Knowl Inf Syst (2014) 40:375–409
DOI 10.1007/s10115-013-0643-9

REGULAR PAPER

Evaluation and comparison of various indexing schemes in single-channel broadcast communication environment

Jiaofei Zhong · Weili Wu · Xiaofeng Gao · Yan Shi · Xiaodong Yue

Received: 27 August 2011 / Revised: 10 March 2013 / Accepted: 5 April 2013 /
Published online: 26 April 2013

Abstract *Wireless Data Broadcasting* is a newly developed data dissemination method for spreading public information to a tremendous number of mobile subscribers. *Access Latency* and *Tuning Time* are two main criteria to evaluate the performance of such system. With the help of indexing technology, clients can reduce tuning time significantly by searching indices first and turning to doze mode during waiting period. Different indexing schemes perform differently, so we can hardly compare the efficiency of different indexing schemes. In this paper, we redesigned several most popular indexing schemes for data broadcasting systems, i.e., distributed index, exponential index, hash table, and Huffman tree index. We created a unified communication model and constructed a novel evaluation strategy by using the probability theory to formulate the performance of each scheme theoretically and then conducted simulations to compare their performance by numerical experiments. This is the first work to provide scalable communication environment and accurate evaluation strategies. Our communication model can easily be modified to meet specific requirements. Our

This work was supported in part by the U.S. National Science Foundation under Grant CNS-0831579, CNS-1016320, and CCF-0829993, partially supported by Shanghai Educational Development Foundation (Chenguang Grant No. 12CG09), the Natural Science Foundation of Shanghai (Grant No. 12ZR1445000), the National Natural Science Foundation of China (Grant numbers 61202024 and 61033002).

J. Zhong (✉) · X. Yue
Department of Mathematics and Computer Science, University of Central Missouri, Warrensburg, MO 64093, USA
e-mail: zhong@ucmo.edu

W. Wu
Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

X. Gao
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
e-mail: gao-xf@cs.sjtu.edu.cn

Y. Shi
Department of Computer Science and Software Engineering, University of Wisconsin-Platteville, Platteville, WI, USA

comparison model can be used by the service providers to evaluate other indexing schemes to choose the best one for their systems.

Keywords Data broadcasting · Indexing scheme · Access latency · Tuning time

1 Introduction

Wireless Data Broadcasting becomes more and more popular in recent years because of its scalability and flexibility to disseminate public information to a mass number of mobile subscribers with common interests, since it can satisfy all pending requests of the same data in one single response. In a typical data broadcasting system, during some time periods, a group of data items (named as a *program*) are broadcasted periodically as RF radio signals by a base station within a certain area. Clients located in the valid region can access broadcasting channel, search for the required data item, wait until the data item appears, and then download it.

In practice, a number of real-world applications utilize data broadcasting techniques, where mobile clients have common interest on a certain group of data. For instance, location-based information such as local attractions, news, traffic, and weather can be broadcasted to visitors who travel to some place for the first time. In addition, wireless broadcasting service and devices by Ambient (www.ambientdevices.com), for example the 7-day weather forecaster, sports devices, as well as the Weather and Information Center Refrigerator by LG Electronics, demonstrate the industry's interest in wireless data broadcasting.

Since majority mobile devices have limited battery power and constraint lifetime, *access latency* and *tuning time* are two main criteria to evaluate the performance of a data broadcasting system. Considering a process from the moment when a client initiates a query to the moment it finishes downloading the data item, access latency denotes the whole time interval of this process, while tuning time denotes the sum of time when a client keeps "active" during the process. According to the architectural enhancements, each mobile device has two modes: *active mode* and *doze mode*. It can operate in active mode and stay idle in doze mode. Usually, the energy consumed in active mode is approximately 20–30 times higher than that in doze mode. Therefore, access latency evaluates the query response time of a system and tuning time evaluates the energy efficiency.

Indexing technologies have been introduced to reduce tuning time for a data broadcasting system. An index is a specific data structure containing the location information of data items. Due to the nature of data broadcasting, indices in data broadcasting system store the "time offset" of target data items. Once a client gets this time offset, it is aware of when the target data item will be broadcasted on the channel. Then, it can turn off to doze mode to save energy and tune in again right before the data item appears. Different indexing technologies have different searching efficiencies. Moreover, if we insert indices into data items, then the whole size of a program will increase, resulting in longer access latency. Therefore, when discussing about an indexing scheme, researchers will always consider the balance between tuning time and access latency.

A lot of traditional disk-based indexing techniques have been modified to fit the requirement of data broadcasting systems, e.g., distributed index [10], Huffman tree [11], spatial index [12], hash table [29], exponential index [27], signature tree [31], but they are constructed under different environments, which brings difficulties to compare their performance. Moreover, the same indexing technique may perform quite differently under distinguished situations. Therefore, it is desirable to construct a unified evaluation strategy to analyze the

efficiency of different indices, which will become a guide to choosing the best indexing scheme for a certain system.

Currently, according to our research, there is only one literature discussing about detailed in-depth comparisons among three indices [28], but they choose the basic indexing techniques with simple design instead of the existing state-of-the-art schemes. For instance, there are several literatures about hashing-based index for data broadcasting [9,23,29,34]. Selecting the latest optimized hashing scheme would contribute to a fair and more competitive comparison. Moreover, indices are compared under a fixed environment, which might not be easily extended to other circumstances [28]. If considering a different communication model, their discussion may have little significance. Yang and Bouguettaya [28] also made strict constraints on data items that every datum has equal size, where a flexible size model would be more practical for real-world applications.

Indexing technologies are developing very fast during recent years, and it is fair and square to choose the state-of-the-art design for comparison. To overcome the aforementioned shortcomings, we are aiming at comparing the performance of various index techniques under all possible situations. To summarize, the communication environment varies from three aspects: broadcast environment, data types, and broadcast scheduling.

- **Broadcast environment**: For a data broadcasting system, broadcast environment can be classified into two categories: single-channel broadcasting and multi-channel broadcasting. In single-channel broadcasting, data items in a program will be broadcasted through only one channel, resulting in an interleaving structure of data and index packets [9,35]. In multi-channel broadcasting, a program will be broadcasted parallel through multiple RF channels. Due to physical constraints, the available channel number is usually no more than 64 [17]. Index and data scheduling techniques are quite different in multi-channel environment.
- **Data type**: In a broadcast program, a group of data items are combined together for dissemination. Each data item can be recognized by its primary key value. Data type consists of two aspects to describe a datum: the size of a datum and the popularity of a datum. For convenience, the earliest researches assume that data items have the same size and the same access probability [21]. Later, people realize that such assumption is not practical for real-world applications. Thus, in the latest index designs, this assumption is relaxed, which allows data items having different sizes and different access probabilities, to describe the real-world information more accurately.
- **Broadcast scheduling**: Broadcast scheduling denotes the methods of how to allocate data items onto broadcasting channel, such that clients can download data more efficiently on average. There are two different broadcast scheduling methods: flat broadcast and skew broadcast. A flat broadcast means in one broadcasting cycle, each datum will be broadcasted only once and then the whole program will be repeated. In a flat broadcast, data items will repeat equal times. On the contrary, a skew broadcast means in one broadcast cycle, the most popular data items will be repeated more than once, such that clients can have more chances to download them faster. Broadcast scheduling methods also vary a lot in single-channel broadcast and multi-channel broadcast.

We strive to study the performance of all commonly used indices in all possible situations. Due to space limitation, such work will be split into a series of papers as future work. In this paper, we mainly evaluate index performance in the most basic communication environment: single-channel broadcast. We assume that data items can have different sizes and access probabilities, such that our mathematical model can be more practical and accurate. Since system performance in skew broadcast heavily relies on broadcast scheduling algorithms/designs,

but we are aiming at the performance of indices, so we only discuss flat broadcast as the first stage. In our future work, all existing situations will be discussed and analyzed accordingly.

In this paper, we mainly choose four types of indexing techniques for further redesign, evaluation and comparison, i.e., distributed index, exponential index, hash table, and Huffman tree index, which are among the most commonly used indices for existing data broadcasting systems. To fairly evaluate different indices, we follow the latest and most efficient indexing designs. We also redesign and modify some indices to ensure that they are applicable in our communication model. The same group of data items will be used to test the efficiency of the schemes. All of these work is trying to make sure that every design is discussed under a unified environment. Otherwise, the comparison will become meaningless. Next, we construct an accurate formulation to evaluate the performance of each indexing scheme, with the help of probability theory. Such idea can be easily extended to other index technique besides the four we mentioned in this paper. We provide more detailed theoretical analysis, with which we hope to help service providers to choose among various indexing schemes. Finally, we simulate the broadcasting environment and provide extended numerical experiments. The results of our simulations prove the system performance fairly and clearly.

The rest of this paper is organized as follows: in Sect. 2, we study recent literatures for wireless data broadcasting problem, including various indexing technologies in different communication environments. In Sect. 3, we illustrate our system model, discuss broadcast environment, data type, and bucket structures in detail. In Sects. 4, 5, 6, and 7, we describe the construction and evaluation of distributed index, exponential index, hash table, and Huffman tree index, respectively. Next, in Sect. 8, we illuminate the process of simulation and discuss index performance based on our numerical experiments. Section 9 further explores the advantages and disadvantages of various indexing schemes. Finally, Sect. 10 gives conclusion and the plan of our next stage work.

2 Related works

In wireless data broadcasting, main research topics always focus on how to design index structures and how to allocate data onto channels. The purpose is to reduce access latency [25] and tuning time, in order to improve the system performance and efficiencies [8,32].

2.1 Traditional schemes

A lot of 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 non-uniform access distribution. Vaidya and Hameed [22] discussed optimization issue with respect to the average access latency when data access distribution is non-uniform. Vlajic et al. [24] presented an optimized data broadcasting strategy in hierarchical cellular organization system. However, none of them implements indexing technique. Moreover, without doze mode, the tuning time is as long as access latency, which causes high power consumption of mobile devices.

2.2 Indexing schemes

There are also many works converting traditional disk-based indexing approaches to air indexing by converting physical address into time offset. Figure 1 illustrates the classification

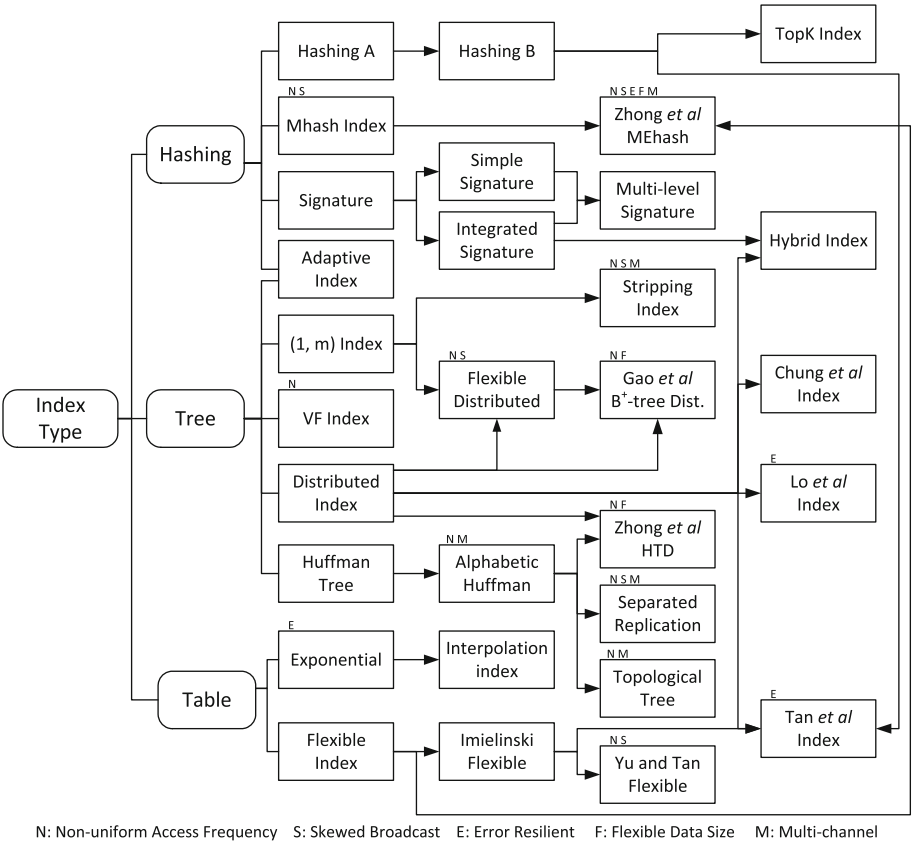


Fig. 1 Classification of some existing indexing schemes

of some of the major existing indexing schemes, with specific parameters indicating their characteristics and features [18]. In general, indexing schemes can be classified into three categories, i.e., hashing, tree, and table-based schemes.

