Improving the Performance of the Wireless Data Broadcast by the Cyclic Indexing Schemes

Long-Sheng Li, Ming-Feng Chang, and Gwo-Chuan Lee

Department of Computer Science and information Engineering,
National Chiayi University,
No.300 Syuefu Rd., Chiayi 60004, Taiwan, R.O.C.
sheng@mail.ncyu.edu.tw, mfchang@csie.nctu.edu.tw, gclee@nuu.edu.tw

Abstract. Wireless data broadcast is an effective approach to disseminate information to a massive number of users. Indexing techniques for broadcasting data can reduce the battery power consumptions of mobile terminals by decreasing the tuning time. The organization of the indexes affects the efficiency of data searching. We investigate how the degree of the index node affects the tuning time, and thus minimize the power consumption of user's terminals. We proposed a performance measurement for the tuning time and a cyclic indexing algorithm. The numerical results suggest the degree of an index node be 3 when the access probabilities of the data tend to be uniformly distributed so that the expected tuning time is minimal. When the access distribution of the data nodes is skewer, the tuning time can be minimized by setting the degree in the index node 2.

Keywords: Broadcast, wireless, tuning time, tuning cost, access time, the Hu-Tucker algorithm.

1 Introduction

Wireless data broadcast is an efficient technology to overcome the limited bandwidth in the ubiquitous computing. Wireless data broadcast over radio channels allows users to access data simultaneously at a cost independent of the number of users. It is a powerful way to disseminate data to a massive number of users in the ubiquitous computing. A centralized server periodically broadcasts the data to a large number of mobile terminals through a wireless medium. The mobile terminals receive the broadcasts and filter out the data that is not desired [2]. This service is especially useful for disseminating data that are commonly accessed, such as traffic information for navigation system and real-time stock information. The location-dependent web service can also utilize wireless data broadcast. One disadvantage of the wireless data broadcast is that the mobile terminals can only wait for the requested data.

Power-efficient wireless communication is another important issue for the wireless data broadcast. A simple way to reduce the power consumption is to add auxiliary information to enable the mobile terminals to receive only the data the user needs. A

mobile terminal can be three power modes: transmission mode, receiving mode, and doze mode.

There are two basic approaches for users to access data through radio channels [8]. One is push-based scheme, where the clients retrieve data by only listening to a certain channel in the receiving mode. The clients cannot inform the broadcast server about what they need due to the lack of uplink communication channels from the users to the server. The other approach is pull-based scheme where the clients send requests to retrieve data. There is an uplink channel through which a client can send requests for specific data items to the broadcast server. The broadcast server may arrange the broadcast sequence according to the request received. In the power management view, the client saves power by avoiding transmissions and waking up from the doze mode only when necessary. The push-based scheme is a better strategy to overcome the limited battery power.

To evaluate the efficiency of the wireless broadcasting, two criteria are often used: access time and tuning time [1]. The access time is the average time from the moment a client requests data identified by a primary key to the point when the requested data are received by the client. The access time determines the response time of data access. The tuning time is the time spent by a client listening to the channel. The tuning time determines the power consumed by the client to retrieve the requested data. Indexing techniques insert auxiliary information indicating when each data item will be broadcasted to reduce the tuning time [1][2][4][5]. After receiving the index, a client waits for the requested data most of time in the doze mode in which low power is consumed and only wakes up to receive data when the requested data is coming. Therefore, the tuning time can be reduced and the battery power is conserved.

2 The System Architecture

A broadcast server broadcasts the data to the clients by radio channels. The clients receive the broadcast data and the requested data are filtered. To consume less power of the clients, the broadcast server inserts indexes before the broadcast data to indicate the offsets to the requested data. The clients receive the indexes and go to doze mode. When the requested data are broadcasted, the client wakes up to the receiving mode and receives the requested data. Moreover, to provide efficient search of the indexes, an alphabetic Huffman tree is used for the indexing tree. The clients using this scheme should tune to the beginning of the indexes to get the offset to the requested data. The waiting time between the start of tuning and the beginning of the indexes is half of a broadcast cycle in average. This is referred to the distributed indexing scheme [1].

