An Efficient Nonuniform Index in the Wireless Broadcast Environments [1]

Jun-Hong Shen and Ye-In Chang

Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
Republic of China
{E-mail: shenjh@cse.nsysu.edu.tw}
{Tel: 886-7-5254350}
{Fax: 886-7-5254301}

Abstract

Data broadcast is an efficient dissemination method to deliver information to mobile clients through the wireless channel. It allows a huge number of the mobile clients simultaneously access data in the wireless environments. In real-life applications, more popular data may be frequently accessed by clients than less popular ones. Under such scenarios, Acharya *et al.*'s Broadcast Disks algorithm (*BD*) allocates more popular data appeared more times in a broadcast period than less popular ones, *i.e.*, the nonuniform broadcast, and provides a good performance on reducing client waiting time. However, mobile devices should constantly tune in to the wireless broadcast channel to examine data, consuming a lot of energy. Using index technologies on the broadcast file can reduce a lot of energy consumption of the mobile devices without significantly increasing client waiting time. In this paper, we propose an efficient nonuniform index called the skewed index, *SI*, over *BD*. The proposed algorithm builds an index tree according to skewed access patterns of clients, and allocates index nodes for the popular data more times than those for the less popular ones in a broadcast cycle. From our experimental study, we have shown that our proposed algorithm outperforms the flexible index and the flexible distributed index.

(**Keywords:** broadcast disks, data broadcast, selective tuning, skewed access patterns, wireless network.)

[1]This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-95-2221-E-110-101 and by National Sun Yat-Sen University. The authors also like to thank "Aim for Top University Plan" project of NSYSU and Ministry of Education, Taiwan, for partially supporting the research.

1 Introduction

Due to asymmetric communications in the wireless environments, wireless data broadcast is an efficient way to simultaneously disseminate information to a large number of mobile clients. Under such environments, servers cyclically broadcast data on the wireless channel, and mobile clients use battery-powered devices, *e.g.*, palmtops, to tune in to the wireless broadcast channel to retrieve data without sending requests to the servers. Data broadcast is good at the scenarios where mobile clients have the same commonly interested data on a regular basis. In addition, it scales well when the number of the mobile clients is dramatically increasing. Microsoft's DirectBand Network, for example, utilizes the broadcast technique to provide wireless data services [21, 27]. This wireless network utilizes unused FM radio spectrum to constantly broadcast frequently changing information such as news, weather, sports, and stocks. Then, low-powered mobile devices, *e.g.*, watches, can continuously retrieve these timely information at anytime and anywhere.

Mobile devices have scarce energy resources, *i.e.,* batteries. Due to power limits, energy conservation is a crucial issue for the mobile devices. When a client requests a desired data, the mobile device must actively listen to the broadcast channel to retrieve and examine data until the desired data is downloaded. That is, the mobile device should be continuously in the *active* mode to retrieve data. This is inefficient in terms of energy consumption for the mobile device to continuously retrieve a lot of data on the channel just to pick up one of them [27]. To solve this problem, *selective tuning* is introduced such that the mobile device can slip into the *doze* mode most of the time and listen to the channel only when the relevant data arrives [7]. As a result, it is advantageous to use index technologies on the wireless data broadcast to guide the clients to retrieve data in the listening process. In this way, the mobile devices can be only activated to retrieve the relevant information, thus reducing a lot of energy consumption and lengthening their operating time without recharging.

In the wireless data broadcast environments, the access time and the tuning time are two performance measures [8]. The access time is the average time elapsed from the moment a client requests a data item identified by its primary key, to the moment when the required data item is downloaded by the client. On the other hand, the tuning time is the amount of

time spent by a client listening to the channel. This will determine the energy consumption of the mobile device to retrieve the required data.

In the literature, many studies have been made on reducing energy consumption of mobile devices in the wireless environments. The studies in [4, 5, 7, 8, 11, 20, 21, 22] have proposed index schemes for the uniform broadcast on a single broadcast channel. On the uniform broadcast, all data items are broadcast once in a broadcast cycle. The variant-fanout index tree [3] and the skewed distributed indexing [15] were presented based on data popularity patterns. In [10], Katsaros *et al.* proposed an unbalanced-tree index to handle partially ordered data under skewed access patterns. For supporting spatial queries on the wireless data broadcast, spatial indexes were proposed in [27, 28]. The study in [6] worked on index caching. The studies in [14, 17, 18, 23, 26] proposed index schemes for the nonuniform broadcast on a single broadcast channel. On the nonuniform broadcast, data items are broadcast according to their access frequencies. While the above schemes assume that data items are broadcast over a single channel, there have also been studies that work on broadcasting over multiple channels. The work in [16] studied index allocation, that in [13, 24] discussed data allocation, and that in [9] focused on index and data allocation on the wireless broadcast over multiple channels. The work in [12, 17] concerned on the issue of fault tolerance.

In real-life applications, more popular data may be frequently accessed by clients than less popular ones, *i.e., skewed data access*. For example, the weather conditions of hot attractions may be more frequently accessed than those of cold ones. Under such the scenario, Acharya *et al.* [1] proposed Broadcast Disks (BD) to allocate more popular data appeared more times in a broadcast cycle than less popular ones, *i.e.*, a nonuniform broadcast. BD provides a way of organizing data into a popularity hierarchy which results in a skewed transmission of data and quicker access to more popular data [24].

Under the nonuniform broadcast environment, BD has a good performance on the average access time [13]. However, when requesting a data item of interest, mobile clients should constantly tune in to the broadcast channel to check the received data items until the request item is downloaded. This increases the tuning time that is, on average, half of a broadcast period of the whole file, *i.e.*, a broadcast cycle. The flexible index (FI)

[26] has been proposed to support selective tuning over *BD*. However, *FI* does not take skewed access patterns of clients into consideration. The flexible distributed index (*FDI*) [14] interleaves the tree-based indexes in one broadcast cycle with a given tuning time on the nonuniform broadcast. However, *FDI* may replicate too many index trees in one broadcast cycle, resulting in the increase of the access time. Therefore, in this paper, we propose an efficient nonuniform index called the *skewed index*, *SI*, which is built according to skewed access patterns, over *BD*. Our proposed algorithm also allocates index nodes for popular data more times than those for less popular ones in a broadcast cycle. From our experimental results, we have shown our proposed algorithm outperforms *FI* and *FDI* in the access time and the tuning time.

The rest of this paper is organized as follows. In Section 2, we give a brief description of Acharya *et al.*'s *BD*. In Section 3, we present our proposed skewed index. In Section 4, we study the performance of the proposed algorithm. Finally, a conclusion is presented in Section 5.

2 Background

Acharya *et al.* [1] have proposed the use of a periodic dissemination architecture in the context of wireless mobile systems. They call the architecture *Broadcast Disks*, *BD*. The algorithm has the following steps:

1. Order data items ($= N$) from the hottest (most popular) to the coldest.

2. Partition the list of the data items ($= N$) into multiple disks ($= S$ disks), where each disk D_i, $1 \leq i \leq S$, contains pages ($= K_i$) with similar access probabilities. That is, $N = \sum_{i=1}^{S} K_i$.

3. Choose the relative frequency λ_i of broadcast for each disk D_i, $1 \leq i \leq S$.

4. Split each disk into a number of smaller units, called *chunks* C_{ij}, where C_{ij} denotes the j'th chunk in disk D_i. First, calculate L as the LCM (Least Common Multiple) of the relative frequencies. Then, split each disk D_i into $NC_i = \dfrac{L}{\lambda_i}$ chunks, $1 \leq i \leq S$, where NC_i denotes the number of chunks in disk D_i.

5. Create the broadcast program by interleaving the chunks of each disk in the following manner:

```
for i := 1 to L do
begin
    for j := 1 to S do
    begin
        k := ((i − 1) mod NC_j) + 1;
        Broadcast chunk C_{j,k};
    end;
end.
```

Figure 1 shows an example of the broadcast program generation, where $S = 3$, $N = 7$, $K_1 = 1$, $K_2 = 2$, $K_3 = 4$, $\lambda_1 = 4$, $\lambda_2 = 2$, and $\lambda_3 = 1$. These disks are split into chunks according to Step 4 of the algorithm. That is, L $(= LCM(4, 2, 1))$ is 4, and we have $NC_1 = 1$ (the number of chunks in disk D_1), $NC_2 = 2$, $NC_3 = 4$. The resulting broadcast consists of 4 *minor cycles* (containing one chunk from each disk), *i.e.,* m_1, m_2, m_3 and m_4, which is the LCM of the relative frequencies. The resulting broadcast has a period of 12 pages. This broadcast produces a three-level memory hierarchy in which disk one is the smallest and fastest level and disk three is the largest and slowest level. Thus, the multi-level broadcast corresponds to the traditional notion of a memory hierarchy [1].

3 A Skewed Index for Broadcast Disks

In this section, to provide efficiently selective tuning over Acharya *et al.*'s Broadcast Disks [1], we present our proposed skewed index and the corresponding access protocol.

