleaved on the same broadcast channel. As in Fig. 4, $\triangle_i$ denotes subtree in the non-replicated part, and is consist of search indices. For instance, $\triangle_2$ is the subtree rooted at B_3^2, with two children B_4^3 and B_4^4. D_i indicates the data buckets that $\triangle_i$ guides to, which is streamed sequentially by their key values. $\text{DFT}(\triangle_i)$ is the depth-first traversal of $\triangle_i$. For example, $\text{DFT}(\triangle_2) = B_3^2, B_4^3, B_4^4$. $\text{PATH}(B_i^j)$ is a path from root B_1^1 to node B_i^j (excluding the endpoint B_i^j), and V_i is a *distributed path* before each $\triangle_i$. For example, from Fig. 3 we can see that the distributed path for B_3^3 should be $V_3 = \{B_1^1, B_2^2\}$. After this, the broadcast sequence is defined as $\mathbb{B} = \{V_1, \text{DFT}(\triangle_1), D_1, V_2, \text{DFT}(\triangle_2), D_2, \cdots, V_R, \text{DFT}(\triangle_R), D_R\}$.

4.1 Performance Analysis of B$^+$-Tree Distributed Index

In this section, we analyze the performance of B$^+$-Tree based distributed index with respect to the expectation of *access latency* and *tuning time*.

Let's consider access latency first. Let R denote the number of subtrees after we cut T. The whole broadcast cycle is divided into $\mathbb{B}_1, \cdots, \mathbb{B}_R$ blocks, where $\mathbb{B}_i = \{V_i, \text{DFT}(\triangle_i), D_i\}$, for $1 \leq i \leq R$. P_i represents the access probability for block $\mathbb{B}_i$, which can be derived by summing up the probability of all data buckets that belong to data block D_i of $\mathbb{B}_i$, i.e. $P_i = \sum_{j \in D_i} p_j, for\ i = 1, \cdots, R$. Let v denote the average length of V_i, u the average length of $V_i + \triangle_i$, and x the average length of D_i. Note that u, v, and x are measured by data bucket(KB), while u_i, v_i, and x_i denote corresponding lengths for specific block $\mathbb{B}_i$. Hence, we have $u = (\|\mathbb{B}\| - \|D\|)/R$, $v = \sum_{i=1}^{R} \|V_i\|/R$, and $x = \|D\|R$.

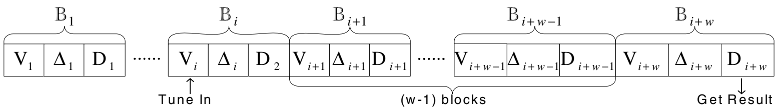

Fig. 5. An Example of a Client Searching for Data

Theorem 1. *If B^+-tree based distributed index and data are interleaved on one broadcast channel, then the average access latency is*

$$E(AL) = \frac{1}{R} \cdot \sum_{i=1}^{R} \Big(\sum_{w=1}^{R-2} ((\frac{1}{2} + w)u + wx) \cdot P_{(i+w)\%R} + (u - \frac{v}{2} + \frac{x}{2}) \cdot \frac{v}{u+x} \cdot P_i$$
$$+ (\frac{u-v}{2} + w(u+x)) \cdot P_i \cdot \frac{u-v+x}{u+x} \Big). \qquad (1)$$

Proof. Assume that a client would like to retrieve data item d_j. It first tunes into the broadcast channel at block $\mathbb{B}_i$, and then waits for another w blocks to reach the block which contains the required datum d_j at $\mathbb{B}_{i+w}$. Next, the client waits for the first data bucket of d_j to come and begins to download, until it gets all the data buckets of d_j. Illustration of the whole process is shown in Fig. 5. There are three possible cases with respect to the length w:

Case 1: $\mathbf{1 \leq w < R}$. We divide this case into 3 phases:

1. the client tunes into block $\mathbb{B}_i$. It takes an average $(u+x)/2$ time to visit $\mathbb{B}_i$;
2. the client waits $(w-1)$ complete blocks, which takes $(w-1)(u+x)$ time;
3. it finds the index directly pointing to d_j in $\triangle_{i+w}$ and download data. The average waiting time is $u + x/2$.