To reduce the access time of the distributed indexing scheme, the broadcast bandwidth is spilt into several physical channels or logical channels [9]. All data are assigned into a data channel. The indexes of the same level are proposed to occupy on the same channel. Fig. 1 shows the indexing tree and the channel assignment of the broadcast data.

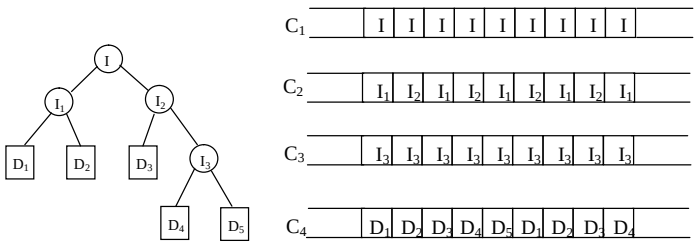


Fig. 1. The indexing tree and the channel assignment of the broadcast data.

The distributed indexing scheme assumed the access probabilities of the data items are the same. Shivakumar and Venkatasubramanian released the assumption [9]. Let n be the number of data items. Every data item has the popularity probability to indicate the expected number of access to the data items. We assume the popularity probabilities of the data items be $f_1, f_2,\ldots, f_n$. If the tuning time for the data item i is T_i, the average tuning time is

$$\left(\sum_{i=1}^{n} f_i \times T_i\right) / \left(\sum_{i=1}^{n} f_i\right).$$

The tuning time T_i is dependent on the depth of the data item in the indexing tree. The problem to construct an indexing tree to minimize the average tuning time is similar to the Huffman coding, but the alphabetic ordering of the data items is preserved. Hu and Tucker proposed an algorithm to optimize the alphabetic-ordered Huffman code [11][12]. Shivakumar and Venkatasubramanian suggested a k-ary Hu-Tucker algorithm to minimize the average tuning time, but didn't describe the algorithm clearly [9].

An open problem is remained unsolved in the k-ary Hu-Tucker algorithm. For some n, it is impossible to construct a tree that the branches of all internal nodes are filled with k nodes. The k-ary Hu-Tucker algorithm constructs the internal nodes with 2 to k branches, but doesn't specify exact rules to construct the internal nodes. A strategy to determine the branches of the internal nodes of an indexing tree to obtain the minimal average tuning time is needed for the k-ary Hu-Tucker algorithm.

The tuning time is determined only by the depth of the indexing tree. If we increase the branches of the index, the tuning time is reduced. However, increasing the branches should increase the capacity of the index. For the wireless broadcasting, the indexes can be broadcasted on the index channels. The size of the index represents the bandwidth requirement of the radio channel. In wireless communications, a radio channel is partitioned into slots of constant size. The 3rd generation personal communication service provides the function to assign the fixed bandwidth of the channel to a dedicated service [8]. Therefore, the size of the indexes should be the same to fit in a time slot. The tuning time should count the number of indexes received and the size of the index. Assume the depth of the data node i is d_i, and the degree of the index is k. β represents the length of the key and the offset. The average tuning time can be expressed as:

$$\overline{T} = \frac{k\beta\sum_{i=1}^{n}(d_i - 1)f_i}{\sum_{i=1}^{n} f_i} = k\beta\sum_{i=1}^{n}(d_i - 1)f_i \quad . \tag{1}$$

3 The Proposed Alphabetic Huffman Algorithms

Huffman tree and minimize the average tuning time. For n data nodes, it may not be possible to construct an index tree where all indexes have exact k downward branches. That is, empty branches are remained in some indexes. We call this category of index as the incomplete index. In our proposed algorithm, we gather those empty links in one index of the index tree, i.e., there is only an incomplete index in the tree. Under this assumption, the number of the non-empty links of the incomplete index can be determined from the number of data nodes, n, and the degree, k. Let a % b represent the remainder of a/b, and k_1 be the number of the non-empty links of the incomplete index. k_1 can be expressed as

$$k_1 = \begin{cases} b + (k-1), & \text{for } n\,\%\,(k-1) = b \text{ and } b = 0 \text{ or } 1 \\ b, & \text{for } n\,\%\,(k-1) = b \text{ and } b \neq 0,1 \end{cases} . \tag{2}$$

Using the techniques of the binary Hu-Tucker tree, we construct the k-ary index tree by merging k nodes with the least access probability into an index node of the index tree. The access probability of the index is the sum of the access probabilities of its k children. The number of the non-empty links of the incomplete index is calculated first. It is due to that we reduce the average tuning time by merging nodes with less access probability into an index in the lower level of the tree. This algorithm will be referred to as the k-ary Incomplete-index First Alphabetic Huffman Algorithm (IFAH). The algorithm is shown in the following.