3.1 Assumptions

This work focuses in broadcast-based wireless environments. Some assumptions should make our work more flexible. The assumptions in this paper are as follows.

1. Data access patterns are skewed.

2. Data items are broadcast over the reliably single channel.

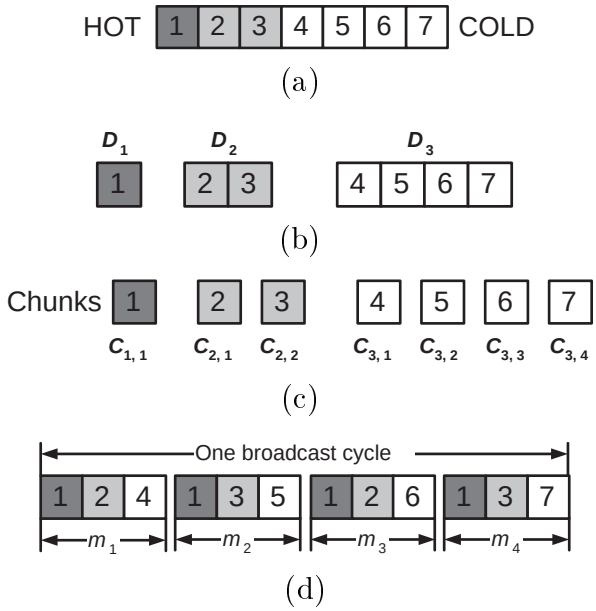


Figure 1: An example of broadcast disks: (a) data items sorted in the descending order of their popularity; (b) three disks with the corresponding data items; (c) chunks for each disk; (d) the final broadcast cycle.

3. Data items are broadcast non-uniformly; that is, the more popular data items appear more times than the less popular ones in one broadcast cycle.

4. Clients request only one data item at one time; *i.e.,* the single query.

5. A bucket is a logical transmission unit on a broadcast channel. An index node can be put into a bucket, the *index bucket*, and a data node can be put into a bucket or more, the *data bucket*.

3.2 The Proposed Algorithm

The basic idea is that we build local index trees for disks, and then combine them from the last disk to the first one to form a skewed index tree. Because fast disks have fewer data items than slow ones, the size of the index trees for the fast disks is smaller than that for the slow ones. Therefore, the combined index tree is skewed. This can reduce the index probes for the popular data. Based on the nonuniform broadcast in *BD*, we can allocate the index nodes for the more popular data items more times than those for the less popular ones in a broadcast cycle.

The algorithm is proceeded as follows.

1. Construct a local index tree for each disk according to the degree of an index node, d, in a bottom-up manner. Note that d is set manually.

2. Combine the local index trees from the last disk to the first one according to d.

3. Distribute index nodes over the broadcast cycle generated by BD by using **Procedure** $AllocateIndexNode(B)$, as shown in Figure 2, where B is a linked list that stores the broadcast cycle generated by BD.

4. Check whether the first node of each minor cycle is the root node. If not, then insert the root node in front of that minor cycle.

Let us use an example to illustrate the proposed algorithm. Take the data items in disks D_1, D_2 and D_3 in Figure 1-(b) as an input. In Step 1, our algorithm first constructs a local index tree for each disk (D_i) in a bottom-up manner as shown in Figure 3-(a), where the degree of each index node, d, is 2. Data items are attached to index nodes with d degrees at the same level and then these index nodes are recursively processed in the same way until the root node is created. For each local tree, branches between a parent node and its children are bi-directional, so a node can easily find its parent and children. Entries in each index nodes are generated through the Bloom filter [2], which is used to test whether an element is a member of a set. Therefore, a data item can be checked whether it is covered by the subtree. It means that through the Bloom filter, a mobile client can determine which branch to follow in the index tree. Applying the Bloom filter may have a very small false-positive (false-drop) probability. In such a case, more than one branch should be followed. However, this can be adjusted to an insignificant effect.

In Step 2, our algorithm then combines these local index trees from the last disk to the first one by considering the degree of an index node, *i.e.*, $d = 2$. Figure 3-(b) shows an index tree after the last two local index trees, the ones for D_2 and D_3, are combined. The final index tree for the data items in Figure 1-(b) is shown in Figure 3-(c). In this skewed index tree, we can observe that the level for the more popular data items is higher than that for the less popular ones.

```
1: procedure AllocateIndexNode (B)
2: begin   /* B is a linked list that stores the broadcast cycle generated by BD. */
    /* c is used to record the current processing data item in B. */
3:     c := the first item of B;
4:     while (c ≠ null) do   /* Check if it is the end of the broadcast cycle. */
5:     begin
6:         copy the content of c to dup to start traversing the index tree;
      /* Traverse the index tree from the data node to the root node. */
7:         while (dup ≠ null) do   /* Check if the root node is traversed. */
8:         begin
9:             if dup = root then    /* root is the root node of the index tree. */
                  /* p is used to record the parent of the current visited node. */
10:                p := null
11:            else
12:                p := the parent of dup;
13:            if dup is the first child of p then
14:            begin
15:                push p into IndexStack;    /* IndexStack is a stack. */
16:                dup := p;    /* The parent p of node dup is then visited. */
17:            end
18:            else
19:            begin
20:                repeat
21:                    pop an index node a from IndexStack and output a;
22:                until IndexStack is empty;
23:                output c;
24:                break;
25:            end;
26:        end;
27:        c := the following item of c in B;    /* Process the next data item. */
28:    end;
29: end;
```

Figure 2: Procedure *AllocateIndexNode*

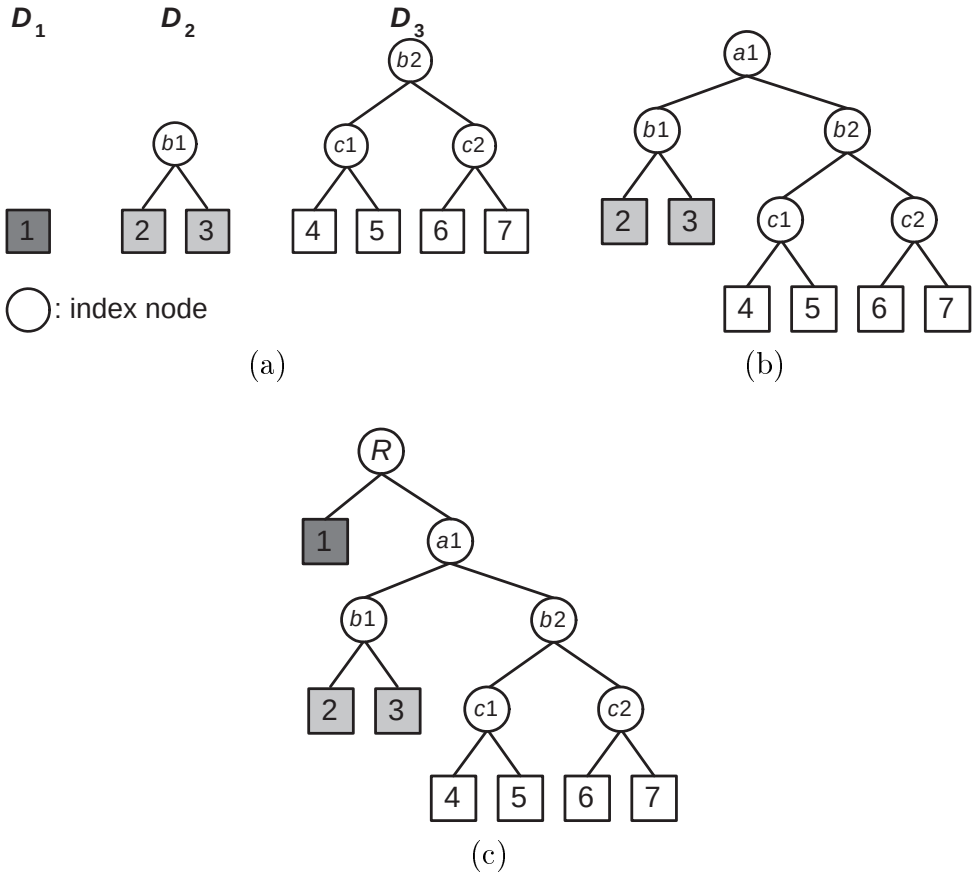
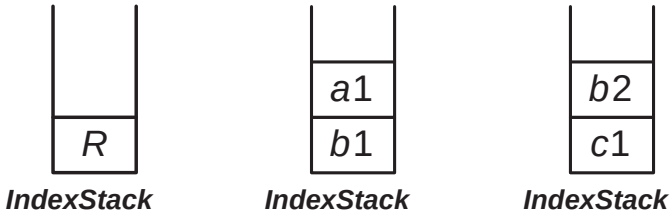


Figure 3: The process of constructing a skewed index tree: (a) local index trees; (b) an index tree after the last two local index trees are combined; (c) the final skewed index tree.