2.2.1 Hashing schemes

Hashing-based schemes utilize hash functions and store index information within data buckets. Imielinski et al. [9] presented two hashing protocols, i.e., Hashing A and Hashing B. The former protocol calculates $h(K)$ and then follows the *Shift* value to find data, the latter one applied a minor modification of the hashing function to improve performance. Later, Yao et al. [29] proposed *MHash*, which considers a two-argument hash function $H(k, l)$ to map each data to a number of slots, thus facilitates skewed access probabilities and reduces access latency. Recently, Zhong et al. [34] further extended *MHash* and developed a multi-channel energy-efficient hashing scheme called *MEHash*. Later, they refined the scheme and proposed *HAMHash* [33]. By applying several hash functions to allocate data onto multiple channels, facilitating skewed broadcast according to non-uniform data access frequencies and allowing flexible number of data replications by introducing adjustable parameters, *HAMHash* achieves almost optimal tuning time and energy efficiency.

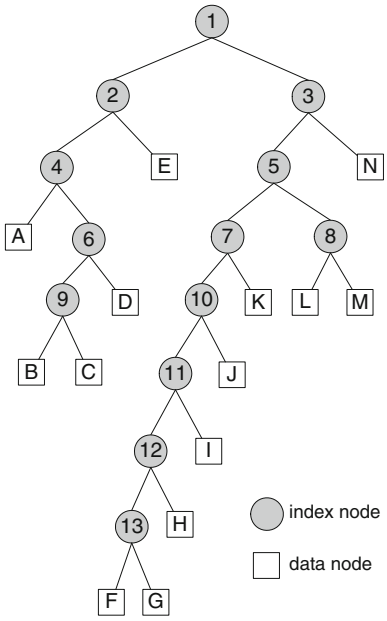


Fig. 2 An example of alphabetic Huffman tree

2.2.2 *Tree-based schemes*

Imielinski et al. proposed $(1, m)$ index [10], which broadcasts the index part m times in front of each fraction of the data file. They also customized distributed index [10], which divides the index tree such as B^+-tree into replicated part and non-replicated part. B^+-tree distributed index (*BTD*) was extended by many other researchers to satisfy different system requirements. One work [26] proposed an index allocation method named TMBT for multi-channel data broadcasting, which creates a virtual *BTD* for each data channel and multiplexes them on the index channel. Hsu et al. [5] modified *BTD* to deal with non-uniform data access frequencies. Gao et al. [4] built a complete multi-channel broadcasting system based on the variation of *BTD* for data set with non-uniform access probabilities and unequal data sizes. In addition, one paper [13] discussed a signature-based approach for information filtering, where the binary hashing code of each datum (as signature) forms a tree to assist searching, which may not perform well under non-uniform access probabilities. Later, Hu et al. [6] designed a hybrid indexing scheme combining *BTD* and signature-based index. One problem of signature scheme is that false drops may occur, where clients download the wrong data records with identical signatures.

Huffman tree is a skewed index tree which takes into account the data access probabilities, where more popular data have a shorter path from the root of the tree, thus the average tuning time is minimized [3,20]. The construction of Huffman tree [20] is similar to Huffman code construction, but it has a problem that the clients may fail to find desired data by traversing that Huffman tree. The other algorithm for constructing skewed Huffman tree [3] has the same problem. There is another kind of Huffman tree, *Alphabetic Huffman Tree* [7], which serves as a binary search tree as illustrated in Fig. 2. Additional examples of Huffman tree and alphabetic Huffman tree can be found in Figs. 12 and 13 of Sect. 7. It is further extended to k-ary search tree [20], so that a tree node will fit in any size wireless packet by adjusting the fan-out of the tree. However, most of the above works discussed their own proposed

Huffman tree on a specific type of data set with special constraints and features under multi-channel environment. Later, Zhong et al. [35] proposed an alphabetic Huffman tree distributed indexing scheme, which minimizes both average access latency and average tuning time, and outperforms the B^+-tree distributed indexing scheme. In addition, Lu et al. [14] designed a scalable and efficient tree-based mechanical scheme for multi-channel broadcast, which is named SETMES.

2.2.3 Table-based schemes

Imielinski et al. [9] presented the flexible index, which divides data file into a flexible number of segments according to one adjustable parameter, and stores indices in the tables within data segments. Another work by Xu et al. [27] gave an idea of exponential index that shares links in different search tables, which allows clients to start searching at an arbitrary index node. However, both approaches may not perform well under non-uniform access probabilities.

2.3 Multi-channel environment

Besides the fact that a large number of indexing schemes are developed under single-channel broadcast environment, multi-channel data broadcasting is also an important field in the literatures, which can be categorized into two types according to the allocation methods: interleaved and non-interleaved broadcasting. The former type means index and data can be interleaved (appear alternately) on each available channel, while the latter means index and data should be assigned to different channels, i.e., either an index channel or a data channel.

When it comes to multi-channel data broadcasting, how to allocate index and data will impose significant impact on the performance of each indexing technique. Multi-channel broadcasting might be similar to single-channel broadcasting in the way that a single-channel system could be considered as part of an interleaved multi-channel broadcasting system, where the index part can serve as local index on its channel, and additional global index might be applied to connect existing channels in the system. Several works [2,30,33] deal with data allocation for multi-channel data broadcasting. However, a certain allocation method can be helpful to a specific index structure, meanwhile reducing the efficiency of another scheme.

2.4 Our contribution

In this paper, we aim at comparing several commonly used indexing approaches under the same conditions, as well as improving them to minimize both average access latency and average tuning time. As a result, we adopt single-channel data broadcast environment to avoid all kinds of influences introduced by a multiplicity of multi-channel allocation methods. We are the first work to compare several popular indexing schemes on a unified data broadcasting system.

Our contribution includes three aspects. First, we construct a unified communication environment for wireless data broadcasting system and provide structured design of four indexing techniques: distributed index, exponential index, hash table, and Huffman tree index. We follow the inspiration of latest and most efficient construction for each indexing scheme and redesign or modify them such that they can be applied in the unified communication environment with higher efficiency. Specifically, we redesign the pointers and bucket structures for B^+-tree index and consider different bucket sizes between index and data buckets. For hash scheme, we redesign the allocation method and provide more details of the bucket structure for extensive data set. Exponential index is redesigned to handle variable lengths

of data, with rearranged control tables and redefined chunk structure. Huffman tree index is adapted to single-channel broadcast with distributed method implemented. Second, we provide general theoretical analysis to evaluate the performance of each index. Such analysis can be applied easily to major indices used in data broadcasting. It can become a reference to evaluate the efficiency of an indexing technique. Finally, we simulate data broadcasting system with plenty of numerical experiments, using the same group of sample data, such that the output will be comparable and reliable. Our results can guide service providers to choose an appropriate indexing scheme for their own system.

3 System model and bucket structure

In this section, we present our novel unified system model and the detailed design of bucket structures for the indexing strategies in wireless data broadcasting.

3.1 System symbols

In our system, the data set to be broadcasted is D, where the number of data items is t and $D = \{d_1, d_2, \ldots, d_t\}$. We assume that data items in D are arranged in a consecutively increasing order on their primary key values. The access frequency or 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. Furthermore, data items may have different sizes due to various applications, so we introduce *bucket* to measure the size of each data item. A *bucket* is the minimum logical unit used for data transmission in wireless data broadcasting system. We assume s_i is the number of buckets that d_i occupies, which can be considered as the "length" or "size" of d_i on the axis of time, and S denotes the length set of D. The base station broadcasts data set D on the wireless broadcasting channel. The clients within the broadcasting region can generate queries on client side and then tune into broadcasting channel to search and download the target data, by following the indices (pointers) to find the target bucket, without implementing complex retrieval methods. More detailed client-side retrieval algorithms can be found in [15,19,34].

In order to reduce tuning time, some tree-based indexing strategies, for instance the B^+-tree Index, are applied to the wireless data broadcasting system. We use T to denote the index tree for tree-based indexing strategies and define k as the maximum number of branches for each node in T. L is the depth or height of T. When it comes to the *distributed index* [10], T will be "cut" at the lth level. A *bcast* means one broadcast sequence on a channel. Table 1 lists most of the symbols used in this paper. Some other symbols and detailed design for different indexing strategies will be illustrated later.

3.2 Bucket and pointer

Data bucket and index bucket have different structures and sizes. Generally, a bucket has two segments, named *head* segment and *payload* segment. The head segment has the following elements:

bId: The id of bucket, used for recognition, is in the format of (i, j, n), which implies the nth recurrence of index B_i^j, or data d_i^j with size of n buckets.
bType: The type of this bucket. For example, *BTD* indexing strategy has three types of buckets, i.e., control index, search index, and data bucket.
bLength: The total length of this bucket, measured in terms of time unit.

Table 1 Symbol description

Symbols	Description	Symbols	Description
D	Data set $D = \{d_1, \ldots, d_t\}$	L	Level of T
P	Probability set $P = \{p_1, \ldots, p_t\}$	l	Threshold to cut T
S	Length set $S = \{s_1, \ldots, s_t\}$	k	Maximum branch number for T
t	Number of data items	$bcast$	One broadcast sequence on a channel
T	An index tree	$\mathbb{B}_i$	The ith block on $bcast$
B_i^j	The jth index at ith level of T	d_i^j	The jth bucket of data item d_i
$\triangle_i$	The ith subtree of level l+1 on T	$\text{MAX}(B_i^j)$	Maximum key value that B_i^j domains
R	Total number of $\triangle_i$ on T	$\text{PATH}(B_i^j)$	A path from B_1^1 to B_i^j
V_i	Distributed path for $\triangle_i$	v_i	Length of V_i on $\mathbb{B}_i$ with average v
D_i	Data block on $\mathbb{B}_i$	u_i	Length of index on $\mathbb{B}_i$ with average u
P_i	Probability for block $\mathbb{B}_i$	x_i	Length of D_i on $\mathbb{B}_i$ with average x
$\lvert \cdot \rvert$	Cardinality of one set	$\lVert \cdot \rVert$	Length measured in data bucket unit
C	Chunk set	Z	Index table size in Exponential index
I	Number of data items in C	$\tau(l)$	Avg. number of visited index buckets
b	First index of hole-free sequence	$H(k)$	Hash function
θ	Zipf distribution parameter	$Dis(k)$	Displacement area

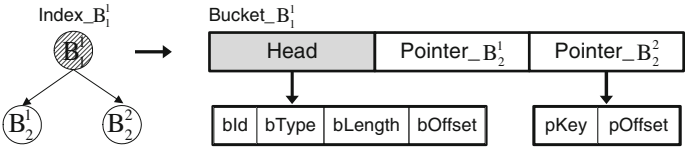


Fig. 3 An example of index bucket structure

bOffset: The offset to the next nearest index. For instance, in *BTD*, it may be the offset to the next control index.

Now, it comes to the *payload* segment. If the bucket is a data bucket, then the payload segment stores the datum. Note that a datum may take up several data buckets, while each data bucket has the same maximum length. On the other hand, if the bucket is an index bucket, then the payload segment stores index information, such as pointers, which indicate the locations of its children on time axis. In our paper, a pointer contains the following elements:

pKey: The *bId* of the index or data bucket it points to, used by clients to find searching direction.
pOffset: An offset from current moment, which allows clients to sleep for "offset" time and tune in again at the moment when target bucket appears.

For tree-based indexing strategies, an index bucket may contain several pointers, corresponding to the design of index tree. Figure 3 illustrates an 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 has two pointers, recording the locations of B_2^1 and B_2^2.

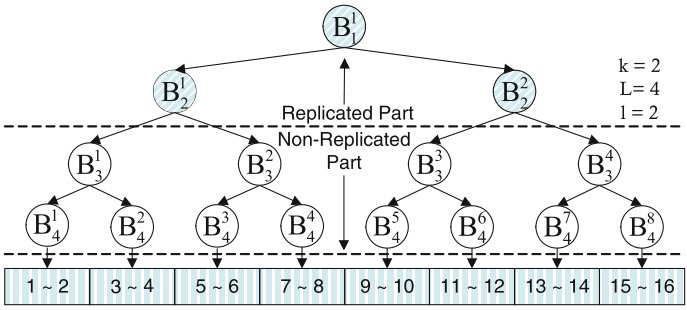


Fig. 4 An example of B^+-tree cut at the 2nd level

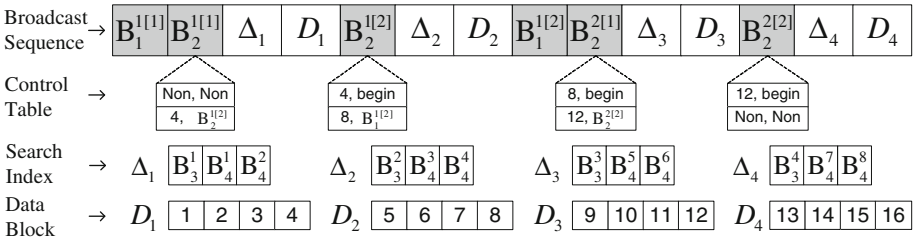


Fig. 5 An example of *bcast* with control Tables

4 B$^+$-tree-based distributed index

In the first place, we present the B$^+$-tree distributed indexing strategy in detail, which is developed based on the observation that a data set would suit real-world applications much better if considering non-uniform data access patterns and unequal data sizes. We take the advantages of both distributed index and B$^+$-tree index [4], with further modifications on the broadcasting strategy and the structures of pointers and buckets, in order to make it more practical, efficient, and realistic, under our unified model for better evaluations.