The expected access latency of this case is shown below, where b denotes the current block number, and d means the distance from b to the block it gets data:

$$E(AL|b = i, d = w) = \frac{u+x}{2} + (w-1)(u+x) + u + \frac{x}{2} = (\frac{1}{2} + w)u + wx$$

Case 2: $\mathbf{w} = \mathbf{0}$. The client tunes into V_i of block $\mathbb{B}_i$, and finds the pointer to required data item which is indeed in the following $\triangle_i$ of the same block $\mathbb{B}_i$. In this case, it takes only aforementioned phases 1) and 3), so we have:

$$E(AL|b = i, d = 0) = \frac{v}{2} + u - v + \frac{x}{2} = u - \frac{v}{2} + \frac{x}{2}$$

Case 3: $\mathbf{w} = \mathbf{R}$. Suppose the client tunes into block $\mathbb{B}_i$, and the required data is also in this block $\mathbb{B}_i$. Unfortunately, the client already missed the control index of this block when it tunes in, so it has to wait for the next control index in the next block to continue searching, and then wait for $\mathbb{B}_i$ to be broadcast again in the next *bcast*. In this case, the expected access latency becomes:

$$E(AL|b = i, d = R) = \frac{u - v + x}{2} + (w - 1)(u + x) + u + \frac{x}{2} = \frac{u - v}{2} + w(u + x)$$

According to the law of total expectation and the above three cases, we can get the average access latency as in Thm. 1.

Theorem 2. *The average tuning time for B^+-Tree based distributed index is*

$$E(TT) = \sum_{i=1}^{R} \frac{3u_i - v_i + (2+r)x_i}{r\|\mathbb{B}\|} + \frac{2L - l}{2} + \sum_{i=1}^{|D|} s_i p_i \qquad (2)$$

Proof. The tuning time of searching and downloading one data item comprises the following phases:

Step I: The client tunes into broadcast channel, and searches for the right *control index.* Since the client can start searching only from a control index, we analyze this phase by three cases:

Case 1: *The first visited bucket is a control index.* Then the client could follow the control table to find the right control index in one more step, which is discussed in [3]. The probability of this case is $\sum_{i=1}^{R} v_i/\|\mathbb{B}\|$, and the average tuning time of this case is $\frac{2}{r}\sum_{i=1}^{R} v_i/\|\mathbb{B}\|$.

Case 2: *The first visited bucket is a search index.* The client needs to wait for the next nearest control index, and follow its control table to reach the target control index. This has a probability of $\sum_{i=1}^{R}(u_i - v_i)/\|\mathbb{B}\|$, and average tuning time is $\frac{3}{r}\sum_{i=1}^{R}(u_i - v_i)/\|\mathbb{B}\|$.

Case 3: *The first visited bucket is a data bucket.* The client also need to wait for the next control index, and then go to the target control index, with probability $\sum_{i=1}^{R} x_i/\|\mathbb{B}\|$. The average tuning time is $(1 + \frac{2}{r})\sum_{i=1}^{R} x_i/\|\mathbb{B}\|$.

Step II: The client searches for the index that directly points to the required data. The average number of visited index bucket in this step is $\frac{1}{r}\left(\frac{l}{2} + (L - l)\right) = \frac{1}{r}(L - \frac{l}{2})$.

Step III: The client sleeps until the required data appears, and then tunes in again to download data. The average downloading time is $\sum_{i=1}^{|D|} s_i p_i$.

Combining above steps, we have the average tuning time as in Thm. 2.

In order to get the actual values of the average access latency and average tuning time, we need to know the values of L, R, $\mathbb{B}$, u, v, and x. The total level L of an index tree is determined by the number of branches k of T and the size t of data set D. Since the total number of pointers at the bottom level of T should be equal to the number of data items, the number of leaf nodes on T should be at least $\lceil t/k \rceil$, and the number of nodes at the second lowest level of T should be at least $\lceil \lceil t/k \rceil /k \rceil$. In this way, we can calculate the size of each level inductively, until we reach the root of T. We define $N(L)$ as the set of nodes at the L^{th} level of T. There is an algorithm in [3] describing how to compute L and $N(L)$. With known L and $|N(L)|$, we can get $R = |N(l+1)|$. What's more, if T is a full k-ary tree, there is a theorem as follows.

Theorem 3. *If T is a full k-ary tree, then the total number of index buckets in a bcast is $\frac{k-k^L}{1-k} + k^l$, where $L = \lceil \log_k t \rceil$, $l < L$, and k, l are fixed parameters.*

The detailed proof of this theorem could be found in [3]. We can also get the value of other variables after the construction of $\mathbb{B}$.