In Step 3, after the skewed index tree is constructed, the algorithm then uses Procedure *AllocateIndexNode*, with the broadcast cycle shown in Figure 1-(d) as an input, to distribute index nodes over the broadcast cycle. Procedure *AllocateIndexNode* allocates the index nodes for the data items in the fast disks to one broadcast cycle more times than those in the slow ones. In Procedure *AllocateIndexNode*, data items of the broadcast cycle stored in linked list B are sequentially processed one by one. The procedure will traverse the index tree from the data node to the root node to determine which index nodes should be put before the processed data item. The policy is described as follows. If the visited node is the first child of its parent, its parent should be put before this node, and then its parent will be visited. Otherwise, the procedure stops traversing the index tree. The reason that we put the corresponding index nodes before their first child node is that these index nodes can cover their corresponding data items in the same broadcast cycle. If we only put these



(a) (b) (c)

Figure 4: The content of stack $IndexStack$ for data items in minor cycle m_1: (a) $IndexStack$ of dealing with data item 1; (b) $IndexStack$ of dealing with data item 2; (c) $IndexStack$ of dealing with data item 4.

index nodes before their other child nodes, some data items covered by these index nodes have been broadcast yet, so that they cannot be retrieved in the same broadcast cycle. This is a directory miss, which will increase the access time. On the other hand, if we put these index nodes before each data node, it will lengthen the broadcast cycle, increasing the access time.

In Procedure *AllocateIndexNode*, for the first data item 1 in the broadcast cycle in Figure 1-(d), since it is not the end of the broadcast cycle, it is copied to *dup* to start traversing the index tree from it to the root node. Since it is not the root node of the skewed index tree shown in Figure 3-(c), its parent node R in that index tree is assigned to p (lines 9–12). (Note that we have to check the relationship between *dup* and p later; that is, we have to check whether *dup* is the first child of p.) From line 13 to line 25, since data item 1 is the first child of its parent p (*i.e.,* R) in the index tree shown in Figure 3-(c), its parent R is pushed into stack $IndexStack$ shown in Figure 4-(a) and its parent R will then be visited. (Note that since we traverse the index tree from the data node to the root node, we need to use stack $IndexStack$ to reverse the output order of the index nodes.) In the second run of the inner while loop, since the current node is the root node in the index tree, p is set to null. Therefore, the if condition about whether *dup* (*i.e.*, R) is the first child of p is false, and all index nodes in $IndexStack$ will be popped out and put before data item 1 (lines 19–25) in the pop-out order. This result is shown in the leftmost part of Figure 5-(a).

For the second data item 2 in Figure 1-(d), since data item 2 and index node $b1$ are the first children of their parents in the index tree shown in Figure 3-(c), their parents $b1$ and $a1$ are pushed into $IndexStack$ shown in Figure 4-(b), respectively. When $a1$ is visited,

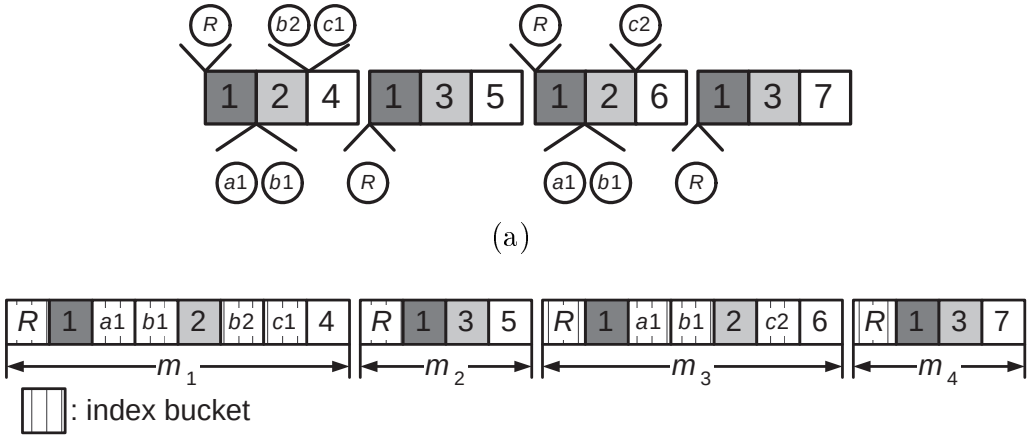
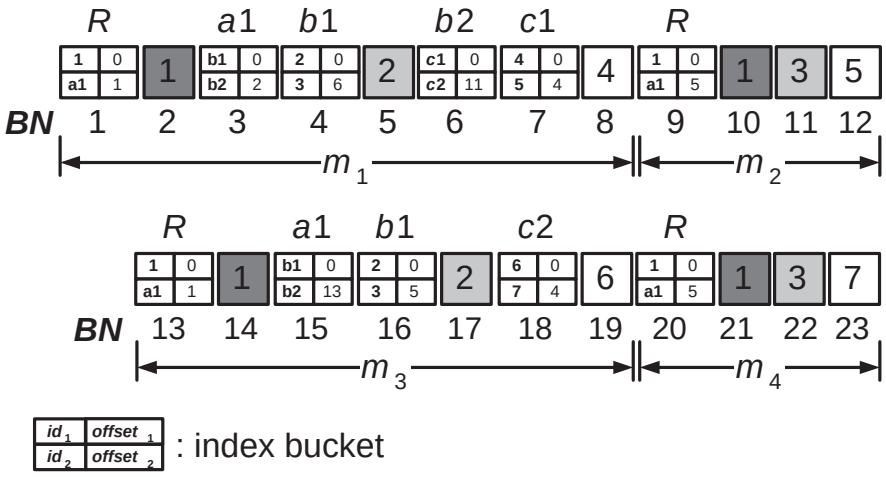


Figure 5: Index distribution: (a) the distribution of index nodes; (b) the final broadcast cycle with index nodes interleaved.

since it is not the first child of its parent R in the index tree, index nodes $a1$ and $b1$ are popped out sequentially, and then put before data item 2 in the broadcast cycle, as shown in Figure 5-(a). For the third data item 4 in Figure 1-(d), since data item 4 and index node $c1$ are the first children of their parents in the index tree, their parents $c1$ and $b2$ are pushed into *IndexStack* shown in Figure 4-(c), respectively. When $b2$ is visited, since it is not the first child of its parent $a1$ in the index tree, index nodes $c1$ and $b2$ are popped out sequentially, and then put before data item 4 in the broadcast cycle, as shown in Figure 5-(a). Until now, the content of minor cycle m_1 is generated, as shown in the leftmost part of Figure 5-(b), where m_i, $1 \leq i \leq 4$, means the ith minor cycle in this broadcast cycle. In the same way, Procedure *AllocateIndexNode* processes all the rest of the data items. Finally, the distribution of index nodes over the broadcast cycle is shown in Figure 5-(a), and the corresponding result of the broadcast cycle with index nodes interleaved is shown in Figure 5-(b).

In Step 4, since clients need the root node to start traversing index nodes, if the first node of a minor cycle is not the root node, our algorithm will insert the root node in front of that minor cycle to help the clients quickly start traversing the corresponding index nodes. In Figure 5-(b), since the first node of each minor cycle is the root node, there is no change for the final result after the processing of Step 4.

Entries in each index node in Figure 5-(b) are shown in Figure 6, where the number



* BN: Bucket number
* $offset_i$ is the offset to the beginning of the bucket containing id_i.

Figure 6: Entries in each index node

below the bucket is the bucket number starting from 1. To guide clients to quickly start traversing the index nodes, except the root nodes, each index or data node also has an entry directing them to the nearest root node, not shown in Figure 6. Each entry in an index node is of the form $< id, offset >$, where id is an identifier generated through the Bloom filter [2] for a data item or index node and $offset$ is the distance in terms of buckets from the end of the current bucket to the beginning of the bucket containing id. For example, in Figure 6, the first entry $< 1, 0 >$ at bucket number 1, *i.e.,* R, represents that data item 1 is 0 bucket away from the current bucket. Note that data item 1 is the first child of root node R as shown in Figure 3-(c). The second entry $< a1, 1 >$ represents that if clients want to get the data items covered by $a1$, they should wait 1 bucket to reach the bucket containing $a1$. Note that, in Figure 6, the second entries $< b2, 13 >$ and $< a1, 5 >$ at bucket numbers 15 and 20 guide the clients to the buckets containing $b2$ and $a1$, respectively, in the next cycle.

3.3 Access Protocol

We now present the access protocol of the proposed algorithm. The access protocol is as follows.

1. Tune in to the broadcast channel to receive the current bucket.

2. Read the current bucket to get the offset of the nearest root node, and go into the doze mode.

3. Tune in to the broadcast channel to receive the nearest root node, and then traverse the index tree until the requested data item is reached or not found. (While waiting for another index node or data node, clients can go into the doze mode for saving energy.)