Step 1. Let $S=(N_1, N_2, \ldots, N_n)$, the ordered list consists of all data nodes sorted by search key in increasing order. $N_i=D_i$.

Step 2. Calculate the number of the non-empty links k_1 of the incomplete index using Equation 2.

Step 3. Construct the incomplete index node.

- Find k_1 consecutive nodes in S whose sum of access probabilities is minimum.
- Replace the k_1 consecutive nodes with an index node in S. The access probability of the index node equals to the sum of the access probabilities of the replaced nodes.

Step 4. If $|S|=1$, then go to Step 7.

Step 5. Construct the k-degree index nodes.

- Find the k "consecutive" nodes, but index nodes can be bypassed, in S that have the minimum sum of access probabilities.
- Replace the k "consecutive" nodes with a new index node in S.

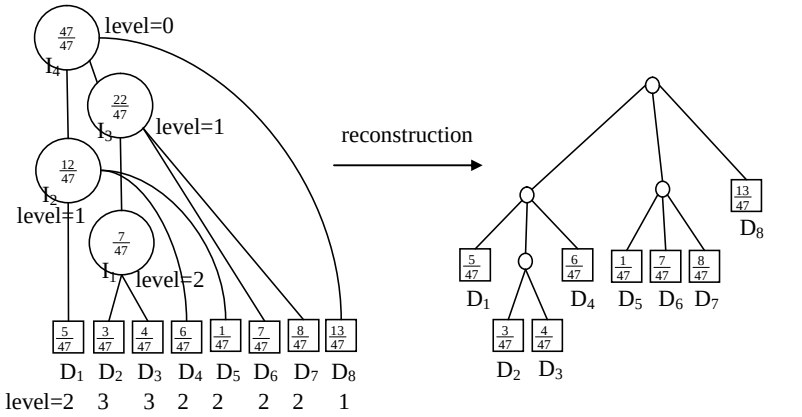
Step 6. If $|S|=1$, then go to Step 7.
Else go to Step 5.

Step 7. Determine the level of each data node in the index tree.

Step 8. Reconstruct the index tree according to the levels of the data nodes.

- Initialize the ordered list *S* as in Step 1.
- Find k_1 consecutive data nodes whose levels are the same. The levels of k_1 consecutive data nodes must be the maximum among the remaining nodes.
- Combine the k_1 consecutive nodes to an index node. Replace the k_1 consecutive nodes with the index node in *S*.
- Find *k* consecutive nodes whose levels are the same and the maximum among the remaining nodes, and combine the *k* consecutive nodes at the highest level first. Then, the nodes on the next-to-highest level are combined.
- The process continues until there is only one node left and its level must be 0.

Fig. 2 shows an example index tree obtained by the IFAH. The boxes represent the data nodes and the numbers in the boxes represent the access probabilities of the data nodes. The circles represent the index nodes and the numbers in the circles represent the combined access probabilities of the linked nodes. *i* is the key for the data node D_i. The IFAH constructs index node I_1 first, because D_2 and D_3 have the minimum sum of access probabilities. In constructing index node I_2, index node I_1 is bypassed, because D_1, D_4, and D_5 are the 3 "consecutive" nodes that have the minimum sum of access probabilities. The index nodes construction process continues until only one node is left in *S*. We assign level 0 to the root. According to the links of the indexes, we assign the level values to all index nodes and data nodes. Then, the index tree is reconstructed from the highest level of the data nodes.



(a) The index tree before reconstruction (b) The index tree after reconstruction

Fig. 2. An example of the IFAH

The format of an index is as follows,

Key 1	Offset 1	Key 2	Offset 2	Key 3	Offset 3

Key *i* is the boundary key value for searching requested data. If the key of the requested data is larger or equal to Key *i* and less than Key (*i*+1), Offset *i* is the offset for the index or data slot of the lower level in the index tree.