Here, index and data are interleaved on the same broadcasting channel. According to B$^+$-tree-based distributed index, we consider *depth-first* index layouts and "cut" index tree at level l. Thus, nodes from level 1 to level l are in the *replicated part*, while other index nodes are in the *non-replicated part*. Furthermore, we append *control indices* on those indices within the replicated part, to make the searching process more efficient. An example of a full binary B$^+$-tree index structure is presented in Fig. 4, which shows a distributed index tree with the maximum branch number $k = 2$, total number of levels $L = 4$, and cutting level $l = 2$. Each index node B_i^j represents the jth index node on the ith level of the tree. All the index nodes above (including) the cutting level of the tree are called *control indices*, while the other index nodes below are called *search indices*.

Next, we traverse T according to distributed rules and then append *control table* for each control index. Figure 5 shows an example of the broadcast sequence *bcast* for the aforementioned index tree example in Fig. 4, where number of data items $t = 16$, maximum branch number $k = 2$, cutting level $l = 2$, and total number of levels $L = 4$. There are 18 indices in *bcast*, 6 of which are control indices and the rest 12 are search indices. $\triangle_i$ denotes the ith subtree in the non-replicated part, which only consists of search indices. For instance, $\triangle_2$ is the subtree rooted at B_3^2, with two children B_4^3, B_4^4.

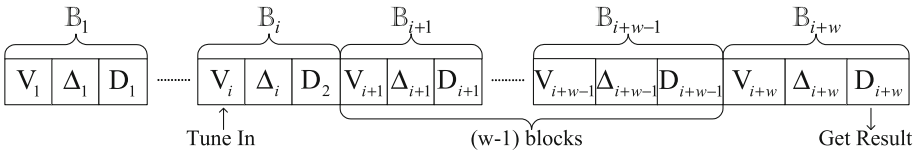


Fig. 6 An example of a client searching for data

Besides, PATH(B_i^j) is a path from root B_1^1 to node B_i^j (excluding the end point), and V_i is a *distributed path* before each $\triangle_i$. For example, from Fig. 4, we can see that the distributed path for B_3^3 should be $V_3 = \{B_1^1, B_2^2\}$. The broadcast sequence is defined as $bcast = \{V_1, \text{DFT}(\triangle_1), D_1, V_2, \text{DFT}(\triangle_2), D_2, \ldots, V_R, \text{DFT}(\triangle_R), D_R\}$. Furthermore, an index bucket may have different size compared to a data bucket, so we define "r" to indicate the ratio of data bucket size to index bucket size, i.e.,

$$\frac{1}{r} = \frac{\text{index bucket size}}{\text{data bucket size}}$$

We use $|bcast|$ to represent the cardinality of set $bcast$ and $\|bcast\|$ to indicate the total length of $bcast$, measured in the unit of data bucket. Moreover, we use DFT($\triangle_i$) to represent the depth-first traversal for $\triangle_i$ and use $B_i^{j[1]}, \ldots, B_i^{j[k]}$ to represent k occurrences of B_i^j, where k is identical to the branch number k of T. Next, control tables are appended onto control index, following the approaches step by step as introduced in [4]. Finally, all the control tables are successfully built as shown in Fig. 5.

4.1 Performance analysis of B$^+$-tree-based distributed index

In this section, we evaluate the system performance of B$^+$-tree-based distributed index by means of analyzing *access latency* and *tuning time*.

First, let us consider access latency, where all index and data buckets are interleaved on one broadcast channel. The whole $bcast$ is divided into $\mathbb{B}_1, \ldots, \mathbb{B}_R$ blocks, where $\mathbb{B}_i = \{V_i, \text{DFT}(\triangle_i), D_i\}$, for $1 \le i \le R$. We use P_i to represent the access probability of block $\mathbb{B}_i$, where P_i can be derived by summing up the probabilities of all data buckets that belong to data block D_i of $\mathbb{B}_i$, i.e., $P_i = \sum_{j \in D_i} p_j$, $for\ i = 1, \ldots, R$. Let v denote the average length of V_i, u indicates the average length of $V_i + \triangle_i$, and x symbolizes the average length of D_i. Therefore, we have $u = \frac{|bcast| - |D|}{rR}$, $v = \frac{\sum_{i=1}^{R} |V_i|}{rR}$, and $x = |D|/R$.

Theorem 4.1 *If distributed indices and data are interleaved on one broadcast channel, then the average access latency for B$^+$-tree-based distributed index is:*

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

Proof First, a client 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 the required datum d_j at $\mathbb{B}_{i+w}$. Second, 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. 6. Hence, according to the law of total expectation, we have the above conclusion. Next, we will present detailed analysis of each step.

– *Case 1:* $1 \leq w < R-1$. We can divide this case into three phases: 1) the client tunes into block $\mathbb{B}_i$ and takes an average $\frac{u+x}{2}$ time in it; 2) it waits through $(w-1)$ complete blocks, which takes $(w-1)(u+x)$ time; and 3) it finds the pointer to the datum, which only exists in $\triangle_{i+w}$, and then downloads the data, so the average waiting time is $u+\frac{x}{2}$. The mean of this period is:

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

– *Case 2:* $w=0$. The client tunes into V_i of block $\mathbb{B}_i$, and the pointer to required data is indeed in the following $\triangle_i$ of the same block $\mathbb{B}_i$. In this case, it only has aforementioned phases 1) and 3), so its mean becomes:

$$E(AL|b=i,d=0)=\frac{v}{2}+u-v+\frac{x}{2}=u-\frac{v}{2}+\frac{x}{2} \tag{3}$$

– *Case 3:* $w=R-1$. Suppose the client tunes into block $\mathbb{B}_i$ and the required data are just 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 broadcasted again in the next *bcast*. The mean of the waiting time is:

$$E(AL|b=i,d=R-1)=\left(\frac{1}{2}+w\right)u+wx \tag{4}$$

Therefore, considering Eqs. (2), (3), (4), and the law of total expectation, we can derive the average access latency as follows:

$$\begin{aligned}
E(AL) &= \sum_{i=1}^{R}\sum_{w=0}^{R-1} E(AL|b=i,d=w)\cdot P(b=i,d=w)\\
&= \sum_{i=1}^{R}\Bigg(\sum_{w=1}^{R-2} E(AL|b=i,d=w)P(b=i,d=w) + E(AL|b=i,d=0)\\
&\quad \cdot P(b=i,d=0) + E(AL|b=i,d=R-1)P(b=i,d=R-1)\Bigg)\\
&= \sum_{i=1}^{R}\Bigg(\sum_{w=1}^{R-2}\left(\left(\frac{1}{2}+w\right)u+wx\right)\frac{P_{(i+w)\%R}}{R} + \left(u-\frac{v}{2}+\frac{x}{2}\right)\frac{P_i}{R}\frac{v}{u+x}\\
&\quad + \left(\left(\frac{1}{2}+w\right)u+wx\right)\frac{P_i}{R}\cdot\frac{u-v+x}{u+x}\Bigg)\\
&= \frac{1}{R}\sum_{i=1}^{R}\Bigg(\sum_{w=1}^{R-2}\left(\left(\frac{1}{2}+w\right)u+wx\right)P_{(i+w)\%R} + \left(u-\frac{v}{2}+\frac{x}{2}\right)\frac{vP_i}{u+x}\\
&\quad + \left(\left(\frac{1}{2}+w\right)u+wx\right)P_i\cdot\frac{u-v+x}{u+x}\Bigg)
\end{aligned}$$

□

Next, we evaluate the computation of average tuning time for B$^+$-tree-based distributed index.

Theorem 4.2 *The average tuning time for B^+-tree-based distributed index is*

$$E(TT) = \sum_{i=1}^{R} \frac{3u_i - v_i + (2+r)x_i}{r\,(uR + |D|)} + \frac{2L - l}{2r} + \sum_{i=1}^{|D|} s_i\, p_i \tag{5}$$

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

Phase 1 The client tunes into broadcast channel and searches for the right *control index*, following which it can get the required data on that block. We analyze this phase by considering three cases.

- *Case 1: The client first tunes into a control index*. Then, the client can follow the control table to find the right control index in one more step, which is discussed in [4]. The probability of this case is $\sum_{i=1}^{R} \frac{v_i}{uR+|D|}$, and the average tuning time of this case is $\frac{2}{r} \sum_{i=1}^{R} \frac{v_i}{uR+|D|}$.
- *Case 2: The first visited bucket is a search index*. The client may 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 $\sum_{i=1}^{R} \frac{u_i - v_i}{uR+|D|}$, and average tuning time is $\frac{3}{r} \sum_{i=1}^{R} \frac{u_i - v_i}{uR+|D|}$.
- *Case 3: The first visited bucket is a data bucket.* The client also needs to wait for the next nearest control index and then goes to the target control index, with a probability of $\sum_{i=1}^{R} \frac{x_i}{uR+|D|}$. The average tuning time is $(1 + \frac{2}{r}) \sum_{i=1}^{R} \frac{x_i}{uR+|D|}$.

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

Phase 3 The client sleeps until the requested data arrive and then tunes in again to download data. The average downloading time is $\sum_{i=1}^{|D|} s_i\, p_i$.

Finally, by summarizing the above steps, we obtain the average tuning time:

$$\begin{aligned} E(TT) &= \frac{2\sum_{i=1}^{R} v_i}{r(uR + |D|)} + \frac{3\sum_{i=1}^{R}(u_i - v_i)}{r(uR + |D|)} + \frac{(2+r)\sum_{i=1}^{R} x_i}{r(uR + |D|)} + \frac{2L - l}{2r} + \sum_{i=1}^{|D|} s_i\, p_i \\ &= \sum_{i=1}^{R} \frac{3u_i - v_i + (2+r)x_i}{r\,(uR + |D|)} + \frac{2L - l}{2r} + \sum_{i=1}^{|D|} s_i\, p_i \end{aligned}$$

□

After analyzing average access latency and average tuning time, now we need to know the value of L, R, $|bcast|$, $|\triangle_i|$, u and v. The total level L of an index tree is determined by the number of branches k of T and S 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, then 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. $N(L)$ is defined as the set of nodes at the Lth level of T. Algorithm 1 shows how to compute L and $|N(L)|$, with which we can get $R = |N(l+1)|$.

5 Exponential index

In the second place, we propose our *exponential index* strategy, based on the idea of generalized exponential index introduced in [27], with extended details of the bucket structure

Algorithm 1 ***Compute L***

Input: t, k
Output: L, $|N(i)|$ $(1 \leq i \leq L)$.
1: $L = 1$; $ns = \lceil t/k \rceil$;
2: **while** $ns \neq 1$ **do**
3: $\quad |N(L)| = ns$; $L = L + 1$; $ns = \lceil ns/k \rceil$;
4: **end while**
5: $|N(L)| = ns$; reverse $N(i)$, $(1 \leq i \leq L)$;

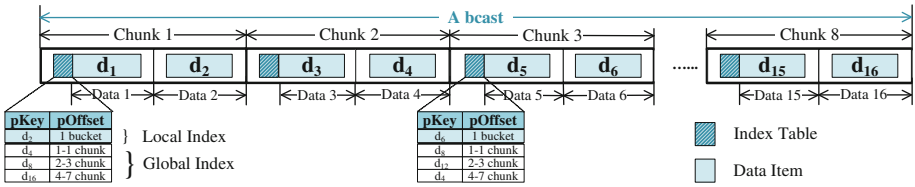


Fig. 7 An example of exponential indexing strategy

design, and refinement to suit our unified model in order to achieve higher efficiency and better evaluations.

Some differences between our exponential indexing strategy and that in [27] are summarized as follows: 1) Data items have unequal lengths, i.e., each datum may take up several data buckets, which is a more realistic assumption under real-world scenarios. 2) For each chunk, we use an independent index bucket to store the index table. 3) We allocate the index bucket at the beginning of a chunk, before all the data buckets of that chunk. 4) We change the number of entries in local index to be I. 5) In an index table, the local index entries appear before the global index entries. 6) The structures of index bucket and data bucket are redesigned and illustrated with more details. One of the main features of exponential index is that it is error resilient, so that it can be easily applied to the broadcasting environment with link errors.