For example, in Figure 6, a client tunes in at bucket number 4 and wants to retrieve data item 1. After retrieving bucket number 4, the client gets an offset to the nearest root node at bucket number 9. The client then goes into the doze mode waiting for the beginning of bucket number 9. After retrieving bucket number 9, the client knows the position of data item 1 from the first entry in it. The client finally retrieves data item 1 at bucket number 10.

4 Performance

In the literature, Tsakiridis *et al.* [20] proposed the interpolation air index to reduce the tuning time on the uniform data broadcast in a single channel. However, this algorithm assumes that data items are evenly accessed by clients in a broadcast cycle, and was not proposed for the environments with skewed access patterns. Katsaros *et al.* [10] proposed an unbalanced-tree index to handle partially ordered data under skewed access patterns for the uniform data broadcast. However, this algorithm was not proposed for the nonuniform data broadcast. Yao *et al.* [23] proposed a hash algorithm to map data to a broadcast cycle for the nonuniform data broadcast. However, their algorithm suffers from the overflow problem, resulting in the increase in both the access time and the tuning time. Yu and Tan [26] proposed the flexible index (FI) to reduce the tuning time over Broadcast Disks on the nonuniform data broadcast. However, this algorithm does not consider skewed access patterns to further reduce the tuning time. Seifert and Hung [14] proposed the flexible distributed index (FDI) for the nonuniform broadcast. However, this algorithm may replicate too many index trees in one broadcast cycle, resulting in the increase of the average access time.

Among the above algorithms, *FI* [26] and *FDI* [14] have the assumptions of the environments similar to our work. Therefore, to evaluate the effectiveness of our proposed algorithm, we compare our proposed algorithm with Broadcast Disks [1] with no index (*NI*), *FI* [26], and *FDI* [14] via a simulation study. Since *NI* has no index in a broadcast cycle, it has the shortest access time and the longest tuning time among these algorithms. It is used to act as the benchmark for the access time and the tuning time. *FI* splits a sorted list of data items in a minor cycle (segment) into several equal-sized sections. At the beginning of each section, there is a control index consisting of a global index, which points to the upcoming section where the data item can be found, and a local index, which points to the data bucket in the current section where the data item can be found. *FDI* partitions the broadcast program into a number of equal-sized segments with a given bounded tuning time, and interleaves the tree-based index, which handles the following segments, before each segment.

4.1 System Model

The parameters used in our performance model are shown in Table 1. First, we generate N data items with the access probability, $Pr(h)$, $1 \leq h \leq N$, based on the *Zipf* distribution. The *Zipf* distribution is typically used to model nonuniform access patterns. The *Zipf* distribution can be expressed as $Pr(h) = \frac{(1/h)^{\theta}}{\sum_{j=1}^{N}(1/j)^{\theta}}$, $1 \leq h \leq N$, where θ is a parameter named the access skew coefficient or the *Zipf* factor [1]. Different values of θ yield the different *Zipf* distribution. When $\theta = 0$, we have the uniform distribution. When the value of θ increases, the access probabilities become increasingly skewed [3]. For example, when $\theta = 1$ and $N = 3$, we have $Pr(1) = \frac{6}{11}$, $Pr(2) = \frac{3}{11}$, and $Pr(3) = \frac{2}{11}$. Then, the number of data items, K_i, in each disk i, $1 \leq i \leq S$, is assigned by Yee *et al.*'s *GREEDY* algorithm [25]. This algorithm partitions N data items into S disks, so as to minimize the average access time. According to the assignment of this algorithm, K_1 has the fewest number of data items, K_2 has the next fewest number of data items, and K_S has the most number of data items. The relative frequency (R_i) of disk i is determined by $\frac{R_i}{R_S} = (S - i)\Delta + 1$, and $R_S = 1$, $1 \leq i \leq S$, where Δ is the factor for relative frequencies.

When considering the demand access probability (from clients), we also apply the *Zipf*

Table 1: Parameters

Parameter	Description
N	The total number of data items
S	The number of broadcast disks
K_i	The number of data items in disk i, $1 \leq i \leq S$
R_i	The relative frequency of disk i
Δ	The factor for relative frequencies
θ	The *Zipf* factor for generating the access probabilities of data items
γ	The *Zipf* factor for generating access probabilities of disks
η	The ratio of the size of a data node to that of an index node
d	The degree of an index node

distribution with a *Zipf* factor γ. Here, we assume that the probability of accessing any data item within a region is uniform; that is, the *Zipf* distribution is applied to these disks [1]. Therefore, we model the demand access probability of the ith disk ($DiskPr(i)$) using the *Zipf* distribution as follows: $DiskPr(i) = \frac{(1/i)^{\gamma}}{\sum_{j=1}^{S}(1/j)^{\gamma}}$, where γ is the *Zipf* factor. In this case, the first disk (K_1), which has the least number of data items, is the most frequently accessed.

Since the size of a data node (item) is larger than that of an index node, we use η to control the ratio of the size of a data node to that of an index node. That is, if an index node occupies one bucket, then a data node occupies η buckets. Parameter d is used to determine the degree of an index node in our proposed algorithm.

4.2 Performance Analysis

Now we analyze the access time and the tuning time of our proposed algorithm. We assume that the access probability for accessing disk j is $DiskPr(j)$, $1 \leq j \leq S$, and the demand access probabilities of data items in each disk are uniform.

For analyzing the access time of accessing data item i, DIi, there are two cases: (1) It can be downloaded in the current cycle; (2) it cannot be downloaded in the current cycle, but the next cycle. Take data item 3, referred to as $DI3$, in Figure 6 for example. If clients tune in the channel before the third root node, R, at bucket 13, *i.e.,* tuning in at minor cycle m_1 or m_2, they can reach $DI3$ at bucket 22 in the current cycle. This is because in

the current cycle, the third root node at bucket 13 is the last root node that contains the related entry, *i.e.,* $< a1, 15 >$, to reach $DI3$. On the other hand, if they tune in at minor cycle m_3 or m_4, they should wait for the next cycle to reach $DI3$.

Assume that $Dis(a, b)$ represents the distance from the beginning of node a to the end of node b. The access time for accessing $DI3$ is

$$\frac{m_1 + m_2}{TotalSize} \times (\frac{m_1 + m_2}{2} + Dis(R_{3rd}, DI3_{2nd})) + \frac{m_3 + m_4}{TotalSize} \times (\frac{m_3 + m_4}{2} + Dis(R_{1st}, DI3_{1st})),$$

where $TotalSize = \sum_{j=1}^{L} m_j$, R_{ith} represents the ith root node and $DI3_{jth}$ represents the jth $DI3$. The first term is for Case 1, and the second one is for Case 2. Note that, R_{1st} and $DI3_{1st}$ are the first appearing root node and the first appearing $DI3$ in the next cycle, respectively.

In general, there are L minor cycles $(m_1, m_2, \ldots, m_L)$ in one broadcast cycle, where L is the least common multiple of all relative frequencies of disks. For data item DIi with relative frequency λ_z, there would be λ_z regions among the whole broadcast cycle for data item DIi. The first $(\lambda_z - 1)$ regions are for Case 1, and the last one region is for Case 2. Therefore, for data item DIi, its access time is calculated by

$$AT(DIi) = \sum_{j=1}^{\lambda_z} \frac{\sum_{k=(j-1)\times\delta_z+1}^{(\delta_z)\times j} m_k}{TotalSize} \times (\frac{\sum_{k=(j-1)\times\delta_z+1}^{(\delta_z)\times j} m_k}{2} + Dis(R_{(\delta_z \times j+1)th}, DIi_{(j+1)th})),$$

where $\delta_z = \frac{L}{\lambda_z}$. Note that, if the value of $(\delta_z \times j + 1)$ in $R_{(\delta_z \times j+1)th}$ is larger than that of L, $R_{(\delta_z \times j+1)th}$ is $R_{(\delta_z \times j+1-L)th}$ in the next cycle. Similarly, if the value of $(j + 1)$ in $DIi_{(j+1)th}$ is larger than that of λ_z, $DIi_{(j+1)th}$ is data item $DIi_{(j+1-\lambda_z)th}$ in the next cycle. As a result, the average access time is calculated by $AvgAT = \sum_{j=1}^{S} \frac{\sum_{i=1}^{K_j} AT(i)}{K_j} \times DiskPr(j)$, $1 \leq j \leq S$.

Assume that $Path(DIi)$ means the number of index buckets from the root node to data item DIi. For data item DIi, its tuning time is calculated as $TT(DIi) = 1 + Path(DIi) + \eta$, where η is the ratio of the size of a data bucket to that of an index bucket. In this equation, one is for the initial probe. As a result, the average tuning time is calculated as $AvgTT = \sum_{j=1}^{S} \frac{\sum_{i=1}^{K_j} TT(i)}{K_j} \times DiskPr(j)$, $1 \leq j \leq S$.