5 Huffman-Tree Based Distributed Index

Huffman-tree index has been applied to the wireless broadcast environment ever since the last decades. It is an efficient index technique because it takes into account the access probability of data items when constructing the Huffman-tree. The popular data with higher probability reside closer to the root in Huffman-tree, which reduces search time when traversing from the root. Considering flat broadcast, we found that the distributed method could be extended to Huffman-tree based broadcast, which is an innovative idea that has not been considered before. In this section, we will discuss the construction of *Huffman-Tree based Distributed Index Scheme* (*HTD*) and perform a theoretical analysis on its efficiency.

The structure of index bucket and data bucket in *HTD* is almost the same as in *BTD*. The first step is to construct a k-ary Alphabetic Huffman-Tree following the methods introduced in [14]. Here we give an example of the construction process based on the data set and access frequency in Fig. 6. Note that if we normalize the access frequency in Fig. 6, we will get the same data set as in Fig. 1. The Alphabetic Huffman-Tree construction is shown in Fig. 7 and 8.

Stage 1: choose data nodes d_i, d_j as candidates to be merged when

1. there are no leaves between them,
2. the sum of their frequencies is the minimum,
3. d_i and d_j are the leftmost nodes among all candidates.

If the above conditions hold, we create a new index node d_i' with frequency equal to the sum of d_i's and d_j's frequencies, and replace d_i, d_j with d_i' in the construction sequence. This stage produces a tree T_0 without alphabetic ordering of the data nodes, as in Fig. 7, where we record the frequencies of each index node inside the circle as index key values.

Stage 2: record the level of each data node (leaf node) of T_0, denoted as L_i of data d_i. The root node level is 1. From bottom to the root, rearrange pointers such that for each level the leftmost two nodes have the same parent, and then the next two, and so on. We can generate an alphabetic Huffman-Tree T in this way, without changing the level of each node in T_0, as shown in Fig. 8.

We could easily extend this algorithm to construct k-ary Huffman-Tree, by merging at most k nodes in stage 1, and combining up to k nodes with the same parent in stage 2.

Data Item Key	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Frequency	23	4	12	10	17	31	15	21	29	19	7	12	16	14	20	48

Fig. 6. An example Data Set of Huffman-Tree based Distributed Index

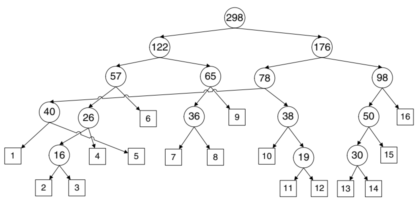

Fig. 7. The first step of constructing T_0

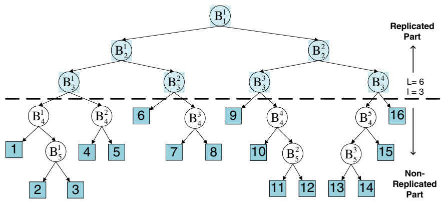

Fig. 8. The final Huffman-Tree T

After generating the alphabetic Huffman tree T, we cut T at level l, and perform a distributed traversal as Sec. 4. The index nodes above l is still called control index, and index nodes below l is search index. We append control tables onto control index in the same way as Sec. 4.

Note that there are two major difference between Huffman tree and B$^+$-tree, that the position of leaf nodes and the subtree sizes below l are different. It is possible that there might be data items above l in a Huffman tree, depending on which level we choose for l, since data items are not restricted to reside at bottom level of Huffman tree (they could also appear in higher level), which is a major difference with B$^+$-tree. Another difference is that sizes of subtrees below l may vary a lot in Huffman tree, but for B$^+$-tree each subtree has similar size.