Exponential index is very efficient in that it shares links in different search trees and thus minimizes storage overhead. Also, it has a linear and distributed structure, which allows searching to begin from any index as well as recovering from link errors quickly. Here is an example in Fig. 7, the server maintains 16 data items that are arranged in a *bcast* in ascending order of their key values. We assume that data items have different sizes, so each of them may take up several data buckets. We assume each chunk contains an index table, and a data part containing I data items, here $I = 2$ in this example. The index table here consists of four entries (rows), including one entry of *local index* and three entries of *global index*. Each entry indexes a segment of buckets in the form of a tuple as {*pKey, pOffset*}, where *pKey* is the maximum key value of this range of buckets, and *pOffset* specifies the distance to the beginning of this range from the current position (denoted as buckets or chunks). The sizes of the segments in one table grow exponentially. For global indices, the first entry describes a single bucket segment (i.e., the next bucket), and for each $i > 1$, the ith entry describes the segment of buckets that are 2^{i-1} to $2^i - 1$ away (i.e., 2^{i-1} chunks away).

5.1 Performance analysis of exponential index

In this section, we evaluate the system performance of exponential index by means of analyzing *access latency* and *tuning time*.

First, let us consider access latency, where all index buckets and data buckets are interleaved on one broadcast channel. We use C to denote the chunk set, and Z to indicate the size of one index table. The whole *bcast* is divided into $C_1, \ldots, C_{|C|}$ blocks, where $|C_i| = \sum_{j=(i-1)I+1}^{\min(iI,t)} s_j + Z$, for $1 \leq i \leq |C|$, and the number of chunks $|C| = \lceil t/I \rceil$. We use P_i to represent the access probabilities for chunk C_i, while P_i can be derived by summing up the probabilities of all data buckets that belong to chunk C_i, i.e., $P_i = \sum_{k=(i-1)\cdot I+1}^{\min(i\cdot I,t)} p_j, for\ i = 1, \ldots, |C|$. Also, we denote the probability of tuning in the ith chunk as P_i', which is equal to $|C_i|/|bcast|$. Furthermore, since the size of index bucket might be different from that of each data bucket, so we also use the aforementioned parameter "r" to indicate the ratio of data bucket size to index bucket size. Recall that $1/r = index\ bucket\ size/data\ bucket\ size$.

Theorem 5.1 *If exponential indices and data items are interleaved on one broadcast channel, then the average access latency for exponential index is*

$$E(AL) = \sum_{i=1}^{|C|} \left(\sum_{j=i+1}^{|C|} \left(\frac{|C_i|}{2} + \sum_{k=i+1}^{j-1} |C_k| + \frac{|C_j|}{2} \right) \cdot P_j + \left(\frac{Z}{2} + \frac{|C_i| - Z}{|C_i|} \cdot |bcast| \right) \cdot P_j + \sum_{j=1}^{i-1} \left(|bcast| - \sum_{k=j}^{i} |C_k| + \frac{|C_i|}{2} + \frac{|C_j|}{2} \right) \cdot P_j \right) \times \frac{|C_i|}{|bcast|} \tag{6}$$

Proof First, a client tunes into the broadcast channel at chunk C_i. Then, it waits for the farthest chunk it can reach which precedes the target chunk containing the required datum. Second, the client may hop several times to repeat this process until it gets the target chunk C_j. Eventually, the client waits for the first data bucket of the required data and then downloads all the data buckets of these data. Hence, according to the law of total expectation, we have the above conclusion. Next, we will present detailed analysis of each step.

- *Case 1:* $i \leqslant j - 1$. We can divide this case into three phases: 1) the client tunes into chunk C_i, which takes an average $\frac{|C_i|}{2}$ time in it; 2) the client waits through k complete blocks, which takes $\sum_{k=i+1}^{j-1} |C_k|$ time; and 3) it finds the required data in chunk C_j and then downloads it, so the average waiting time is $\frac{|C_j|}{2}$. Here, the access probabilities P_j for chunk C_j is $\sum_{k=(j-1)I+1}^{j\times I} P_k$. The mean of the access latency during this period is:

$$E(AL_1) = \left(\frac{|C_i|}{2} + \sum_{k=i+1}^{j-1} |C_k| + \frac{|C_j|}{2} \right) \cdot P_j \tag{7}$$

- *Case 2:* $i = j$. Suppose the client tunes into broadcast channel at chunk C_i and luckily gets the index table. Fortunately, the request datum is in the same chunk C_i, so after checking the local index, client can find the request data and download it in the same chunk. Considering the probability, the mean of the access latency during this period is:

$$\begin{aligned} E(AL_2) &= \left(\frac{Z}{|C_i|} \times \frac{|C_i|}{2} + \frac{|C_i| - Z}{|C_i|} \times \left(|bcast| - |C_i| + \frac{|C_i|}{2} + \frac{|C_i|}{2} \right) \right) \cdot P_j \\ &= \left(\frac{Z}{2} + \frac{|C_i| - Z}{|C_i|} \times |bcast| \right) \cdot P_j \end{aligned} \tag{8}$$

– *Case 3: $i > j$*. In this case, the target chunk C_j locates in front of chunk C_i where the client tunes in, so the client needs to wait until the next occurrence of chunk C_j in the next *bcast*. Just like the above cases, we can derive the mean of the period in this case as:

$$E(AL_3) = \left(|bcast| - \sum_{k=j}^{i} |C_k| + \frac{|C_i|}{2} + \frac{|C_j|}{2} \right) \cdot P_j \tag{9}$$

Therefore, considering Eqs. (7), (8), (9), and the law of total expectation, we can conclude the average access latency as follows:

$$\begin{aligned} E(AL) &= \sum_{i=1}^{|C|} \left(\sum_{j=i+1}^{|C|} E(AL_1) + E(AL_2) + \sum_{j=1}^{i-1} E(AL_3) \right) \times P'_i \\ &= \sum_{i=1}^{|C|} \left(\sum_{j=i+1}^{|C|} (\frac{|C_i|}{2} + \sum_{k=i+1}^{j-1} |C_k| + \frac{|C_j|}{2}) \cdot P_j + \left(\frac{Z}{2} + \frac{|C_i| - Z}{|C_i|} \cdot |bcast| \right) \cdot P_j \right. \\ &\quad \left. + \sum_{j=1}^{i-1} \left(|bcast| - \sum_{k=j}^{i} |C_k| + \frac{|C_i|}{2} + \frac{|C_j|}{2} \right) \cdot P_j \right) \times \frac{|C_i|}{|bcast|} \end{aligned} \tag{10}$$

□

Next, we examine the average tuning time for exponential index.

Theorem 5.2 *The average tuning time for exponential index is*

$$E(TT) = \sum_{i=1}^{t} \sum_{l=0}^{|C|-1} \left[\left(1 + \frac{1}{r} \right) \frac{\Sigma s_i}{|bcast|} + \frac{1}{r} \cdot \frac{Z \cdot |C|}{|bcast|} + \tau(l) + s_i \right] \cdot p_i \tag{11}$$

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

Phase 1 The client tunes into broadcast channel and searches for the first index table, which is known as *initial probe*.

– *Case 1: The client tunes into an index bucket*. Then, the average tuning time of its initial probe is $\frac{1}{r}$.
– *Case 2: The first visited bucket is a data bucket*. The client may need to wait for the next nearest index bucket, so the average tuning time is $1 + \frac{1}{r}$.

Phase 2 After that, the client searches for the target index bucket that directly points to the required data in the same chunk. Suppose it is l chunks away. We define the average number of visited index buckets in this step as $\tau(l)$.

$$\tau(l) = \begin{cases} 0 & \text{if } l = 0 \\ \tau(l - x) + 1 & \text{if } l > 0 \end{cases} \tag{12}$$

Here, x is the maximum value less than or equal to l in the set of $\{1, 2, \lfloor a + 2 \rfloor, \ldots, \lfloor \frac{a^{n_g} - 1}{a - 1} \rfloor + 1\}$, where a is the base value for exponential index, and n_g is the number of global index entries in each index table.

Phase 3 The client sleeps until the required data appear and then tunes in again to download data, thus an additional tuning time of s_i is required.

Finally, summarizing the above steps, we can get the average tuning time

$$E(TT) = \sum_{i=1}^{t} \sum_{l=0}^{|C|-1} \left[\left(1 + \frac{1}{r}\right) \frac{\Sigma s_i}{|bcast|} + \frac{1}{r} \cdot \frac{Z \cdot |C|}{|bcast|} + \tau(l) + s_i \right] \cdot p_i \qquad (13)$$

□

6 Hash scheme

In the third place, we present the novel *hash scheme*-based broadcasting strategy. Hashing is a well-known data access approach for traditional database systems. Nowadays, it is also implemented in wireless data broadcasting environment. In this paper, we introduce an energy-efficient hash scheme, which stores hash parameters in head segment of data buckets, functioning as index without introducing additional index buckets. We take advantages of the indexing scheme called *MHash* introduced in [29], and the hash functions applied in [23], as well as the idea of *Hashing B* protocol in [9], and then improve them by adapting the scheme to more extensive data items with different sizes, and further extend them with more details of the broadcast structure design, in order to achieve unified conditions for better evaluations.

In our proposed hash scheme, a broadcast cycle consists of a sequence of data buckets, each of which contains head segment and data segment. There is no index bucket in a broadcast cycle, while the hash parameters are stored in the head segment of each bucket. In addition, the head segment contains the bucket ID denoted as *bId*, data item key denoted as *bKey*, *Hash Functions*, and *Global Pointer*. Buckets in the broadcast cycle are numbered as $1, 2, \ldots, \|D\|$. Data items may have different lengths, thus they may take up different number of buckets. Therefore, the *cycle length* or total number of buckets in one *bcast* is $\|D\| = \sum_{i=1}^{|D|} s_i$. Figure 8 shows the specific bucket structure in our hash scheme.

6.1 Hash scheme for simple data set

First of all, let us see how to allocate simple data set onto broadcast channel, where data items have the same size of one data bucket. First, we use a hash function $H(key)$ as below, to map all the data items onto corresponding buckets:

$$H(key) = [(A * key + B) \mod 2^{31}] \,\%\, |bcast| + 1, \qquad (14)$$

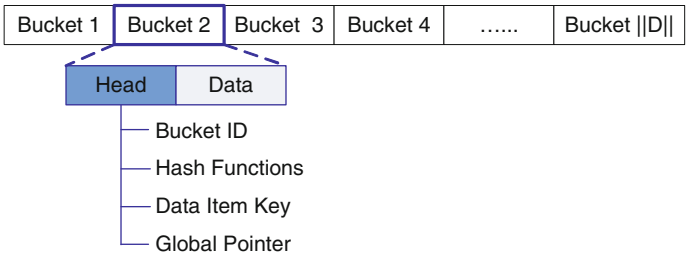


Fig. 8 The structure of buckets in hash scheme indexing strategy

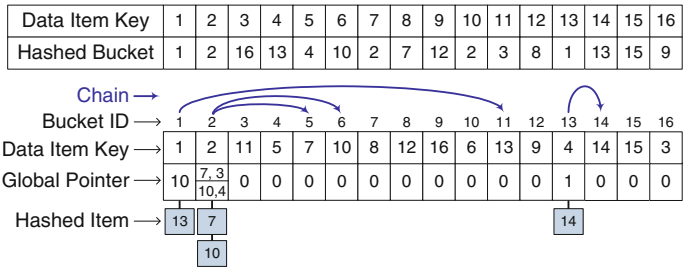


Fig. 9 The chaining method of hash scheme for simple data set

where A = 1103515245, B = 12345, $|bcast|$ means the total number of buckets or length of broadcast cycle, and key is the key value of that data item [23,29]. After that, we apply chaining method to resolve collisions. If multiple items are hashed to the same bucket, we link them into a chain by decreasing order of their access probabilities: the first item with the highest access probability would be stored in this hashed bucket, while others are sequentially placed in the following empty buckets. We break the tie between items having the same probability by selecting the item with smaller key first.

We assign *Global Pointer* on each bucket, which records an offset or a control table of (*key*, *offset*) pairs that refer to the chaining bucket(s) storing the item(s) on the chain. For example, in Fig. 9, data items 1 to 16 are in decreasing order of access probability, while data 2, 7, 10 are hashed to the same bucket 2. After allocating all the other data into their hashed buckets, we place data 7 at the first empty bucket 5 and data 10 at the second empty bucket 6, and then assign the global pointer on bucket 2 for them. Next, in order to create a hole-free broadcast cycle, we apply the second hash function $H'(key)$:

$$H'(key) = (H(key) + |bcast| - b) \mod |bcast| + 1, \tag{15}$$

where b is the smallest index such that buckets b to $|bcast|$ are hole-free under H [29]. The main purpose of second hash is to eliminate the empty buckets or holes that appear in the first broadcast cycle.