4.3 Experimental Results

For our proposed skewed index (SI), the default values for the parameters are shown in Table 2. For the flexible index (FI), we set the default values for the number of sections per segment and the number of local index entries to 10 and 60, respectively. For the

Table 2: Default settings

Parameter	Default value
N	10000..12000
S	4
Δ	1
θ	1.0
γ	0.8
η	20
d	10

flexible distributed index (*FDI*), we let the bounded tuning time as close as the tuning time of our proposed *SI*. Moreover, the number of the distinct index tree and the number of the replicated index tree in *FDI* are set according to the equations described in [14]. The following experimental results are the average of 100 cases in which the total number of data items, N, is uniformly chosen from 10000 to 12000. Note that the average access time and the average tuning time are measured in terms of buckets. In this performance evaluation, we will first show the effects of changing five parameters, (a) Δ, (b) S, (c) θ, (d) γ, and (e) η, individually. Next we will show how the degree of an index node affects the performance in our proposed *SI*.

To provide a fairly statistic basis for performance comparison between our proposed algorithm and the compared ones, we present confidence intervals for our experimental results. A confidence interval for a population mean is an interval of values that is likely to contain the true value of the population mean [19]. The 95% confidence interval for the population mean provides a good balance between precision and reliability. Therefore, we present the 95% confidence interval for our experimental results. A 95% confidence interval for the population mean is given by $(\overline{x} - 1.96 \times \frac{s}{\sqrt{ns}}, \overline{x} + 1.96 \times \frac{s}{\sqrt{ns}})$, where $\overline{x}$ is the sample mean, s is the sample standard deviation, and ns is the number of the samples [19]. The first term is called the lower confidence limit and the second is called the upper confidence limit. If the upper confidence limit of the experimental results for our proposed *SI* is less than the lower confidence limit of those for the compared algorithms, we can conclude that *SI* has a statistically significant better performance than the compared algorithms.

In the first experimental result, we vary the value of Δ, which affects the relative fre-

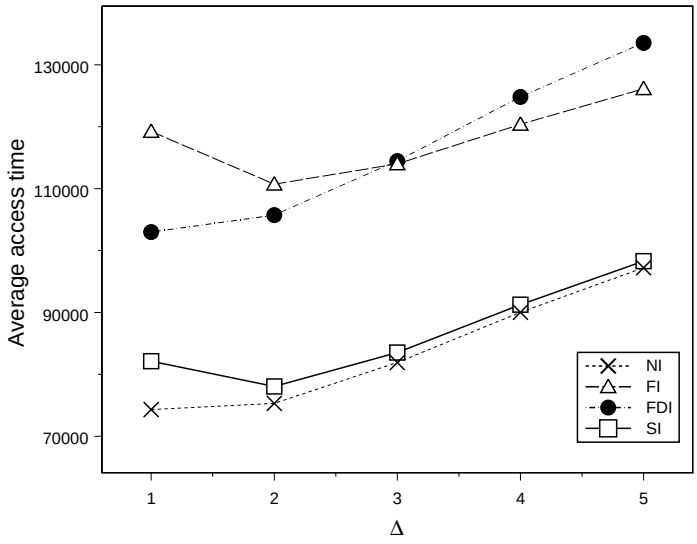


Figure 7: Effect of Δ: the average access time

Table 3: Effect of Δ: the lower confidence limit and the upper confidence limit for the average access time

Δ	*NI_L*	*NI_U*	*FI_L*	*FI_U*	*FDI_L*	*FDI_U*	*SI_L*	*SI_U*
1	73676.74	74978.33	118198.59	120271.17	102094.53	103901.85	81409.76	82847.71
2	74558.15	76041.31	109631.35	111791.42	104687.16	106775.49	77275.01	78811.68
3	81152.26	82704.35	112919.02	115055.78	113375.41	115551.17	82744.03	84325.6
4	89126.35	90938.23	119191.43	121587.2	123481.93	126177.23	90340.04	92174.91
5	96249.82	98173.59	124894.72	127361.27	132226.18	134878.06	97323.07	99265.68

quencies of disks, from 1 to 5, while the other parameters are set to the default values. Figure 7 shows the average access time with the increase of the value of Δ. In Figure 7, x-axis represents the value of Δ, and y-axis represents the average access time. Moreover, Table 3 lists the corresponding lower confidence limit and upper confidence limit for the average access time. *NI_L*, *FI_L*, *FDI_L*, and *SI_L* represent the lower confidence limits for *NI*, *FI*, *FDI*, and *SI*, respectively, under the 95% confidence level. *NI_U*, *FI_U*, *FDI_U*, and *SI_U* represent the upper confidence limits for *NI*, *FI*, *FDI*, and *SI*, respectively, under the 95% confidence level.

As the value of Δ increases, the difference between the relative frequencies of disks becomes larger. This means that the size of the broadcast cycle will increase, resulting

Table 4: Effect of Δ: the average tuning time

Δ	*NI_L*	*NI*	*NI_U*	*FI_L*	*FI*	*FI_U*	*FDI_L*	*FDI*	*FDI_U*	*SI_L*	*SI*	*SI_U*
1	73676.74	74327.53	74978.33	358.27	359.12	359.97	27	27	27	26.57	26.57	26.57
2	74558.15	75299.73	76041.31	396.4	397.29	398.17	27	27	27	26.57	26.57	26.57
3	81152.26	81928.3	82704.35	440.72	441.6	442.47	27	27	27	26.57	26.57	26.57
4	89126.35	90032.29	90938.23	486.14	487.12	488.1	27	27	27	26.57	26.57	26.57
5	96249.82	97211.7	98173.59	530.42	531.43	532.44	27	27	27	26.57	26.57	26.57

in the increase of the average access time, as shown in Figure 7. We can observe that *NI* has the shortest average access time among these four algorithms, as can be also seen in the following experimental results. This is because there is no index used in *NI*. However, using *NI*, mobile devices should constantly tune in to the wireless channel to examine data, consuming a lot of energy. Adding index information to the broadcast cycle will increase the size of the broadcast cycle, resulting in the increase of the average access time. At the same time, it will reduce the average tuning time. In Figure 7, although the average access time of *SI* is longer than that of *NI*, it is shorter than that of *FI* and *FDI*. *SI* has average improvements of 26.72% and 25.35% on the average access time over *FI* and *FDI*, respectively. (Note that the percentage of the improvement from our *SI* to algorithm X is computed as $\frac{(X-SI)}{X} \times 100$.) Moreover, in Table 3, the values of *SI_U* are always less than those of *FI_L* and *FDI_L*. (Note that those values are with an underline.) Therefore, we can conclude that *SI* has a statistically significant shorter access time than *FI* and *FDI*.

Table 4 shows the corresponding average tuning time with the increase of the value of Δ. Since there is no index used in *BD*, clients should tune in to the channel all the time while examining the broadcast data items. Therefore, *NI* has the longest average tuning time among these four algorithms. We can observe that *SI* has the shortest average tuning time among these four algorithms. This is because, in our *SI*, we construct the index tree according to access probabilities of data items. *SI* has average improvements of 99.97%, 93.89% and 1.59% on the average tuning time over *NI*, *FI* and *FDI*, respectively. Therefore, the average tuning time of *SI* is reduced as compared with that of the others. Moreover, the values of *SI_U* are always less than those of *NI_L*, *FI_L* and *FDI_L*. Therefore, we can conclude that *SI* has a statistically significant shorter tuning time than *NI*, *FI* and *FDI*.

In the second experimental result, we vary the value of S, the number of disks, from 4

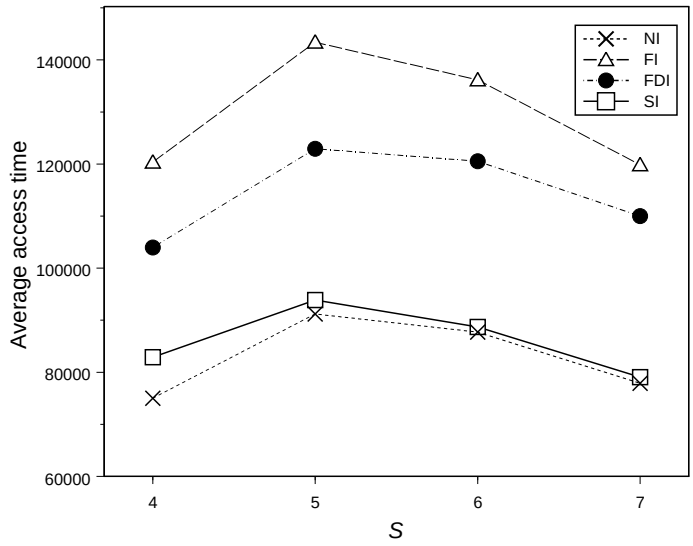


Figure 8: Effect of S: the average access time

to 7, while the other parameters are set to the default values. Figure 8 shows the average access time with the increase of the value of S, in which x-axis represents the value of S, and Table 5 lists the corresponding lower confidence limit and upper confidence limit for the average access time. In Figure 8, as the value of S increases, the average access time decreases except $S = 4$. We can also observe that as the value of S increases, the difference of the average access time between *SI* and *NI* is reduced. In addition, the average access time of *SI* is shorter than that of *FI* and *FDI*. *SI* has average improvements of 33.63% and 24.61% on the average access time over *FI* and *FDI*, respectively. Moreover, in Table 5, the values of *SI_U* are always less than those of *FI_L* and *FDI_L*. Therefore, we can conclude that *SI* has a statistically significant shorter access time than *FI* and *FDI*. Table 6 shows the corresponding average tuning time with the increase of the value of S. Among these four algorithms, *SI* has the shortest average tuning time. *SI* has average improvements of 99.97%, 92.97% and 1.69% on the average tuning time over *NI*, *FI* and *FDI*, respectively. Moreover, the values of *SI_U* are always less than those of *NI_L*, *FI_L* and *FDI_L*. Therefore, we can conclude that *SI* has a statistically significant shorter tuning time than *NI*, *FI* and *FDI*.