Note that the index tree in Fig. 3 places D_1 in its left-most leaf, i.e., the index tree starts from D_1, the first data node. However, a *k*-ary alphabetic tree does not necessarily start from D_1; it can starts from any data node. Fig. 3 shows the index trees starting from different data nodes. The numbers on the links under the index nodes are the boundary key values of the index nodes. Fig. 3 (a) is an example of *k*-ary alphabetic tree starting from D_1. Fig. 3 (b) shows an example of *k*-ary alphabetic tree starting from D_2; the data node before the D_2 is rotated to the end of the ordered list. In this example, we show how to retrieve data node D_1. The boundary key values of the root index are 2, 5, and 6. The key of D_1 is less than 2. Therefore, we chose the offset of boundary key value 6 to obtain the index of the next level. The index of the next level shows the offset of the requested data node D_1. This shows that a *k*-ary alphabetic tree can start from any data nodes. That is, the alphabetic order of the data nodes in the index tree can be treated as a cycle. The average tuning times are 5.62 in Fig. 3 (a) and 4.49 in Fig. 3 (b), respectively. We apply the rotatability to improve the IFAH. The new algorithm will be referred to as the *k*-ary Cyclic Incomplete-index First Alphabetic Huffman Algorithm (CIFAH). The CIFAH modifies Step 3 and 5 of the IFAH. In CIFAH, we treat the ordered list as a cycle and find the minimum sum of access probabilities.

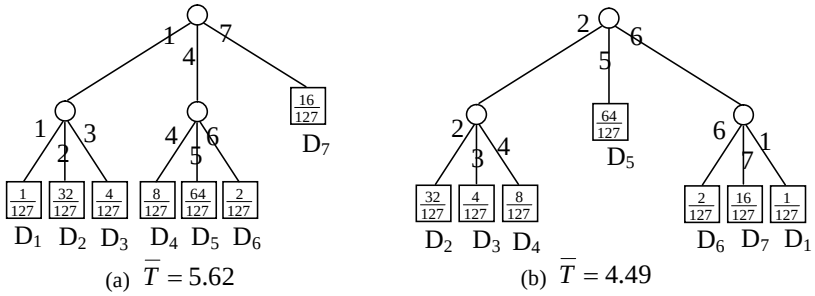


Fig. 3. The index trees with different alphabetic orders.

4 The Numerical Analysis

To simplify the analysis of the tuning cost of the proposed alphabetic Huffman algorithms, we made the following assumptions:

- There is no fault in the broadcasting or reception.
- The initial probe is uniformly distributed in the broadcast cycle.

First, consider a special case where the access probabilities of the data nodes are identical, and the optimal index tree is a full *k*-ary tree that has no incomplete index. Let *k* be the degree in each index, and *d* be the depth of the index tree. The number of data nodes is $n = k^{d-1}$. The average tuning time is $\overline{T} = k(d-1)$. If *k* could be any real number, the average tuning time can be minimized when $\frac{d\overline{T}}{dk} = 0$.

$$\frac{d\overline{T}}{dk} = \frac{d(\frac{k \ln n}{\ln k})}{dk} = 0 \Rightarrow k = e = 2.71828...\cdot$$

Since *k* is a natural number, the result suggests that the average tuning time may be minimized when the degree of the index is 3.

Release the limitation of the full *k*-ary tree, we assume the probability distributions of all data nodes are uniformly distributed. That is, $f_1 = f_2 = f_3 = \cdots = f_n = 1/n$ and $d = \lceil \log_k n \rceil$. The index tree is a full *k*-ary tree when $n = k^d$. For $k^d - k + 2 \le n < k^d$, all leaves are at the same level (level *d*). The average tuning time is $\overline{T} = k(d-1)$. For $k^d - 2k + 3 \le n \le k^d - k + 1$, there are one leaf at level (*d*-1) and (*n*-1) leaves at level *d*. The average tuning time is $\overline{T} = k(n(d-1)-1)/n$. For $k^d - 3k + 4 \le n \le k^d - 2k + 2$, there are two leaves at level (*d*-1) and (*n*-2) leaves at level *d*. The average tuning time is $\overline{T} = k(n(d-1)-2)/n$. For $k^d - k^{d-1}k + (k^{d-1}+1) \le n \le k^d - (k^{d-1}-1)k + (k^{d-1}-1)$, there are *k*-1 leaves at level (*d*-1) and (n-*k*+1) leaves at level *d*. The average tuning time is $\overline{T} = k(n(d-1) - (k^{d-1}-1))/n$. Therefore, if $k^d - (i+1)k + (i+2) \le n \le k^d - ik + i$,we have