6.2 Hash scheme for extended data set with different sizes

Next, let us see how to allocate extended data set which has different item sizes instead of equal size onto broadcast channel. Given a set of data items and their sizes, we first apply hash function $H(key)$ according to (14), where $|bcast|$ is set to be the total number of data buckets denoted as $\|D\|$. And then, we apply the second hash function $H'(key)$ according to (15), and here the value of b can be figured out through analyzing the values of $H(key)$ obtained in the first step. After that, we sort the data items by their $H'(key)$ values in ascending order. To break the tie, we choose the data item with higher probability and larger size first. If two data items have the same probability and size, we choose the one with smaller key value first.

After we figure out such an order, we can allocate data items in this order one by one onto broadcast channel from the first bucket, while all the buckets of one data item are consecutively placed one after another on the channel. The last step is to double-check and/or assign global pointers. Figure 10 demonstrates an example of the whole process of extended hash scheme data allocation method with 16 data items in the data set. Note that the initial allocation in (c) is not necessary in the real process but just an illustration, because we can find out the last hole directly from the $H(k)$ values in table (a), and through that we can figure out the

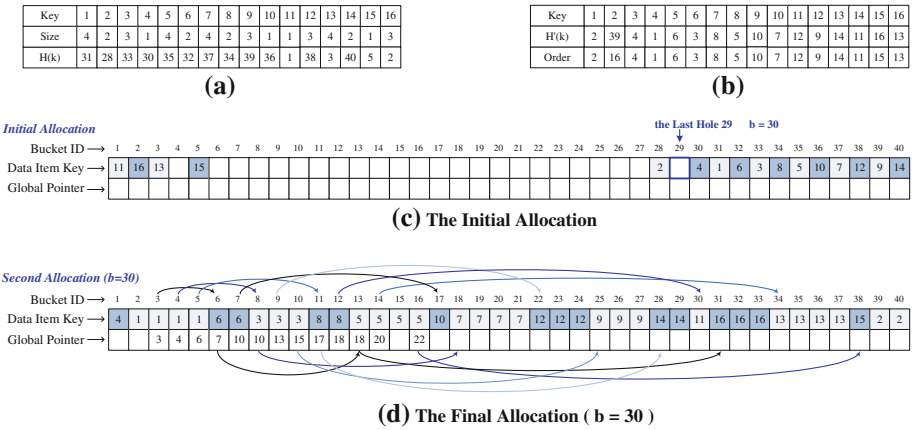


Fig. 10 An example of data allocation method for extended hash scheme

value of b. Table (b) is derived after the initial allocation, and figure (d) is the final allocation according to table (b). We conclude the construction of hash scheme-based indexing strategy in Algorithm 2. Taking into account the complexity of assigning global pointers for each data bucket, the worst-case time complexity of Algorithm 2 would be $O(|D| \times \|D\|)$.

Algorithm 2 ***Construct Extended Hash Scheme and Data Allocation***

Input: D, S, P
Output: *bcast*.
1: **for** $i := 1$ to $|D|$ **do**
2: $H(i) = [(A \times i + B) \mod 2^{31}]\%\|D\| + 1$;
3: **end for**
4: Find the last hole h_0;
5: $b = h_0 + 1$;
6: **for** $i := 0$ to $|D|$ **do**
7: $H'(i) = (H(i) + \|D\| - b) \mod \|D\| + 1$;
8: **end for**
9: Sort data set in ascending order of $H'(i)$ to D';
10: Allocate D' with all its data buckets consequently onto broadcast channel;
11: Assign global pointers for each data bucket;
12: Print(*bcast*);

6.3 Performance analysis of extended hash scheme

In this section, we analyze the system performance of hash scheme by using the metrics of *access latency* and *tuning time*.

First, let us consider access latency. We use s to denote the bucket size, P_i to represent the access probability of bucket i, while $P_i = p_k/s_k$, if bucket i is taken up by datum k with size s_k. Here, we introduce $Dis(k)$ to indicate the displacement area, which means the difference between the physical bucket where datum k resides (denoted as $Phy(k)$) and the designated bucket or hashed bucket for k (denoted as $h'(k)$), thus $Dis(k) = Phy(k) - h'(k)$. Similar to the method in [9], we can calculate the expected access latency for hash scheme by means of calculating each datum k's access time and then average it out, while the expected access time for each datum k is the combination of the following two cases:

– *Case 1: The initial probe is between the hashed bucket and the physical bucket of the data*. Then, the client missed the hashed bucket as well as the global pointer, despite that the physical bucket is still ahead in the current broadcast. Hence, the client has to wait till another *bcast* to get the global pointer in its hashed bucket. This case is calculated as

$$E_1(AL) = (Dis(k)/|bcast|) \times (|bcast| + 1/2 \times Dis(k)) \tag{16}$$

– *Case 2: The initial probe locates outside the displacement area.* In this case, the client has to wait on average between the displacement area and the *bcast*. Thus, we have

$$\begin{aligned} E_2(AL) &= (1 - Dis(k)/|bcast|) \times ((|bcast| - Dis(k))/2 + Dis(k)) \\ &= (1 - Dis(k)/|bcast|) \times (|bcast| + Dis(k))/2 \end{aligned} \tag{17}$$

The expected access latency is calculated as the sum of each datum's expected access time divided by the total number of data items while considering their item size and probabilities.

Next, let us look at the average tuning time for extended hash scheme. It can be calculated by the basic average tuning time plus the average downloading time $\sum_{i=1}^{|D|} s_i p_i$. The maximum value of basic tuning time is 3, which is calculated as the first step for initial probe, the second step for the hashed bucket, and the last (optional) step following the global pointer to the physical bucket. Also for a given data set, we can calculate each datum's tuning time and then average them out based on their access probabilities to get the more accurate result.

7 Huffman tree-based distributed index

In the fourth place, it comes to the *Huffman tree-based distributed indexing* strategy. Huffman tree index has been applied to the wireless broadcasting environment ever since last decades, which is efficient because of its consideration of the access probabilities of data items. The popular data with higher probability reside in higher level of Huffman tree, which reduces search time when traversing from the root node. Considering flat broadcast, we found that the distributed method can be extended to Huffman tree-based broadcast, which is an innovative idea and has not been considered or published before.

Now, it comes to the construction of Huffman tree-based distributed index. The structure of index bucket and data bucket is almost the same as our B^+-tree-based index. First, we construct the k-ary alphabetic Huffman tree by following the methods introduced in [20], based on our sample data set and corresponding access frequencies in Fig. 11.

Take binary alphabetic Huffman tree as an example. In the first stage, we construct a Huffman tree by choosing data nodes i, j as candidates to be merged when all of the following conditions are satisfied: 1) there are no leaves between them, 2) the sum of their frequencies is the minimum over all pairs, and 3) i and j are the leftmost nodes among all pairs.

If the above conditions hold, we create a new index node with frequency equal to the sum of i's and j's frequencies, and replace i and j with this new index node in the node set or construction sequence. This stage produces a tree T_0 without alphabetic ordering of the data nodes, as shown in Fig. 12.

In the second stage, we record the level of each datum node (leaf node) in T_0, denoted as L_i of data d_i, while the root node level equals 1. Next, from the lowest level to the root, we rearrange pointers such that for each level, the leftmost two nodes have the same parent, and then the next two and so on. Thus, we produce an alphabetic Huffman tree T in this way, as shown in Fig. 13, without changing the cost of the tree. We can easily extend this algorithm

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. 11 An example data set of Huffman tree-based distributed index

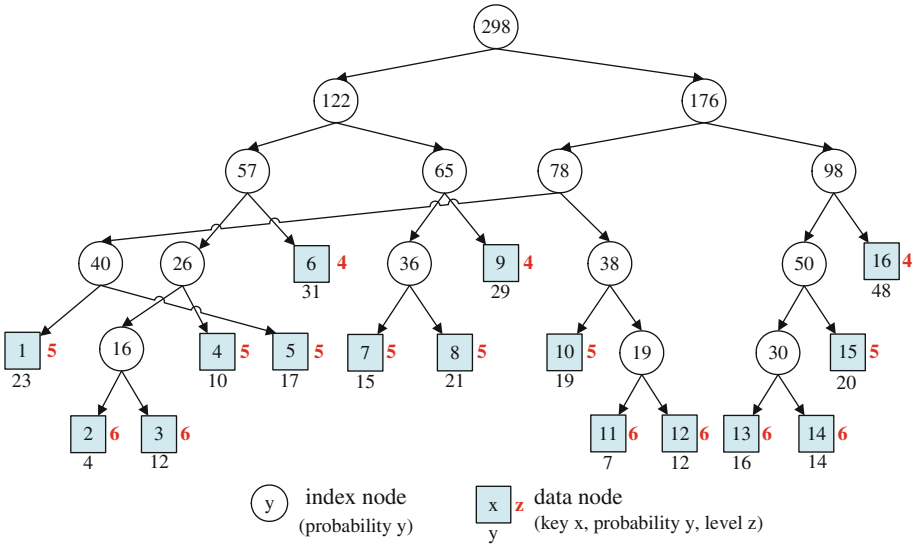


Fig. 12 The first step of constructing Huffman tree

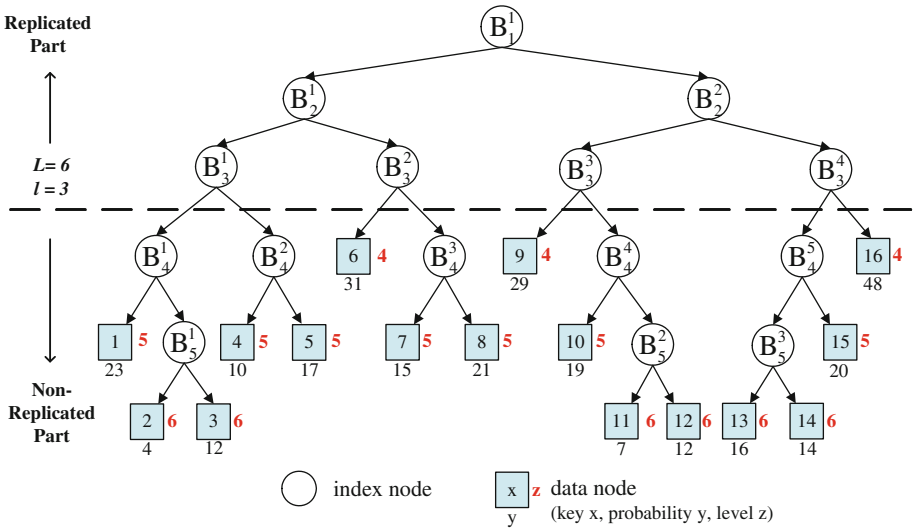


Fig. 13 The final Huffman tree cut at the third level

to construct k-ary Huffman tree, by allowing at most k nodes to be merged in the first stage, and combining up to k nodes with the same parent in the second stage.

Second, we cut this tree T at level l and perform a distributed traversal as in Sect. 4. The index nodes above cutting level is still called control index, and index nodes below cutting level is search index. We append control tables onto control index in the same way

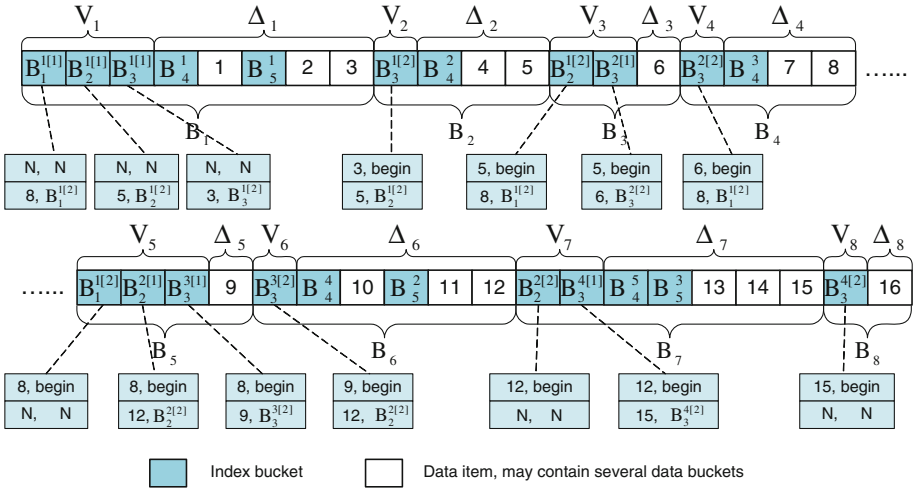


Fig. 14 The broadcast sequence of Huffman tree-based index

as Sect. 4. The final broadcast sequence is illustrated in Fig. 14, and the whole process of constructing the Huffman tree-based distributed index and the broadcasting sequence is presented in Algorithm 3, of which the computational complexity would be $O(|D|^2)$ in the worst case.