Table 5: Effect of S: the lower confidence limit and the upper confidence limit for the average access time

S	NI_L	NI_U	FI_L	FI_U	FDI_L	FDI_U	SI_L	SI_U
4	74231.73	75804.4	119082.32	121586.54	102864.87	105048.67	82022.77	83760.22
5	90396.88	92006.07	142150.94	144665.23	121831.12	124003.22	93056.18	94712.1
6	86865.21	88537.3	134859.87	137437.89	119395.8	121698.62	87859.92	89548.81
7	77031.21	78725.11	118536.94	121122.66	108789.95	111192.4	78206.01	79910.43

Table 6: Effect of S: the average tuning time

S	NI_L	NI	NI_U	FI_L	FI	FI_U	FDI_L	FDI	FDI_U	SI_L	SI	SI_U
4	74231.73	75018.06	75804.4	359	360.02	361.05	27	27	27	26.57	26.57	26.57
5	90396.88	91201.48	92006.07	385.96	386.99	388.02	27	27	27	26.61	26.61	26.61
6	86865.21	87701.25	88537.3	386.07	387.13	388.18	27	27	27	26.65	26.65	26.65
7	77031.21	77878.16	78725.11	377.51	378.57	379.63	27	27	27	26.35	26.35	26.35

In the third experimental result, we vary the value of θ, the $Zipf$ factor for generating the access probabilities of data items, from 0.8 to 1.3, while the other parameters are set to the default values. With the same total number of data items, as the value of θ increases, the access probabilities of data items become increasingly skewed. Therefore, as the value of θ increases, the number of data items in disk 1 decreases and that in the last disk increases by using Yee *et al.*'s algorithm [25] for assigning data items to disks. Figure 9 shows the average access time with the increase of the value of θ, in which x-axis represents the value of θ, and Table 7 lists the corresponding lower confidence limit and upper confidence limit for the average access time. In Figure 9, as the value of θ increases, the average access time decreases. In that figure, the average access time of *SI* is shorter than that of *FI* and *FDI*. *SI* has average improvements of 31.14% and 20.45% on the average access time over *FI* and *FDI*, respectively. Moreover, in Table 7, the values of *SI_U* are always less than those of *FI_L* and *FDI_L*. Therefore, we can conclude that *SI* has a statistically significant shorter access time than *FI* and *FDI*. Table 8 shows the average tuning time with the increase of the value of θ. Among these four algorithms, *SI* has the shortest average tuning time. *SI* has average improvements of 99.96%, 92.63% and 2.59% on the average tuning time over *NI*, *FI* and *FDI*, respectively. Moreover, the values of *SI_U* are always less than those of *NI_L*, *FI_L* and *FDI_L*. Therefore, we can conclude that *SI* has a statistically significant shorter tuning time than *NI*, *FI* and *FDI*.

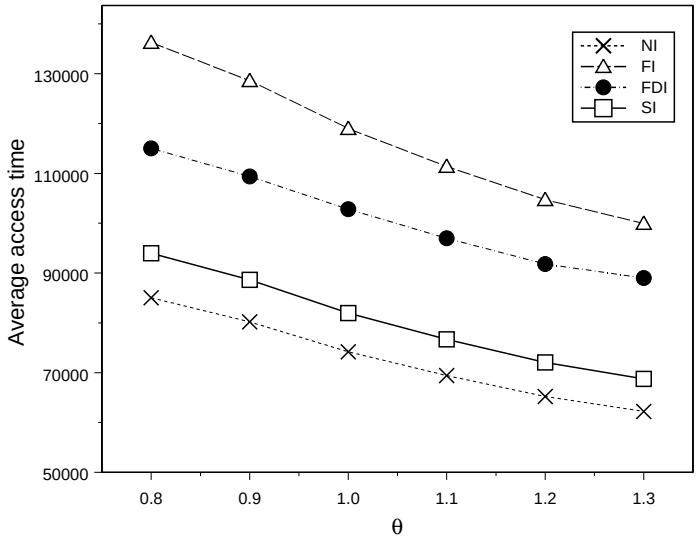


Figure 9: Effect of θ: the average access time

Table 7: Effect of θ: the lower confidence limit and the upper confidence limit for the average access time

θ	NI_L	NI_U	FI_L	FI_U	FDI_L	FDI_U	SI_L	SI_U
0.8	84160.8	85920.76	134892.75	137695.2	113824.42	116206.27	92991.6	94936.15
0.9	79408.8	81009.93	127325.96	129875.5	108282.79	110468.89	87741.87	89510.79
1.0	73419.18	74954.71	117788.48	120233.55	101737.24	103869.36	81125.24	82821.65
1.1	68697.8	70118.53	110270.45	112532.72	95978.61	97969.08	75907.86	77477.46
1.2	64540.49	65877.07	103650.6	105778.9	90871	92759.07	71313.5	72790.28
1.3	61595.93	62848.85	98961.87	100956.95	88105.55	89903.81	68060.61	69444.91

Table 8: Effect of θ: the average tuning time

θ	NI_L	NI	NI_U	FI_L	FI	FI_U	FDI_L	FDI	FDI_U	SI_L	SI	SI_U
0.8	84160.8	85040.78	85920.76	371.95	373.1	374.25	27	27	27	26.57	26.57	26.57
0.9	79408.8	80209.36	81009.93	365.75	366.8	367.84	27	27	27	26.57	26.57	26.57
1.0	73419.18	74186.95	74954.71	357.93	358.94	359.94	27	27	27	26.57	26.57	26.57
1.1	68697.8	69408.16	70118.53	351.77	352.7	353.63	27	27	27	26.32	26.32	26.32
1.2	64540.49	65208.78	65877.07	346.35	347.22	348.09	27	27	27	25.89	25.89	25.89
1.3	61595.93	62222.39	62848.85	342.5	343.32	344.14	27	27	27	25.89	25.89	25.89

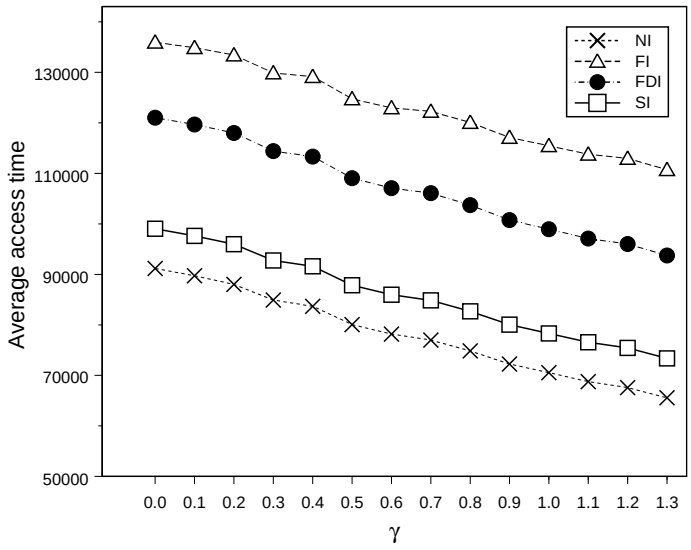


Figure 10: Effect of γ: the average access time

In the fourth experimental result, we vary the value of γ, the *Zipf* factor for generating the access probabilities of disks, from 0.0 to 1.3, while the other parameters are set to the default values. As the value of γ increases, the access probability for data items in disk 1 increases and that in the last disk decreases. Figure 10 shows the average access time with the increase of the value of γ, in which x-axis represents the value of γ, and Table 9 lists the corresponding lower confidence limit and upper confidence limit for the average access time. In Figure 10, as the value of γ increases, the average access time decreases. In that figure, the average access time of *SI* is shorter than that of *FI* and *FDL*. *SI* has average improvements of 30.37% and 19.88% on the average access time over *FI* and *FDI*, respectively. Moreover, in Table 9, the values of *SI_U* are always less than those of *FI_L* and *FDL_L*. Therefore, we can conclude that *SI* has a statistically significant shorter access time than *FI* and *FDL*. Table 10 shows the average tuning time with the increase of the value of γ. Among these four algorithms, *SI* has the shortest average tuning time. *SI* has average improvements of 99.97%, 92.77% and 1.47% on the average tuning time over *NI*, *FI* and *FDI*, respectively. Moreover, the values of *SI_U* are always less than those of *NI_L*, *FI_L* and *FDI_L*. Therefore, we can conclude that *SI* has a statistically significant shorter