The final broadcast sequence $\mathbb{B}$ generated in this example is illustrated in Fig. 9; and the access protocol is described in Alg. 1. When searching in a control table, client firstly compares the key value of request data and that of the first entry in control table; if request key is less than or equal to the first key, it should wait until the root of next bcast; if request key is greater, the client will go on to compare with the next entry in control table, and turn to doze mode

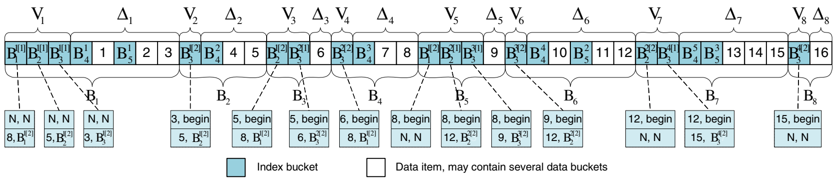

Fig. 9. The broadcast sequence of Huffman-Tree based Index

for *offset* time if request key is not greater than the next key in control table; otherwise, it will continue comparing with the rest entries of control table until it finds such an entry, or go to default bucket (the next index) if not found.

Algorithm 1. ***Retrieve Data***

```
    Input: $key_{req}$                              ▷ key value of request data $d_{req}$
    Output: $d_{req}$.
 1: Access randomly onto broadcast channel;
 2: $B_0$ = current bucket;
 3: if $B_0$ is data bucket & $B_0$.bId $= (key_{req}, 1, s_{req})$ then
 4:     Download data $d_{req}$;
 5: end if
 6: if $B_0$.bType $\neq$control then
 7:     Doze $B_0$.bOffset time till the next control index;
 8:     $B_0$ = current bucket;
 9: end if
10: Follow $B_0$'s control table and go to the pointed bucket;
11: Download data $d_{req}$;
```

5.1 Performance Analysis of Huffman-Tree Distributed Index

In this section, we analyze the system performance of Huffman-Tree based distributed index by evaluating its expected access latency and tuning time.

First let's consider access latency. Similar as *BTD*, all index and data buckets are interleaved on one broadcast channel. The whole broadcast cycle is divided into $\mathbb{B}_1, \cdots, \mathbb{B}_R$ blocks, where $\mathbb{B}_i = \{V_i, \text{DFT}(\triangle_i)\}$, for $1 \leq i \leq R$. Note that here V_i and $\triangle_i$ are different with those in *BTD*; V_i is still distributed path, but $\triangle_i$ may contain data. If there is no data node above l, V_i is the same as in *BTD*, and $\triangle_i$ is the whole subtree including data; however, if there is data node above l, V_i represents distributed path excluding data nodes, and $\triangle_i$ indicates all data items following V_i above l. We use P_i to represent the access probability for block $\mathbb{B}_i$, while P_i can be derived by summing up the probabilities of all data buckets that belong to $\mathbb{B}_i$, i.e. $P_i = \sum_{j \in B_i} p_j, for\ i = 1, \cdots, R$. Let v_i denote the length of V_i, and δ_i indicate the length of $\triangle_i$. Furthermore, an index bucket may have different size compared to a data bucket, so we continue to use "r" as the ratio of data bucket size to index bucket size.

Theorem 4. *If Huffman-Tree based distributed index and data are interleaved on one broadcast channel, then the average access latency is*

$$E(AL)=\frac{1}{\|\mathbb{B}\|}\sum_{i=1}^{R}(\sum_{w=1}^{R-2}(\frac{v_i+\delta_i}{2}+\sum_{j=i+1}^{i+w-1}(v_j+\delta_j)+v_{i+w}+\frac{\delta_{i+w}}{2})P_{(i+w)\%R}(v_i+\delta_i)$$
$$+(\frac{v_i+\delta_i}{2})P_i v_i+\sum_{i=1}^{R}(v_i+\delta_i)P_i\delta_i). \quad (3)$$

Proof. Assume a client want to get data item d_j. It first tunes into the broadcast channel at block $\mathbb{B}_i$. Then, it waits for another w blocks to reach the index which contains the pointer to d_j at $\mathbb{B}_{i+w}$. Within $\mathbb{B}_{i+w}$, the client waits for the first data bucket of d_j to be broadcast and begins to download, until it gets all the data buckets of d_j. There are three possibilities about the length of w:

Case 1: $\mathbf{1 \leq w < R}$. We can divide this case into three phases: 1) the client tunes into block $\mathbb{B}_i$, and takes an average $(v_i+\delta_i)/2$ time in it; 2) it waits through $(w-1)$ complete blocks, which takes $\sum_{j=i+1}^{i+w-1}(v_j+\delta_j)$ time; and 3) it finds the pointer to the datum in $\triangle_{i+w}$, and then download data, so the average waiting time is $v_{i+w}+\delta_{i+w}/2$. The expected access latency of this case:

$$E(AL|b=i,d=w)=\frac{v_i+\delta_i}{2}+\sum_{j=i+1}^{i+w-1}(v_j+\delta_j)+v_{i+w}+\frac{\delta_{i+w}}{2} \quad (4)$$