Algorithm 3 ***Construct Huffman Tree-Based Distributed Index***

Input: D, P, S
Output: T, *bcast*.
Arrange data set D in alphabetic order → construction sequence CS;
while $CS.size > 1$ **do** ▷ Construct the First Tree T_0;
  if $f_i + f_j = min(f_x + f_y)$ **then**
    if no leaves between i, j **then**
      if i, j are the leftmost nodes among all pairs **then**
        Merge i, j into index node i' with $f_{i'} = f_i + f_j$;
        Insert i' into CS before i; $CS = CS \backslash \{i, j\}$;
      end if
    end if
  end if
end while
Print(T_0);
Traverse T_0 and mark each data d_i's level as L_i;
for $i := max(L_i)$ to 1 **do**
  Rearrange pointers to level i so that each pair of nodes from the left side have the same parent;
end for
Print(T);
Cut T at level l;
Perform a distributed traversal of T to produce the broadcast sequence;
Append control tables onto control index;
Print the final broadcast sequence *bcast*;

7.1 Performance analysis of Huffman tree distributed index

In this section, we analyze the system performance of Huffman tree-based distributed index by metrics of evaluating *access latency* and *tuning time*.

First, let us consider access latency, where all index buckets and data buckets are interleaved on one broadcast channel. The whole *bcast* is divided into $\mathbb{B}_1, \ldots, \mathbb{B}_R$ blocks, where $\mathbb{B}_i = \{V_i, \text{DFT}(\triangle_i)\}$, for $1 \le i \le R$. We use P_i to represent the access probability for block $\mathbb{B}_i$, where 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 $i = 1, \ldots, R$. Let v_i denote the length of V_i, and δ_i indicate the length of $\triangle_i$. Since an index bucket may have different size compared to a data bucket, then we continue to use "r" as the ratio of data bucket size to index bucket size.

Theorem 7.1 *If distributed indices and data are interleaved on one broadcast channel, the average access latency for Huffman tree-based distributed index is*

$$E(AL) = \frac{1}{\|bcast\|} \sum_{i=1}^{R} \left(\sum_{w=1}^{R-2} \left(\frac{v_i + \delta_i}{2} + \sum_{j=i+1}^{i+w-1} (v_j + \delta_j) + v_{i+w} + \frac{\delta_{i+w}}{2} \right) P_{(i+w)\%R}(v_i + \delta_i) + \left(\frac{v_i + \delta_i}{2} \right) P_i v_i + \sum_{i=1}^{R} (v_i + \delta_i) P_i \delta_i \right). \tag{18}$$

Proof First, a client tunes into the broadcast channel at block $\mathbb{B}_i$. Then, it waits for w blocks to reach the index which contains the pointer to the required datum d_j at $\mathbb{B}_{i+w}$. Second, the client waits for the first data bucket of d_j to be broadcasted and begins to download, until it gets all the data buckets of d_j. Hence, according to the law of total expectation, we have the above conclusion.

- *Case 1:* $1 \le w < R - 1$. We can divide this case into three phases: 1) the client tunes into block $\mathbb{B}_i$ and takes an average $\frac{v_i + \delta_i}{2}$ 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 downloads data, so the average waiting time is $v_{i+w} + \frac{\delta_{i+w}}{2}$. The mean of the above period is:

$$E(AL|b = i, d = w) = \frac{v_i + \delta_i}{2} + \sum_{j=i+1}^{i+w-1} (v_j + \delta_j) + v_{i+w} + \frac{\delta_{i+w}}{2} \tag{19}$$

- *Case 2:* $w = 0$. The client tunes into V_i of block $\mathbb{B}_i$, and the pointer to the requested 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 its mean becomes:

$$E(AL|b = i, d = 0) = \frac{v_i}{2} + \frac{\delta_i}{2} = \frac{v_i + \delta_i}{2} \tag{20}$$

- *Case 3:* $w = R - 1$. Suppose the client tunes into block $\mathbb{B}_i$, and the required data are just in this block $\mathbb{B}_i$. Unfortunately, the client already missed the index buckets of this block, 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 broadcasted again in the next *bcast*. In this case, the mean of the waiting time is:

$$E(AL|b = i, d = R - 1) = \frac{\delta_i}{2} + \sum_{j=i+1}^{i+w-1} (v_j + \delta_j) + v_i + \frac{\delta_i}{2} = \sum_{i=1}^{R} (v_i + \delta_i) \tag{21}$$

Therefore, considering Eqs. (19), (20), (21), and the law of total expectation, we can conclude the average access latency as follows:

$$
\begin{aligned}
E(AL) &= \sum_{i=1}^{R}\sum_{w=0}^{R-1} E(AL|b=i,d=w)\cdot P(b=i,d=w) \\
&= \sum_{i=1}^{R}\left(\sum_{w=1}^{R-2}\left(\frac{v_i+\delta_i}{2}+\sum_{j=i+1}^{i+w-1}(v_j+\delta_j)+v_{i+w}+\frac{\delta_{i+w}}{2}\right)\frac{P_{(i+w)\%R}(v_i+\delta_i)}{\|bcast\|}\right. \\
&\quad \left.+\left(\frac{v_i+\delta_i}{2}\right)\frac{P_i v_i}{\|bcast\|}+\sum_{i=1}^{R}(v_i+\delta_i)\frac{P_i\delta_i}{\|bcast\|}\right)
\end{aligned}
$$

□

Next, let us look at the average tuning time for Huffman tree distributed index.

Theorem 7.2 *The average tuning time for Huffman tree-based distributed index is*

$$
E(TT) = \frac{2\sum_{i=1}^{R} v_i + (2+r)|D| + 3\sum_{i=1}^{R}\delta_i}{r\|bcast\|} + \left(\frac{l}{2r}+\frac{1}{r}(L_i-l)+s_i\right)p_i \quad (22)
$$

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

Phase 1 The client tunes into broadcast channel and searches for the right index, following which it can get the required data on that same block. We analyze this phase by considering three cases.

- *Case 1: The client first tunes into a control index*. Then, the client can follow the control table to find the right control index in one more step, which is discussed in [4]. The probability of this case is $\sum_{i=1}^{R}\frac{v_i}{\|bcast\|}$, and the average tuning time of this case is $\frac{2}{r}\sum_{i=1}^{R}\frac{v_i}{\|bcast\|}$.
- *Case 2: The first visited bucket is a data bucket*. The client needs to wait for the next nearest control index and then go to the target control index, with a probability of $\frac{|D|}{\|bcast\|}$. The average tuning time is $(1+\frac{2}{r})\frac{|D|}{\|bcast\|}$.
- *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 $\sum_{i=1}^{R}\frac{\delta_i}{\|bcast\|}$, and average tuning time is $\frac{3}{r}\sum_{i=1}^{R}\frac{\delta_i}{\|bcast\|}$.

Phase 2 Next, the client searches for the pointer that directly points to the required data. Then, it sleeps until the required data appears and then tunes in again to download data. The average tuning time of this step is $(\frac{l}{2r}+\frac{1}{r}(L_i-l)+s_i)p_i$, where L_i is the level of data d_i in the Huffman tree.

Finally, by summarizing the above steps, we can obtain the average tuning time of Huffman tree-based distributed index as follows:

$$
\begin{aligned}
E(TT) &= \frac{2}{r}\sum_{i=1}^{R}\frac{v_i}{\|bcast\|}+\left(1+\frac{2}{r}\right)\frac{|D|}{\|bcast\|}+\frac{3}{r}\sum_{i=1}^{R}\frac{\delta_i}{\|bcast\|}+\left(\frac{l}{2r}+\frac{L_i-l}{r}+s_i\right)p_i \\
&= \frac{2\sum_{i=1}^{R} v_i + (2+r)|D| + 3\sum_{i=1}^{R}\delta_i}{r\|bcast\|} + \left(\frac{l}{2r}+\frac{L_i-l}{r}+s_i\right)p_i
\end{aligned}
$$

□

Table 2 Simulation parameters

System parameters	Range
Database size (the number of broadcast data items)	1,000–10,000
Total number of simulations for each parameter settings	100,000
Size of a data item (the number of data buckets)	1–4
Size of a data bucket	1 KB
Data bucket size/index bucket size	1–50

8 Experiments and performance evaluation

In this section, we simulate the unified wireless data broadcasting system, analyze, and compare the performance of each aforementioned indexing scheme, i.e., the B^+-tree-based distributed index (B^+-*tree*), the exponential index (*Exponential*), the extended hash scheme (*HASH*), and the Huffman tree-based distributed index (*Huffman*). Simulations of the performance comparison were implemented using JAVA NetBeans IDE 6.7.1 and carried out on a 64-bit Intel Xeon E5520 2.27 GHz Quad-Core Server with 6 GB memory. The various parameters used in our simulations are tabulated in Table 2.

We simulate a base station that continuously broadcasts a database with 1,000 to 10,000 data items onto a broadcasting channel, while there are multiple clients within the broadcasting region requesting different sets of data items. Data items have different sizes varying from 1 to 4 data bucket(s). Each data bucket is set to be of size 1 KB, and the size of each index bucket can be calculated to be 0.1 KB [35]; thus, we set the ratio of index bucket size to data bucket size as $1/r = 0.1$, while other existing works rarely discussed about this ratio. They always assume that data items have equal size, and that index bucket has the same size as data bucket, which is not accurate in practice. Therefore, in order to produce more accurate results and get much closer to the reality scenario, we consider different data item sizes and bucket size ratio r.

Our simulator works as follows: For each type of indexing method, our simulator first generates the broadcast sequence and allocates them onto the broadcast channel. Next, it will generate a series of requests according to the data access probabilities. For each generated request, the simulator executes searching by following the certain type of indices implemented in the system.

The access probability of data items satisfy the Zipf distribution [16], which is a model for non-uniform (or skewed) data access pattern of mobile clients [11,26]. It produces more skewed access patterns as the parameter θ becomes larger. When $\theta = 0$, the access pattern satisfies uniform distribution. The default value of θ is set to 1. In each group of experiments, we generate 100,000 requests for each parametric settings, and thereafter we calculate the *average access latency* (*AAL*) and *average tuning time* (*ATT*) during data retrieval for each type of indexing scheme, respectively, where AAL and ATT are measured in milliseconds.

8.1 Varying database size

In the first set of experiments, we vary the number of data items in the database from 1,000 to 10,000 and evaluate the performance of each indexing scheme.

Figures 15 and 16 demonstrate the comparisons among AALs and ATTs for the aforementioned indexing schemes as well as the plain broadcast without index. In Fig. 15, exponential scheme has the longest AAL among indexing schemes, due to its exponential feature, whereas

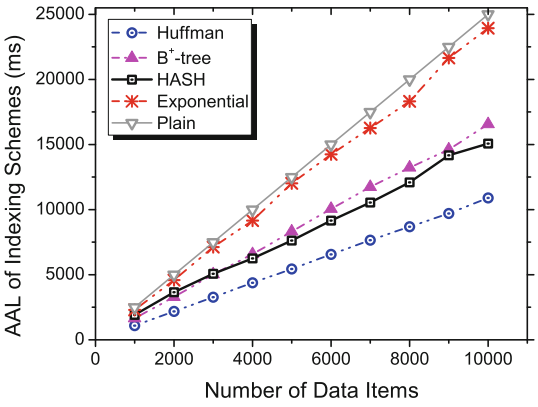


Fig. 15 AAL *w.r.t.* database size

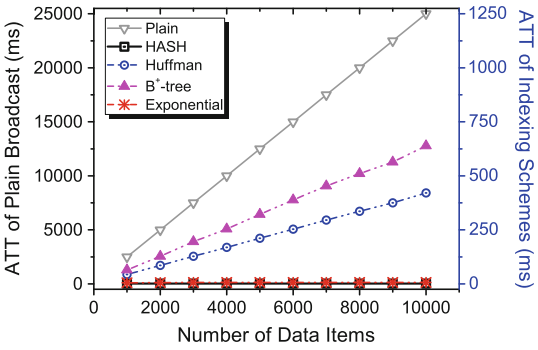


Fig. 16 ATT *w.r.t.* database size

Huffman scheme has the shortest AAL, which is always less than half of that of exponential scheme. Both hash and B^+-tree schemes have average AAL, and the former is larger at first but after the database size exceeds 3,000, it becomes smaller than the latter. From Fig. 16, we can find that the ATT of HASH is extremely small, and that the ATT of exponential scheme is also quite small, while the ATT of B^+-tree is the largest among indexing schemes. Note that the scale of the left y-axis for plain broadcast is 20 times as much as that on the right-hand side for the other indexing schemes, which indicates that our indexing schemes achieve significant improvements on energy efficiency.