$$\overline{T} = k(n(d-1) - i)/n \text{ , for } i=0, 1, \ldots, k^{d-1}-1.$$

The average tuning time can be expressed as

$$\overline{T} = \frac{k(n(d-1) - \left\lfloor \frac{k^d - n}{k-1} \right\rfloor)}{n}, \text{ where } d = \lceil \log_k n \rceil \qquad (3)$$

Fig. 4 shows the tuning time as functions of the number of data nodes and the degree of the index node. The access probabilities of the data nodes are all equal. The number of the data nodes varies from 2 to 1000. The five curves, in the figure, represent the average tuning time for the cases where the degrees are 2, 3, 4, 5, and 6, respectively. The average tuning time increases as the number of data nodes increases due to the increasing height of the index tree. The tuning time increases as the degree of the index is larger than 3. Therefore, when the access probabilities are uniformly distributed, the index nodes of degree 3 tend to minimize the average tuning time.

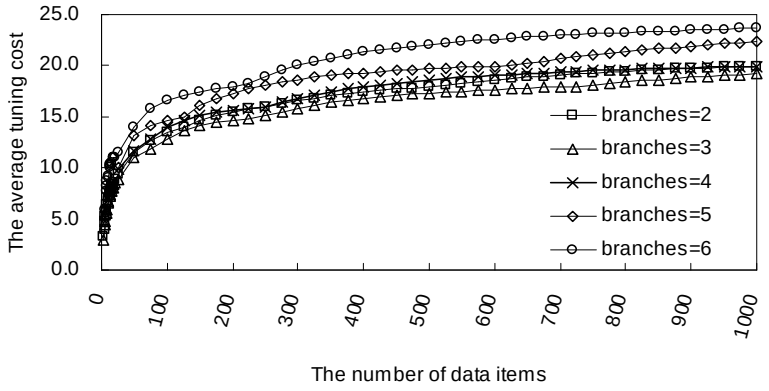


Fig. 1. The average tuning time for data with the uniform distribution.

Consider the case where the access probabilities are non-uniformly distributed. We assume the distribution of the access probabilities is Zipfian [9][14][13]. For n data nodes, the access probability of a data node D_i is as follows,

$$f_i = \frac{1}{i^r \times \sum_{i=1}^{n} 1/i^r},$$

, where r is the rank of the distribution.

Note that, the larger the rank r is, the skewer the probability distribution is. In addition, f_i decreases as i increases. In this sector, we use the rank r to set the access probabilities of data nodes. Then, reorder the sequence of the data nodes using a random number generator. The number of possible sequence orders is $n!$. Therefore, it is impossible to evaluate all possible sequence orders for a large number of data nodes. To simplify the computation, the sequence order is randomly generated. In our experiments, we generate 10000 random sequences for each Zipfian distribution, and then generate the index tree for each random sequence order, and calculate the average tuning time.

Fig. 5 shows the results of the average tuning time for different ranks of Zipfian distribution. For a small rank (e.g., r=0.2) and a large number of data nodes, the minimum average tuning time can be obtained when the degree is 3. The results are consistent with that of the uniform distribution. It is because that a smaller rank for Zipfian distribution results in the less skew probabilities distribution. For a large rank (e.g., r=2) in Zipfian distribution, the minimal average tuning time is found when the degree is 2. This is because the large number of branches increases the tuning time of every data node in the index tree. Consider the index trees of a given degree. The skewer the access probability distribution is, the less the tuning time is. This is because as the access distribution gets skewer, fewer data nodes commands more access probability. The data nodes of large access probabilities trends to be placed at the lower levels of the index tree. As a result, the tuning time decreases.