Case 2: $\mathbf{w = 0}$. The client tunes into V_i of block $\mathbb{B}_i$, and the pointer to required data is indeed in the following bucket of the same block $\mathbb{B}_i$. In this case, it only contains aforementioned phases 1) and 3) of the first case, so the expected access latency becomes:

$$E(AL|b=i,d=0)=\frac{v_i}{2}+\frac{\delta_i}{2}=\frac{v_i+\delta_i}{2} \quad (5)$$

Case 3: $\mathbf{w = R}$. Suppose the client tunes into block $\mathbb{B}_i$, and the required data is in the same block $\mathbb{B}_i$. Unfortunately, the client already missed the index buckets, so it has to wait for the next available index in the next block to continue searching, and then wait for $\mathbb{B}_i$ to be broadcast again in the next broadcast cycle. In this case, the expected access latency is:

$$E(AL|b=i,d=R)=\frac{\delta_i}{2}+\sum_{j=i+1}^{i+w-1}(v_j+\delta_j)+v_i+\frac{\delta_i}{2}=\sum_{i=1}^{R}(v_i+\delta_i) \quad (6)$$

Combining equation (4), (5), (6), using law of total expectation, we can get the average access latency as in Thm. 4.

Theorem 5. *The average tuning time for alphabetic Huffman-Tree based distributed index scheme is*

$$E(TT)=\frac{2\sum_{i=1}^{R}v_i+(2+r)|D|+3\sum_{i=1}^{R}\delta_i}{r\|\mathbb{B}\|}+\sum_{i=1}^{|D|}(\frac{l}{2r}+\frac{1}{r}(L_i-l)+s_i)p_i \quad (7)$$

Proof. The tuning time of searching and downloading one data item comprises the following steps:

Step I: The client tunes into broadcast channel, and search for the right index, following which it can get the required data on that same block. Consider these three cases:

Case 1: *The client first tunes into a control index.* Then the client could follow the control table to find the right control index in one more step, which is discussed in [3]. The probability of this case is $\sum_{i=1}^{R} v_i/\|\mathbb{B}\|$, and the average tuning time of this case is $\frac{2}{r}\sum_{i=1}^{R} v_i/\|\mathbb{B}\|$.

Case 2: *The first visited bucket is a data bucket.* The client need to wait for the next nearest control index, and then go to the target control index, with a probability of $|D|/\|\mathbb{B}\|$. Thus, the average tuning time of this case is $(1+\frac{2}{r})|D|/\|\mathbb{B}\|$.

Case 3: *The first visited bucket is a search index.* The client also need to wait for the next nearest control index, and follow its control table to reach the target control index. This has a probability of $\sum_{i=1}^{R} \delta_i/\|\mathbb{B}\|$, and average tuning time is $\frac{3}{r}\sum_{i=1}^{R} \delta_i/\|\mathbb{B}\|$.

Step II: Next, the client searches for the pointer that directly points to the required data. Then it sleeps until the required data appears, and tunes in again to download data. The average time of this step is $\sum_{i=1}^{|D|}(\frac{l}{2r} + \frac{1}{r}(L_i - l) + s_i)p_i$, where L_i is the level of data d_i in the Huffman-Tree.

Finally, summarizing the above steps, we can get the average tuning time of Huffman-Tree based distributed index as in Thm. 5.

6 Simulation

In this section, we use simulation results to evaluate the performance of *HTD* and *BTD*. Our system is implemented using Java 1.6.0 on an Intel(R) Xeon(R) E5520 computer with 6GB memory, and Windows 7 v6.1 operating system.

6.1 Simulation Settings

We simulate a base station with single broadcast channel, broadcasting a database with 10,000 data items [20], each of which has different sizes from 1KB to 4KB, and multiple clients requesting various sets of data items. The access probability of each data item satisfies the zipf distribution [13], a model for non-uniform access patterns [10,18]. Each data bucket is set of size 1KB, and we could calculate the size of each index bucket as [3] to be 0.1KB. Normally, in real applications, the ratio of index bucket size over data bucket size is $\frac{1}{r} = 0.1$, but other papers never discuss about this. They assume index bucket is of the same size as data bucket, which is not accurate. Therefore, in our simulation we set up $r = 10$, which is much closer to the reality scenario. Moreover, for each group of experiments, we generate 10,000 requests based on data access probabilities, in order to calculate the average access latency (AAL) and average tuning time (ATT) during data retrieval more accurately.