Specifically, the AAL and ATT of B^+-tree scheme reveal that the AAL of B^+-tree increases linearly as the database size increases, approximately 59.9 % of the total length of one broadcast cycle, while the trend of B^+-tree's ATT also grows linearly as the database size increases, from the minimum value of around 64.781 to the maximum value of 638.764.

When we consider the AAL and ATT of exponential scheme, we find out that the AAL of exponential scheme increases rapidly as the database size increases, nearly approaching 92.1 % of the total length of one broadcast cycle. On the other hand, the trend of its ATT grows quite slowly as the database size increases, from the minimum value of around 5.341 to the maximum value of 6.952, and then almost keeps stable there. We can also see that the ATT of exponential scheme is extremely small in our system.

The AAL and ATT of HASH scheme indicate that the AAL of HASH increases linearly as the database size increases, which is approximately 68.4 % of the total length of one broadcast cycle, whereas the ATT of HASH remains stable around 1.5, which is extremely small compared to the database size as large as 10,000 in our system.

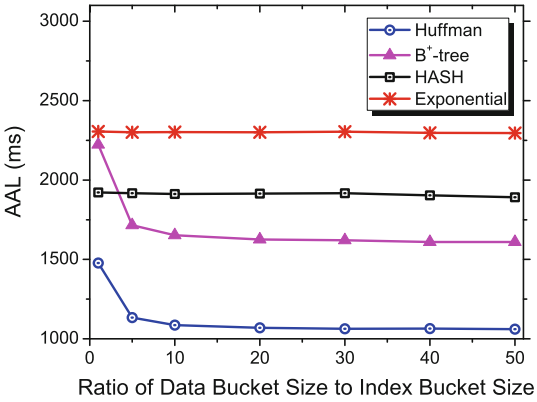


Fig. 17 AAL *w.r.t.* bucket size ratio *r*

If we look into the AAL and ATT of Huffman scheme, it can be found that the AAL of Huffman increases gradually as the database size increases, approximately 41.9 % of the total length of one broadcast cycle, whereas the ATT of Huffman increases linearly as the database size increases, growing from the minimum value of around 43.158 to the maximum value of 420.523.

From this set of experiments, we can conclude that hash scheme is a remarkable energy-efficient indexing scheme with well-acceptable AAL for wireless data broadcasting, and that Huffman scheme is a remarkable time-efficient indexing scheme which returns query data much faster than the other schemes. Exponential scheme is a promising energy-efficient indexing scheme; although its AAL is the longest, it is still acceptable. Last but not least, B^+-tree is a type of time-efficient indexing scheme with average AAL.

8.2 Varying bucket size ratio

As we mentioned before, the size of an index bucket is very small compared to that of a data bucket and the size differences may have significant influence on the performance of various indexing schemes. Thus, in this set of experiments, we evaluate the effect of size ratio r of a data bucket size to an index bucket size by varying it from 1 to 50.

The results shown in Figs. 17 and 18 illustrate that the bucket size ratio r has great impact on Huffman scheme and B^+-tree scheme, but has little impact on hash and exponential schemes. For Huffman scheme and B^+-tree scheme, when the ratio r increases from 1 to 20, both AAL and ATT of them decrease sharply at first and then tend to be stable. Specifically, when the ratio r increases from 20 to 50, the values of both AAL and ATT remain stable. On the other hand, for hash and exponential schemes, the values of both AAL and ATT remain stable no matter how the ratio r changes. The reason is that hash scheme and exponential scheme do not require index buckets in their broadcast cycle, but Huffman scheme and B^+-tree scheme need a number of index nodes to facilitate searching. When the bucket size ratio r increases, the size differences increase, which means that the index bucket becomes comparatively smaller and produces less impact on the broadcast cycle. Therefore, Huffman scheme and B^+-tree scheme perform better when the bucket sizes ratio r is larger.

8.3 Construction complexity and computing complexity

In this set of experiments, we try to evaluate the construction complexity and computing complexity for different types of indexing schemes.

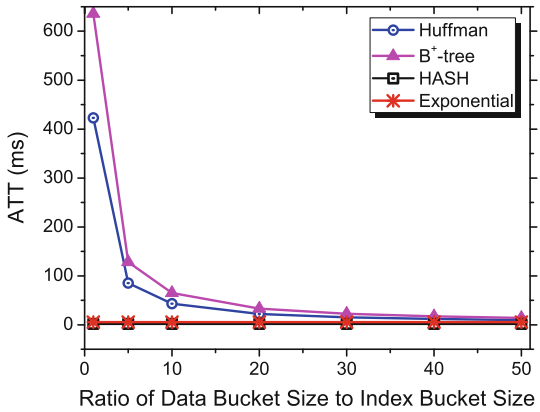


Fig. 18 ATT *w.r.t.* bucket size ratio r

Table 3 The ratio of searching time to construction time for different indexing schemes

	HASH	Huffman tree	B^+-tree	Exponential
Construction time CT (ms)	3.0915	28.1141	0.6207	0.0106
Searching time ST (ms)	0.1326	0.0438	0.0637	1.0477
Ratio = ST/CT	0.0429	0.0016	0.1026	98.750
$C = \alpha ST + \beta CT$	0.1329	0.0466	0.0638	1.0476

From Table 3, we can see that Huffman tree scheme has the shortest average searching time among the four schemes, followed by B^+-tree and hash schemes, whereas the exponential scheme has the longest searching time, which is almost 23 times longer than that of Huffman scheme. On the other hand, the average construction time of Huffman scheme is the longest among the four schemes. However, it is only 28.1141 milliseconds, which is extremely short in practice. Since we only perform construction once but may query thousands of data items, so the searching time plays a more crucial role in this set of experiments. Therefore, we set up the cost function of searching time and construction time with parameters $\alpha = 0.9999$ and $\beta = 0.0001$. The results confirm that Huffman tree scheme has the best performance and the lowest computing complexity.

8.4 The length of broadcast cycle

In this set of experiments, we compare the length of one full *bcast*, (i.e., $\|bcast\|$) among hash, Huffman, B^+-tree, and exponential schemes, as well as the plain broadcast without indices, which indicates the overhead of each indexing schemes.

Figure 19 demonstrates that for the same set of data items, exponential scheme has the longest *bcast*, followed by hash scheme, while B^+-tree scheme and Huffman tree scheme have the shortest *bcast*. As the total number of data items increases from 1,000 to 10,000, the length differences of $\|bcast\|$ become more remarkable. The reason is that the growth of the total number of index buckets in one *bcast* might expand the total length of that *bcast* in either Huffman scheme or B^+-tree scheme, while the total number of data items may increase the number of entries of each index table in exponential scheme.

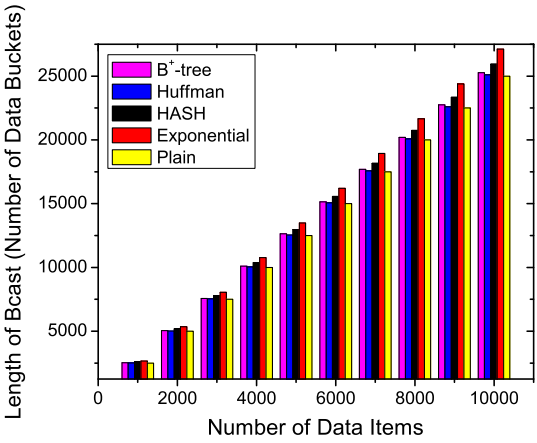


Fig. 19 ||*bcast*|| *w.r.t.* database size

Table 4 The comparison of different indexing schemes

Features	HASH	Huffman	B^+-tree	Exponential
Energy efficiency	Excellent	Good	Fair	Excellent
Time efficiency	Good	Excellent	Good	Fair
Efficient for skewed data	No	Yes	No	No
Better with smaller index	No	Yes	Yes	No
Resilient to link errors	Yes	Sometimes	Sometimes	Yes
Ease of construction	Fair	Poor	Good	Excellent
Ease of searching	Fair	Excellent	Good	Acceptable
Has short Bcast	Good	Excellent	Good	Fair
Flexibility	No	Yes	Yes	Yes
Clustered or non-clustered	Non-clustered	Non-clustered	Clustered	Clustered

9 Comparison of different indexing schemes

Now, after a series of detailed comparisons, we present another group of more intuitive comparisons in the form of a table, for the following measurement criteria: energy efficiency and time efficiency, effect of skewed access probability and index bucket size, performance under link errors, construction complexity and searching complexity, length of *bcast*, flexibility to tune between AAL and ATT, and clustered or non-clustered features. Note that exponential scheme can be extended to non-clustered broadcasting, but that requires further improvement and modification, which is beyond our consideration in this paper.

In Table 4, the performance of these schemes has been categorized into four grades, i.e. *excellent*, *good*, *fair*, and *poor*, based on the simulation results. The measurement "Ease of Searching" mainly evaluates the complexity of the searching algorithms in terms of the average time to answer one query in each scheme. From Table 4, we obtain the following conclusion in general.

> ***B^+-tree:*** It is easy to construct and performs well in searching, especially with smaller index buckets. It has short access time, is flexible and sometimes resilient to link errors. Service providers may consider our B^+-tree scheme when the data set is often updated

and most clients prefer shorter response time compared to low energy consumption, especially where data sizes are different in the database.

Exponential: It is the easiest one to construct. It is resilient to link errors and is flexible, also has short *bcast* and consumes less energy. Exponential scheme should be considered in a system where the data set needs to be updated frequently and link errors occurs a lot, whereas most clients prefer low energy consumption rather than fast response. Actually, sometimes clients might expect a long waiting time to obtain the target data.

HASH: This scheme consumes less energy, has short *bcast* and short access time. It is also resilient to link errors. Hash scheme is the best choice under the circumstance that most clients require both short response time and minimum energy consumption, since hash scheme can achieve almost optimal tuning time. It works better in those systems that need not be updated frequently, because it may take some time constructing the broadcasting sequences. Also, the hash functions may need further modifications for different data sets.

Huffman tree: It performs better in searching, especially for the data set with more skewed access probabilities and smaller index buckets. It is flexible, has short *bcast* and best time efficiency, consumes less energy, and sometimes is resilient to link errors. Service providers should consider Huffman tree scheme when most clients require the minimum response time and low energy consumption, especially when the data items have quite skewed access probabilities in the data set. However, it might not be a good choice when the data set needs to be updated frequently.

9.1 Extended comparison

In the previous sections, we mainly focus on four of the most popular indexing schemes. Actually, other indexing schemes can also be evaluated under our model. Among them, tree-based indexing schemes (if they are balanced trees) should produce similar but hardly better performance than B^+-tree index, or just similar to Huffman tree scheme (if they are not balanced trees). Among non-tree-based indices, function-based schemes that use a function to map data key values to the locations on the channel should have a similar performance to hash scheme, and table-based schemes might perform like exponential index. Non-flat broadcasting works better for the case that the data set is skewed on the access frequency, especially when some data items have extremely higher frequencies than others.

Generally speaking, multi-channel broadcasting should produce better performances than single-channel broadcasting. For example, under the ideal scenario, the average access latency in a dual-channel broadcasting system should be reduced by half of that in a single-channel system, if using exactly the same type of indexing scheme and omitting the overhead of channel pointers and synchronization mechanism. In a real scenario, although we may not achieve the ideal performance of multi-channel broadcasting, we should still be able to get a better performance than single-channel broadcasting. For instance, [33] recently proposed a hash-based scheme called *HAMHash*, which is a kind of interleaved non-flat broadcasting scheme with good performance. More in-depth comparison of HAMHash and other schemes is beyond the scope of this paper, yet can be found in the work [33].

In Table 5, we present an additional group of intuitive comparisons among some other indexing schemes, where the measurement criteria include: skewed access patterns, multi-channel or single channel, skewed or flat broadcast, correct response or false results. According to the performance baseline established in Table 4 (from excellent to poor of the aforementioned four schemes), and the strong interrelationships among flexible, $(1, m)$, signature, and HAMHash schemes, we provide further comparisons for this set of schemes.