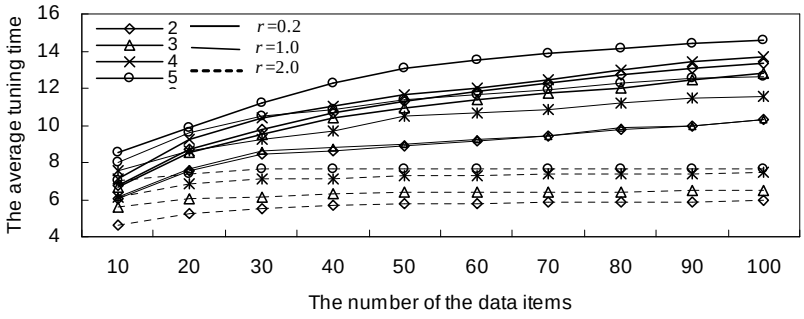


Fig. 2. The average tuning time of different number of the branches with r=0.2, 1.0, and 2.0 with the IFAH.

Fig. 6 shows that the average tuning time of the CIFAH is less than that of the IFAH. The CIFAH is an efficient algorithm in reducing the average tuning time. It is because we can find the minimum tuning time from the selected nodes in the cycle sequence to build low cost indexes in the index tree. The improvement ratio of the CIFAH with large rank r is larger than that with small rank. This is because there are data nodes of larger access probabilities for the skewer access probabilities distributions. The CIFAH has the capability to find an ordered list for the data nodes to construct an index tree that places those frequently accessed data nodes in the lower level. Therefore, the improvement ratio of the tuning time increases.

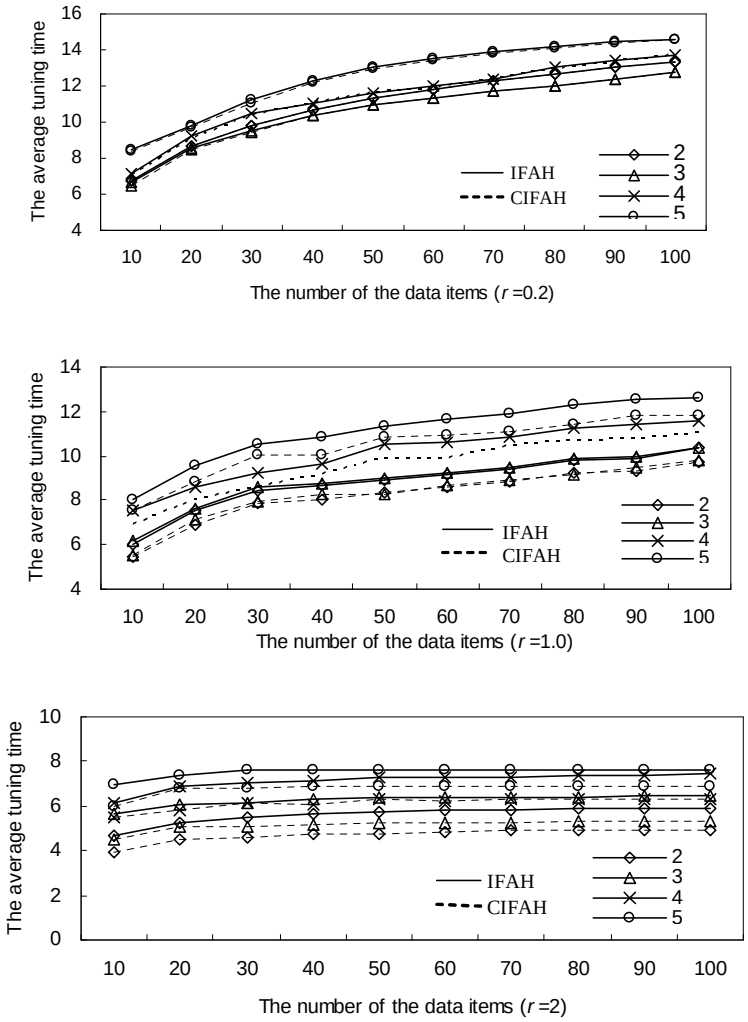


Fig. 3. The average tuning time of the IFAH and CIFAH.

5 Conclusions

In this paper, we proposed indexing schemes to obtain minimal tuning time in the wireless broadcast system. The IFAH is an algorithm similar to the Hu-Tucker algorithm in organizing the indexes. To reduce the tuning time, the CIFAH can improve the IFAH by rotating the sequence of the data nodes.