Table 9: Effect of γ: the lower confidence limit and the upper confidence limit for the average access time

γ	*NI_L*	*NI_U*	*FI_L*	*FI_U*	*FDI_L*	*FDI_U*	*SI_L*	*SI_U*
0.0	90301.72	92041.2	134656.34	137230.76	119859.41	122172.7	98079.61	99968.59
0.1	88815.14	90629.76	133512.38	136219.95	118446.28	120870.87	96644.55	98618.8
0.2	87117.14	88918.64	132066.31	134777.13	116755.86	119174.63	94981.95	96945.73
0.3	83995.94	85818.85	128467.03	131234.25	113150.51	115611.09	91765.79	93756.96
0.4	82841.82	84499.74	127844.75	130384.44	112193.17	114442.83	90696.96	92511.74
0.5	79258.43	80852.61	123479.51	125944.49	107929.19	110104.44	86965.02	88713.91
0.6	77405.94	78922.93	121752.89	124121.14	106006.57	108088.66	85125.74	86793.75
0.7	76162.65	77813.57	120967.96	123570.67	104914.87	107193.61	83954.58	85774.08
0.8	74078.8	75597.73	118838.81	121257.46	102652.63	104761.98	81853.71	83531.78
0.9	71507.6	73014.08	115886.91	118310.1	99689.68	101794.68	79206.73	80875.16
1.0	69769.7	71288.46	114227.27	116695.24	97864.52	99999.14	77474.51	79160.68
1.1	68144.63	69412.98	112711.6	114793.87	96177.46	97971.29	75860.89	77272.61
1.2	66905.27	68284.41	111791.73	114079.13	95015.35	96977.9	74670.07	76208.95
1.3	64884.06	66286.54	109524.79	111874.63	92714.35	94722.68	72598.27	74167.25

Table 10: Effect of γ: the average tuning time

γ	*NI_L*	*NI*	*NI_U*	*FI_L*	*FI*	*FI_U*	*FDI_L*	*FDI*	*FDI_U*	*SI_L*	*SI*	*SI_U*
0.0	90301.72	91171.46	92041.2	407.24	408.29	409.35	27	27	27	26.75	26.75	26.75
0.1	88815.14	89722.45	90629.76	401.71	402.82	403.93	27	27	27	26.73	26.73	26.73
0.2	87117.14	88017.89	88918.64	395.94	397.05	398.16	27	27	27	26.71	26.71	26.71
0.3	83995.94	84907.39	85818.85	388.44	389.57	390.7	27	27	27	26.69	26.69	26.69
0.4	82841.82	83670.78	84499.74	383.42	384.46	385.5	27	27	27	26.66	26.66	26.66
0.5	79258.43	80055.52	80852.61	375.39	376.4	377.41	27	27	27	26.64	26.64	26.64
0.6	77405.94	78164.44	78922.93	369.61	370.58	371.55	27	27	27	26.62	26.62	26.62
0.7	76162.65	76988.11	77813.57	364.7	365.77	366.83	27	27	27	26.59	26.59	26.59
0.8	74078.8	74838.27	75597.73	358.8	359.79	360.78	27	27	27	26.57	26.57	26.57
0.9	71507.6	72260.84	73014.08	352.35	353.34	354.33	27	27	27	26.54	26.54	26.54
1.0	69769.7	70529.08	71288.46	347.09	348.1	349.12	27	27	27	26.52	26.52	26.52
1.1	68144.63	68778.81	69412.98	342.11	342.96	343.82	27	27	27	26.5	26.5	26.5
1.2	66905.27	67594.84	68284.41	337.77	338.71	339.65	27	27	27	26.47	26.47	26.47
1.3	64884.06	65585.3	66286.54	332.5	333.46	334.43	27	27	27	26.45	26.45	26.45

tuning time than *NI*, *FI* and *FDI*.

In the fifth experimental result, we vary the value of η, the ratio of the size of a data node to that of an index node, from 10 to 50, while the other parameters are set to the default values. Figure 11 shows the average access time with the increase of the value of η, in which x-axis represents the value of η, and Table11 lists the corresponding lower confidence limit and upper confidence limit for the average access time. In Figure 11, as the value of η increases, the average access time increases. For the same size of an index node and the same total number of data items, as the value of η increases, the size of the broadcast cycle increases, resulting in the increase of the average access time. In that figure, the average access time of *SI* is shorter than that of *FI* and *FDL*. *SI* has average improvements of

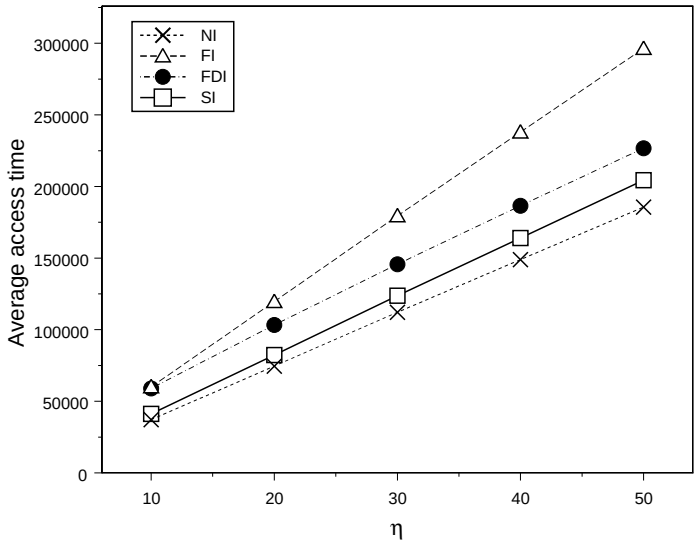


Figure 11: Effect of η: the average access time

Table 11: Effect of η: the lower confidence limit and the upper confidence limit for the average access time

η	*NI_L*	*NI_U*	*FI_L*	*FI_U*	*FDI_L*	*FDI_U*	*SI_L*	*SI_U*
10	36700.2	37515.61	59319.49	60617.89	58353.24	59653.95	40781.01	41686.76
20	73787.85	75257.54	118375.51	120715.76	102248.99	104289.63	81532.41	83156.06
30	111038.35	113274.87	177690.87	181252.17	144170.62	147079.21	122461.92	124928.21
40	147442.97	150342.41	235659.29	240276.17	184706.71	188343.81	162458.59	165653
50	183761.84	187368.51	293491.18	299234.22	224479.83	228891.3	202361.64	206333.07

31.11% and 17.47% on the average access time over *FI* and *FDI*, respectively. Moreover, in Table 11, the values of *SI_U* are always less than those of *FI_L* and *FDL_L*. Therefore, we can conclude that *SI* has a statistically significant shorter access time than *FI* and *FDL*. Table 12 shows the average tuning time with the increase of the value of η. Among these four algorithms, *SI* has the shortest average tuning time. *SI* has average improvements of 99.97%, 91.6% and 1.39% on the average tuning time over *NI*, *FI* and *FDI*, respectively. Moreover, the values of *SI_U* are always less than those of *NI_L*, *FI_L* and *FDI_L*. Therefore, we can conclude that *SI* has a statistically significant shorter tuning time than *NI*, *FI* and *FDI*.