Table 5 Further comparison of additional indexing schemes

Features	HAMHash	Signature	Flexible	$(1, m)$ Index
Energy efficiency	Excellent	Good	Capable	Fair
Time efficiency	Good	Good	Capable	Acceptable
Skewed access	Yes	No	No	No
Resilient to link errors	Yes	No	No	No
Multi-channel	Yes	No	No	No
Correct response	Yes	Sometimes	Yes	Yes
Skewed broadcast	Yes	No	No	No
Flexibility	Yes	No	Yes	Yes
Clustered or non-clustered	Non-clustered	Clustered	Clustered	Clustered

On the other hand, when it comes to the non-interleaved broadcasting, the broadcasting scheme varies a lot. There can be a number of different allocation methods for each single type of index on multiple channels. Also, it is not easy to achieve a truly fair comparison among different types of index schemes under various allocation scenarios. We need much more detailed discussions on that, which is beyond the scope of this paper. Due to space limitation, such work will be split into a series of papers as future work.

To sum up, it is hard to say which indexing scheme performs better, since every scheme has its own features, advantages, and disadvantages. Therefore, in this paper, we provide all these comparisons in order to guide the service providers to choose from various indexing schemes according to their specific needs and requirements for their systems.

10 Conclusion

In this paper, we construct a novel evaluation strategy with unified communication environment to evaluate and compare the performance of various indexing technologies. Among a number of commonly used indexing schemes, we choose four of the most popular indices, namely the distributed index, exponential index, hash scheme, and Huffman tree index, redesign these schemes and try to improve their performance under our unified broadcasting environment, in order to evaluate their features, performance, and efficiencies with the same criteria.

First, we set up a unified communication environment as a base for comparison and redesign the index structures such that they can work smoothly under the system model. Next, we create a novel evaluation strategy, use probability theory to formulate the performance of each scheme theoretically, and construct the simulation model to evaluate their performance by numerical experiments.

To conclude, for a given data set, the most efficient method in energy aspect is the hash scheme; the most efficient method in time aspect is the Huffman scheme. B^+-tree scheme is easy to construct and performs well too. Exponential scheme is also easy to construct and resilient to link error.

In summary, we are the first work to provide a scalable communication model and accurate evaluation strategies. Service providers can easily modify the communication environment or introduce other indexing techniques to our system and use our comparison model to choose the best indexing scheme to satisfy their specific requirements for their data broadcasting systems.

We strive to study the performance of all commonly used indices in all possible situations. Due to space limitation, such work will be split into a series of papers as future work. Since system performance in skewed broadcast heavily relies on data scheduling design and algorithms, but here we are aiming at the performance of indices, so we only discuss flat broadcast as the first stage. In our future work, all existing situations will be discussed and analyzed accordingly.

References

1. Acharya S, Alonso R, Franklin M et al (1995) Broadcast disks: data management for asymmetric communication environments. In: Michael C, Donovan S (eds) Proceedings of the ACM SIGMOD international conference on management of data, San Jose, CA 24(2):199–210
2. Chen CC, Lee C, Wang SC (2009) On optimal scheduling for time-constrained services in multi-channel data dissemination systems. Inf Syst 34(1):164–177
3. Chen M, Yu P, Wu K (1997) Indexed sequential data broadcasting in wireless mobile computing. In: Proceedings of the international conference on parallel and distributed systems Seoul, Korea, pp 124–131
4. Gao X, Shi Y, Zhong J et al (2012) SAMBox: a smart asynchronous Multi-channel black box for wireless data broadcast. In: Proceedings of the 21st international conference on software engineering and data engineering, Los Angeles, CA
5. Hsu C, Lee G, Chen A (2002) Index and data allocation on multiple broadcast channels considering data access frequencies. In: Proceedings of the third international conference on mobile data management, Singapore, pp 87–93
6. Hu Q, Lee W, Lee D (2004) A hybrid index technique for power efficient data broadcast. Distrib Parallel Databases 9(2):151–177
7. Hu T, Tucker A (1971) Optimal computer search trees and variable-length alphabetic codes. SIAM J Appl Math 21(4):514–532
8. Im S, Choi J (2012) MLAIN: multi-leveled air indexing scheme in non-flat wireless databroadcast for efficient window query processing. Comput Math Appl 64(5):1242–1251. doi:10.1016/j.camwa.2012.03.068
9. Imielinski T, Viswanathan S, Badrinath B (1994) Power efficient filtering of data on air. In: Proceedings of the international conference on extending database technology, Cambridge, UK, pp 245–258
10. Imielinski T, Viswanathan S, Badrinath BR (1997) Data on air: organization and access. IEEE Trans Knowl Data Eng 9(3):353–372
11. Jung S, Lee B, Pramanik S (2005) A tree-structured index allocation method with replication over multiple broadcast channels in wireless environment. IEEE Trans Knowl Data Eng 17(3):311–325
12. Lee WC, Zheng B (2005) A fully distributed spatial index for wireless data broadcast. In: Aberer K, Franklin M, Nishio S (eds) Proceedings of the international conference on data engineering, Tokyo, Japan, pp 417–418
13. Lee W, Lee D (1996) Using signature techniques for information filtering in wireless and mobile environments. Distrib Parallel Databases 4(3):205–227
14. Lu X, Gao X, Yang Y, Zhong J (2013) SETMES: a scalable and efficient tree-based mechanical scheme for multi-channel wireless data broadcast. In: Proceedings of the ACM international conference on ubiquitous information management and communication, Kota Kinabalu, Malaysia
15. Lu Z, Wu W, Fu B (2012) Optimal data retrieval scheduling in the multi-channel wireless broadcast environments. IEEE Trans Comput PP(99):1. doi:10.1109/TC.2012.139
16. Manning C, Schütze H (1999) Foundations of statistical natural language processing. MIT Press, Cambridge
17. Pichevar R, Najaf-Zadeh H, Thibault L, Lahdili H (2011) Auditory-inspired sparse representation of audio signals. Speech Commun 53(5):643–657
18. Shen J (2008) Data access mechanisms for skewed access patterns in wireless information systems. National Sun Yat-sen University, Dissertation
19. Shi Y, Gao X, Zhong J, Wu W (2010) Efficient parallel data retrieval protocols with MIMO antennae for data broadcast in 4G wireless communications. In: Proceedings of the international conference on database and expert systems applications, pp 80–95
20. Shivakumar N, Venkatasubramanian S (1996) Efficient indexing for broadcast based wireless systems. J Mobile Netw Appl 1(4):433–446

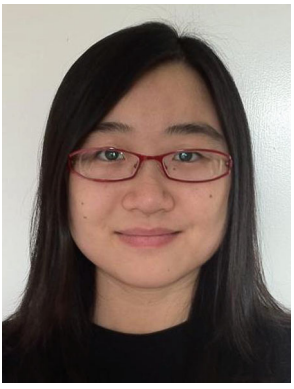
21. Tsakiridis F, Bozanis P, Katsaros D (2007) Interpolating the air for optimizing wireless data broadcast. In: Zomaya A, Zeadally S (eds) Proceedings of the ACM international workshop on mobility management and wireless access, Chania, Crete Island, Greece, October 2007, pp 112–119
22. Vaidya N, Hameed S (1999) Scheduling data broadcast in asymmetric communication environments. Wirel Netw 5(3):171–182
23. Vijayalakshmi M, Kannan A (2008) A hashing scheme for multi-channel wireless broadcast. J Comput Inf Technol 16(3):197–207
24. Vlajic N, Charalambous C, Makrakis D (2003) Wireless data broadcast in systems of hierarchical cellular organization. In: Proceedings of the IEEE international conference on communications, Anchorage, Alaska 3:1863–1869
25. Wang J (2012) Set-based broadcast scheduling for minimizing the worst access time of multiple data items in wireless environments. Inf Sci 199:93–108
26. Wang S, Chen H (2007) Tmbt: an efficient index allocation method for multi-channel data broadcast. In: Proceedings of the international conference on advanced information networking and applications workshops
27. Xu J, Lee W, Tang X et al (2006) An error-resilient and tunable distributed indexing scheme for wireless data broadcast. IEEE Trans Knowl Data Eng 18(3):392–404
28. Yang X et al (2002) Bouguettaya A (2002) Broadcast-based data access in wireless environments. In: Jensen C, Jeffery K, Pokorny J (eds) Proceedings of the international conference on extending database technology. Czech Republic, Prague, pp 553–571
29. Yao Y, Tang X, Lim E, Sun A (2006) An energy-efficient and access latency optimized indexing scheme for wireless data broadcast. IEEE Trans Knowl Data Eng 18(8):1111–1124
30. Yee WG, Navathe SB, Omiecinski E, Jermaine C (2002) Efficient data allocation over multiple channels at broadcast servers. IEEE Trans Comput 51(10):1231–1236
31. Zheng B, Lee WC, Liu P et al (2009) Tuning on-air signatures for balancing performance and confidentiality. IEEE Trans Knowl Data Eng 21(12):1783–1797
32. Zhong J (2012) Data management in wireless environment. The University of Texas at Dallas, Dissertation
33. Zhong J, Gao Z, Wu W et al (2013) High performance energy efficient multi-channel wireless data broadcasting system. In: IEEE Wireless communications and networking conference, Shanghai, China
34. Zhong J, Gao Z, Wu W et al (2012) Multi-channel energy-efficient hash scheme broadcasting. In: Proceedings of the 21st international conference on software engineering and data engineering, Los Angeles, CA, June 2012
35. Zhong J, Wu W, Shi Y et al (2011) Energy-efficient tree-based indexing scheme for efficient retrieval under mobile wireless data broadcasting environment. In: Yu J, Kim M, Unland R (eds) Proceedings of the 16th international conference on database systems for advanced applications, Hong Kong, China, April 2011, LNCS 6588:335–351

Author Biographies

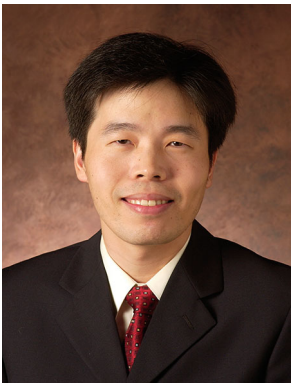
Jiaofei Zhong is currently an Assistant Professor of Computer Science at University of Central Missouri. She received her PhD in Computer Science in 2012 and M.S. degree in 2010, both from the University of Texas at Dallas. Dr. Zhong has served as a peer reviewer for a number of international conferences and journals, and has been the Publicity Chair, Financial Chair, and OCS co-Chair in the organizing committees of several international conferences. Her research interests are in the areas of data engineering and information management, especially in Wireless Communication Environment, including Data Broadcasting, Vehicle Ad hoc Networks, and Sensor Database.

Weili Wu is an associate professor in Department of Computer Science, University of Texas at Dallas. She received her PhD in 2002 and M.S. in 1998 from the Department of Computer Science, University of Minnesota, Twin City. Her research mainly deals with the general research area of data communication and data management. Her research focuses on the design and analysis of algorithms for optimization problems that occur in wireless networking environments and various database systems. She has published more than 100 research papers in various prestigious journals and conferences such as IEEE Transaction on Knowledge and Data Engineering (TKDE), IEEE Transactions on Mobile Computing, IEEE Transactions on Multimedia, ACM Transactions on Sensor Networks, IEEE Transactions on Parallel and Distributed Systems, IEEE/ACM Transactions on Networking, Journal of Global Optimization, Journal of Optical Communications and Networking, Optimization Letters, IEEE Communications Letters, Journal of Parallel and Distributed Computing, Journal of Computational Biology, Discrete Mathematics, Social Network Analysis and Mining, Discrete Applied Mathematics, IEEE INFOCOM, ACM SIGKDD, International Conference on Distributed Computing Systems (ICDCS), International Conference on Database and Expert Systems Applications (DEXA), SIAM Conference on Data Mining, etc.

Xiaofeng Gao received the B.S. degree in Mathematics from Nankai Univ., China, the M.S. degree in Operations Research from Tsinghua Univ., China, and the PhD degree from Univ. of Texas at Dallas, USA. She is currently an associate professor at Department of Computer Science and Engineering, Shanghai Jiao Tong Univ., China. She has published more than 50 refereed journal and conference papers and has given several invited presentations at international conferences. She has served on organizing and/or TPC for numerous international conferences. Her research interests include data engineering, data center, and combinatorial optimization in networks.



Yan Shi is currently an Assistant Professor of Software Engineering in the Department of Computer Science and Software Engineering at University of Wisconsin-Platteville. She received her PhD in Computer Science from The University of Texas at Dallas in 2011. Her research interests focus on data management and data engineering in wireless communications, data mining, and software quality.



Xiaodong Yue received his BS and MS from Shanghai Jiao Tong University in 1996 and 1999, respectively. He received his PhD from the University of Cincinnati in 2004. Dr. Yue joined the Department of Mathematics and Computer Science at the University of Central Missouri as an assistant professor in 2004 and was promoted to associate professor in 2009. He was a visiting professor with the Department of Computer Science at the Winston Salem State University between 2003 and 2004. Dr. Yue's research interests include wireless communications and signal processing and his research is supported by the National Science Foundation (NSF) and Google. Dr. Yue is a senior member of IEEE.