6.2 Simulation Results

When the size of the request set is 10,000, the ratio of index bucket size over data bucket size $\frac{1}{r} = 0.1$, we vary the size of database to compare AAL and ATT of HTD and BTD. From Fig. 10 we can see that HTD has much shorter average access latency than BTD, while both HTD's and BTD's average access latencies gradually increases as the database size increases. Note that the measurement of Y-axis denotes the unit time to read one data bucket. Thus, we can claim that HTD reduces the average response time of request retrievals. Moreover, as the database size increasing, the average access latency gap between HTD and BTD is growing larger, and the advantage of HTD becomes more obvious. From Fig. 11 we can see that no matter how database size increased, HTD always needs less average tuning time than BTD during retrieval, which means HTD is more energy-efficient. As the database size increasing, the average tuning times gap between HTD and BTD is growing larger, which implies that the energy advantage of HTD is getting more obvious.

Next, we evaluate the broadcast cycle length of HTD and BTD. Due to different tree structures, these two approaches have different Bcast length after index and data allocation, although the distributed traversal methods are similar. We use $||\mathbb{B}||$ to represent the length of one Bcast on broadcast channel. We consider the bucket size ratio r when analyzing the length of Bcast. As in Fig. 12, the Bcast of BTD is always longer than that of HTD. The reason is that BTD has much more index nodes than HTD due to its tree structure, even when they use the same data set and same cutting level. Therefore, using HTD for data broadcast will reduce the total length of data stream on broadcast channel.

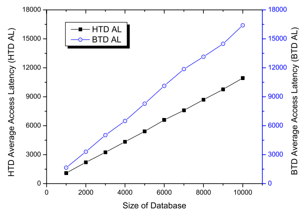

Fig. 10. AAL w. r. t. $|D|$

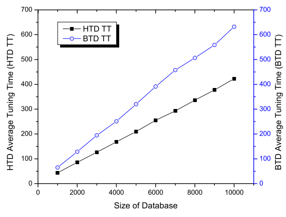

Fig. 11. ATT w. r. t. $|D|$

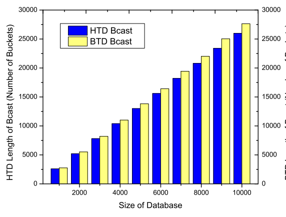

Fig. 12. $||\mathbb{B}||$ w. r. t. $|D|$

7 Conclusion

In conclusion, we are the first work to propose a promising strategy under wireless data broadcast environment, which is a novel Huffman-Tree index scheme combined with distributed index strategy. Specifically, we formally define an uniform communication environment, redesign and enhance *B^+-tree distributed index* (BTD) structure and broadcasting scheme, propose a novel *Huffman-tree based distributed index* (HTD), theoretically analyze each scheme under the same environment and same criteria, and then evaluate the performance of them by experiments. Simulation results show two major advantages of *HTD*: (1) it is more energy efficient, and (2) it reduces response time significantly. Therefore, *HTD* outperforms *BTD* in all major criteria.

All in all, our contribution includes three aspects. Firstly, we construct the uniform communication environment for wireless data broadcast system, and provide structured design of distributed index and Huffman-tree index. We follow the latest and most efficient construction for both index technologies, and redesign/modify some part of them such that they could be applied in the uniform communication environment. Next, we provide a general theoretical analysis to evaluate the performance of each index. Such analysis can be applied easily to majority indices commonly used in data broadcast. It can be a standard to evaluate the efficiency of an index technique. Thirdly, we simulate data broadcast system with a large number of numerical experiments, using the same group of sample data, such that the output will be reliable. Simulation results reveals that Huffman-tree distributed index is more power efficient and also responses much faster. Our future work includes developing more efficient index schemes for wireless data broadcast and provide more theoretical analysis on them.