So far, we have shown that *SI* has a better performance than the compared ones. In the

Table 12: Effect of η: the average tuning time

η	*NI_L*	*NI*	*NI_U*	*FI_L*	*FI*	*FI_U*	*FDI_L*	*FDI*	*FDI_U*	*SI_L*	*SI*	*SI_U*
10	36700.2	37107.91	37515.61	300.01	300.54	301.07	17	17	17	16.57	16.57	16.57
20	73787.85	74522.69	75257.54	358.42	359.38	360.33	27	27	27	26.57	26.57	26.57
30	111038.35	112156.61	113274.87	417.04	418.49	419.95	37	37	37	36.57	36.57	36.57
40	147442.97	148892.69	150342.41	474.55	476.44	478.33	47	47	47	46.57	46.57	46.57
50	183761.84	185565.18	187368.51	531.95	534.31	536.66	57	57	57	56.57	56.57	56.57

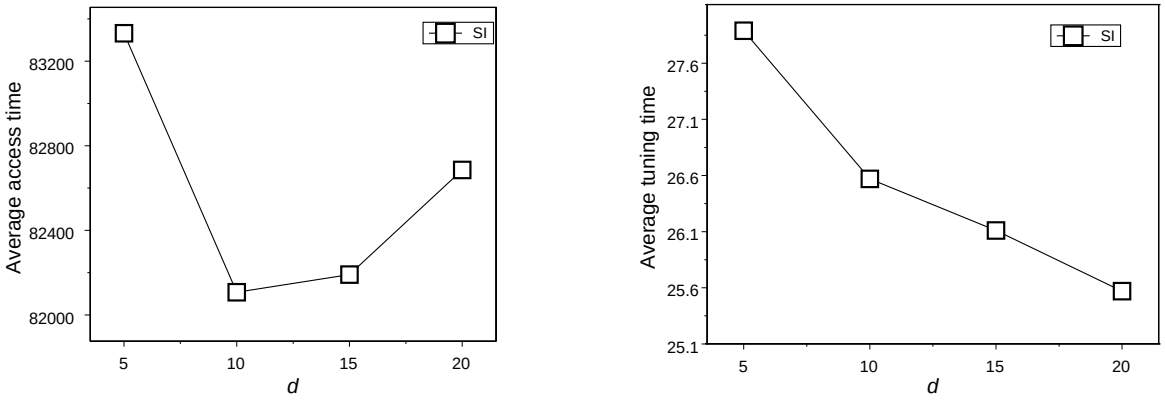


Figure 12: Effect of d: (a) the average access time; (b) the average tuning time.

last experimental result, we evaluate how the degree of an index node affects the average access time and the average tuning time in our proposed *SI*. In this result, we vary the degree of an index node, d, from 5 to 20, while the other parameters are set to the default values. Figure 12-(a) and Figure 12-(b) show the average access time and the average tuning time, respectively, with the increase of the value of d. In Figure 12-(a), SI has the shortest average access time with $d = 10$ under the setting. The average access time is affected by the index allocation so that there is no direct affection by the increase of the value of d. Moreover, as the value of d increases, the more number of index entries in a bucket should be examined. On the other hand, in Figure 12-(b), as the value of d increases, the average tuning time decreases. For the same total number of data items, as the value of d increases, the height of the skewed index tree may decrease, resulting in the decrease of the average tuning time.

5 Conclusions

In this paper, we have proposed a skewed index over Broadcast Disks under skewed access patterns on the nonuniform data broadcast. Our proposed algorithm allocates the index nodes for the popular data items more times than those for the less popular ones in a broadcast cycle. In this way, both the access time and the tuning time can be reduced in the proposed algorithm. From our experimental results, we have shown that the proposed algorithm has average improvements of up to 33.63% and 25.35% on the average access time over the flexible index and the flexible distributed index, respectively. Moreover, the proposed algorithm has average improvements of up to 93.89% and 2.59% on the average tuning time over the flexible index and the flexible distributed index, respectively. How to investigate the index structure for data with skewed access patterns over multiple channels is the possible future work.

References

[1] S. Acharya, M. Franklin, S. Zdonik, and R. Alongso, "Broadcast Disks: Data Management for Asymmetric Communications Environments," *Proc. of the 1995 ACM SIGMOD Int. Conf. on Management of Data*, pp. 199–210, 1995.

[2] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Comm. of the ACM*, Vol. 13, No. 7, pp. 422–426, July 1970.

[3] M. S. Chen, K. L. Wu, and P. S. Yu, "Optimizing Index Allocation for Sequential Data Broadcasting in Wireless Mobile Computing," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 15, No. 1, pp. 161–173, Jan./Feb. 2003.

[4] Y. D. Chung and M. H. Kim, "An Index Replication Scheme for Wireless Data Broadcasting," *The Journal of Systems and Software*, Vol. 51, No. 3, pp. 191–199, May 2000.

[5] Q. L. Hu, W. C. Lee, and D. L. Lee, "Indexing Techniques for Wireless Data Broadcast Under Data Clustering and Scheduling," *Proc. of the 8th Int. Conf. on Information and Knowledge Management*, pp. 351–358, 1999.

[6] J. J. Hung and Y. Leu, "Efficient Index Caching for Data Dissemination in Mobile Computing Environments," *The Journal of Systems and Software*, Vol. 79, No. 1, pp. 93–106, Jan. 2006.

[7] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Power Efficient Filtering of Data on Air," *Proc. of the 4th Int. Conf. on Extending DataBase Technology*, pp. 245–258, 1994.

[8] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Data on Air: Organization and Access," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 9, No. 3, pp. 353–372, May/June 1997.

[9] S. Jung, B. Lee, and S. Pramanik, "A Tree-Structured Index Allocation Method with Replication over Multiple Broadcast Channels in Wireless Environments," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 17, No. 3, pp. 311–325, March 2005.

[10] D. Katsaros, N. Dimokas, and Y. Manolopoulos, "Generalized Indexing for Energy-Efficient Access to Partially Ordered Broadcast Data in Wireless Networks," *Proc. of the 10th Int. Database Eng. and Applications Symp.*, pp. 89–96, 2006.

[11] W. C. Lee and D. L. Lee, "Signature Caching Techniques for Information Filtering in Mobile Environments," *ACM Wireless Networks*, Vol. 5, No. 1, pp. 57–67, Jan. 1999.

[12] S. C. Lo and A. L. P. Chen, "An Adaptive Access Method for Broadcast Data Under an Error-Prone Mobile Environment," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 12, No. 4, pp. 609–620, July/Aug. 2000.

[13] W. C. Peng and M. S. Chen, "Efficient Channel Allocation Tree Generation for Data Broadcasting in a Mobile Computing Environment," *Wireless Networks*, Vol. 9, No. 2, pp. 117–129, March 2003.

[14] A. Seifert and J. J. Hung, "FlexInd: A Flexible and Parameterizable Air-Indexing Scheme for Data Broadcast Systems," *Proc. of the 10th Int. Conf. on Extending Database Technology, LNCS*, Vol. 3896, pp. 902–920, 2006.

[15] J. H. Shen and Y. I. Chang, "A Skewed Distributed Indexing for Skewed Access Patterns on the Wireless Broadcast," *The Journal of Systems and Software*, Vol. 80, No. 5, pp. 711–723, May 2007.

[16] N. Shivakumar and S. Venkatasubramanian, "Efficient Indexing for Broadcast Based Wireless Systems," *ACM/Baltzer Mobile Networks and Applications*, Vol. 1, No. 4, pp. 433–446, Dec. 1996.

[17] K. L. Tan and B. C. Ooi, "On Selective Tuning in Unreliable Wireless Channels," *Data and Knowledge Eng.*, Vol. 28, No. 2, pp. 209–231, Nov. 1998.

[18] K. L. Tan, J. X. Yu, and P. K. Eng, "Supporting Range Queries in a Wireless Environment with Nonuniform Broadcast," *Data and Knowledge Eng.*, Vol. 29, No. 2, pp. 201–221, Feb. 1999.

[19] M. F. Triola, *Elementary Statistics.* Addison Wesley Longman, Inc., 7 ed., 1998.

[20] F. Tsakiridis, P. Bozanis, and D. Katsaros, "Interpolating the Air for Optimizing Wireless Data Broadcast," *Proc. of the 5th ACM Int. Workshop on Mobility Management and Wireless Access*, pp. 112–119, 2007.

[21] J. Xu, W. C. Lee, X. Tang, Q. Gao, and S. Li, "An Error-Resilient and Tunable Distributed Indexing Scheme for Wireless Data Broadcast," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 18, No. 3, pp. 392–404, March 2006.

[22] X. Yang and A. Bouguettaya, "Adaptive Data Access in Broadcast-Based Wireless Environments," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 17, No. 3, pp. 326–338, March 2005.

[23] Y. Yao, X. Tang, E. P. Lim, and A. Sun, "An Energy-Efficient and Access Latency Optimized Indexing Scheme for Wireless Data Broadcast," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 18, No. 8, pp. 1111–1124, Aug. 2006.

[24] W. G. Yee, S. B. Navathe, and E. Omiecinski, "Efficient Data Allocation over Multiple Channels at Broadcast Servers," *IEEE Trans. on Computers*, Vol. 51, No. 10, pp. 1231–1236, Oct. 2002.

[25] W. G. Yee, S. B. Navathe, E. Omiecinski, and C. Jermaine, "Bridging the Gap Between Response Time and Energy-Efficiency in Broadcast Schedule Design," *Proc. of the 8th Int. Conf. on Extending Database Technology*, pp. 572–589, 2002.

[26] J. X. Yu and K. L. Tan, "An Analysis of Selective Tuning Schemes for Nonuniform Broadcast," *Data and Knowledge Eng.*, Vol. 22, No. 3, pp. 319–344, May 1997.

[27] B. Zheng and D. L. Lee, "Information Dissemination via Wireless Broadcast," *Comm. of the ACM*, Vol. 48, No. 5, pp. 105–110, May 2005.

[28] B. Zheng, W. C. Lee, and D. L. Lee, "Spatial Queries in Wireless Broadcast Systems," *Wireless Networks*, Vol. 10, No. 6, pp. 723–736, Nov. 2004.