From the experiments, we have the following results for the indexing schemes.

- If the access probabilities of the data are uniformly distributed, the tuning time is minimal when the degree of the index node is 3.
- For the data nodes whose access probabilities are Zipfian distributed, the tuning time increases as the number of the data nodes increases. It is because that the depth of the index tree increases as the number of the data nodes increases.
- The CIFAH can effectively reduce the tuning time when the access probabilities are of Zipfian distribution, since it is more likely to find consecutive nodes with less access probability to be merged into an index node in the rotatable data cycle.
- For the Zifian distribution, the improvement ratio of the CIFAH increases as rank *r* increases, i.e., the distribution gets more distorted. It is because skewer access probabilities let the CIFAH have more chances to find *k* consecutive nodes of less tuning access probability in the rotatable broadcast cycle to construct an index node in the index tree.
- The tuning time increases as the degree of the index increases, since index of large degree increases the tuning time of every data node in the index tree.

We provide the cyclic indexing construction schemes to reduce the average tuning time. To reduce the tuning time, the degree of the index in the index tree is suggested to be 2 or 3. The frequencies of the broadcasted data may not be uniform in a broadcast cycle. In the future, we can schedule the broadcast sequence according the access probabilities and a new indexing scheme is required to reduce the tuning time.

References

1. T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Energy Efficiency Indexing on Air," In Proceedings of the International Conference on SIGMOD, (1994) 25-36.
2. T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Data on Air: Organization and Access," IEEE Transactions on Lnowledge and Data Engineering, vol. 9, no. 3, (1997) 353-372.
3. C.-J. Su and L. Tassiulas, "Joint Broadcast Scheduling and User's Cache Management for Efficient Information Delivery," Wireless Networks, vol. 6, (2000) 279-288.
4. W. C. Lee and D. L. Lee, "Using Signature Techniques for Information Filtering in Wireless and Mobile Environments," Distributed and Parallel Databases, vol. 4, no. 3, (1996) 205-227.
5. C. K. Lee, H. V. Leong and A. Si, "A Semantic Broadcast Scheme for a Mobile Environment Based on Dynamic Chunking," 20th International Conference on Distributed Computing Systems, (2000) 522-529.
6. H. Saran, R. Shorey, and A. Kumar, "Policies for Increasing Throughput and Decreasing Power Consumption in Bluetooth MAC," 2000 IEEE International Conference on Personal Wireless Communications, (2000) 90-94.
7. S.-C. Lo and L. P. Chen, "An Adaptive Access Method for Broadcast Data under an Error-Prone Mobile Environment," IEEE Transactions on Knowledge and Data Engineering, vol. 12, no. 4, (2000) 609-620.
8. J.-H. Hu, K.-L. Yeung, Gang Feng and K.-F. Leung, "A Novel Push-and-Pull Hybrid Data Broadcast Scheme for Wireless Information Networks," 2000 IEEE International Conference on Communications, vol.3, (2000) 1778-1782.

9. N. Shivakumar and S. Venkatasubramanian, “Energy-Efficient Indexing For Information Dissemination In Wireless Systems,” ACM-Baltzer Journal of Mobile Networks and Nomadic Applications, vol. 1, (1996) 433-446.
10. W.-C. Peng and M.-S. Chen, “Dynamic Generation of Data Broadcasting Programs for a Broadcast Disk Array in a Mobile Computing Environment,” Proc. Of the ACM 9th International Conf. on Information and Knowledge Management (CIKM-00), Nov. (2000) 6-11.
11. T. C. Hu and A. C. Tucker,, “Optimal computer search trees and variable-length alphabetic codes,” SIAM Journal Applied Math., vol. 21, no. 4, (1971) 514-532.
12. D. E. Knuth, “Dynamic Huffman Encoding,” Journal Algorithms, vol. 6, no. 2, (1985) 163-180.
13. W. Li, “Random texts exhibit Zipf’s law-like word frequency distribution,” IEEE Trans. Information Theory, vol. 36, no. 6, (1992) 1842.
14. G. K. Zipf, “Human Behavior and the Principle of Least Effort,” Addison-Wesley Press, Cambridge, Massachusetts (1949)
15. 3GPP TS 25.324: “Radio Interface for Broadcast/Multicast Services.”