References

1. Acharya, S., Alonso, R., Franklin, M., Zdonik, S.: Broadcast disks: Data management for asymmetric communication environments, pp. 199–210 (1995)
2. Chen, M., Yu, P., Wu, K.: Indexed Sequential Data Broadcasting in Wireless Mobile Computing. In: ICDCS 1997 (1997)
3. Gao, X., Shi, Y., Zhong, J., Zhang, X., Wu, W.: SAMBox: A Smart Asynchronous Multi-Channel Blackbox for B^+-Tree based Data Broadcast System under Wireless Communication Environment, Submitted to Information Sciences (2010)
4. Hsu, C., Lee, G., Chen, A.: Index And Data Allocation On Multiple Broadcast Channels Considering Data Access Frequencies. In: MDM 2002, pp. 87–93 (2002)
5. Hu, Q., Lee, W., Lee, D.: A Hybrid Index Technique for Power Efficient Data Broadcast. Distrib. Parallel Dat. 9(2), 151–177 (2004)
6. Hu, T., Tucker, A.: Optimal Computer Search Trees and Variable-length Alphabetic Codes. SIAM J. Appl. Math. 21(4), 514–532 (1971)
7. Hurson, A., Muñoz-Avila, A., Orchowski, N., Shirazi, B., Jiao, Y.: Power-Aware Data Retrieval Protocols for Indexed Broadcast Parallel Channels. Pervasive and Mobile Computing 2(1), 85–107 (2006)
8. Imielinski, T., Viswanathan, S., Badrinath, B.: Power Efficient Filtering of Data on Air. In: Jarke, M., Bubenko, J., Jeffery, K. (eds.) EDBT 1994. LNCS, vol. 779, pp. 245–258. Springer, Heidelberg (1994)
9. Imielinski, T., Viswanathan, S., Badrinath, B.: Data on Air: Organization and Access. IEEE TKDE 9(3) (1997)
10. Jung, S., Lee, B., Pramanik, S.: A Tree-Structured Index Allocation Method with Replication over Multiple Broadcast Channels in Wireless Environment. TKDE 17(3) (2005)
11. Lee, W., Zheng, B.: A Fully Distributed Spatial Index for Wireless Data Broadcast. In: ICDE 2005, pp. 417–418 (2005)
12. Lee, W., Lee, D.: Using signature techniques for information filtering in wireless and mobile environments. Distrib. Parallel Dat. 4(3), 205–227 (1996)
13. Manning, C., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press, Cambridge (1999)

14. Shivakumar, N., Venkatasubramanian, S.: Energy-Efficient Indexing For Information Dissemination In Wireless Systems. ACM, Journal of Wireless and Nomadic Application (1996)
15. Vijayalakshmi, M., Kannan, A.: A Hashing Scheme for Multi-channel Wireless Broadcast. Journal of Computing and Information Technology-CIT 16 (2008)
16. Vaidya, N., Hameed, S.: Scheduling data broadcast in asymmetric communication environments. Wireless Networks 5, 171–182 (1996)
17. Vlajic, N., Charalambous, C., Makrakis, D.: Wireless data broadcast in systems of hierarchical cellular organization. In: ICC 2003, vol. 3, pp. 1863–1869 (2003)
18. Wang, S., Chen, H.: Tmbt: An Efficient Index Allocation Method for Multi-Channel Data Broadcast. In: AINAW 2007 (2007)
19. Xu, J., Lee, W., Tang, X., Gao, Q., Li, S.: An Error-Resilient and Tunable Distributed Indexing Scheme for Wireless Data Broadcast. IEEE TKDE 18(3), 392–404 (2006)
20. Yee, W., Navathe, S.: Efficient data access to multi-channel broadcast programs. In: CIKM 2003, pp. 153–160 (2003)
21. Yang, X., Bouguettaya, A.: Broadcast-based data access in wireless environments. In: Jensen, C.S., Jeffery, K., Pokorný, J., Šaltenis, S., Hwang, J., Böhm, K., Jarke, M. (eds.) EDBT 2002. LNCS, vol. 2287, pp. 553–571. Springer, Heidelberg (2002)
22. Yao, Y., Tang, X., Lim, E., Sun, A.: An Energy-Efficient and Access Latency Optimized Indexing Scheme for Wireless Data Broadcast. IEEE TKDE 18(8), 1111–1124 (2006)
23. Zheng, B., Lee, W., Liu, P., Lee, D., Ding, X.: Tuning On-Air Signatures for Balancing Performance and Confidentiality. IEEE TKDE 21(12), 1783–1797 (2009)