A Skewed Distributed Indexing for Skewed Access Patterns on the Wireless Broadcast [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: changyi@cse.nsysu.edu.tw}
{Tel: 886-7-5252000 (ext. 4334)}
{Fax: 886-7-5254301}

Abstract

Data broadcast is an efficient way to disseminate information to a large number of mobile clients on the wireless environment. Adding an index data organization to the broadcast file can save client power consumption with little increase in client waiting time. The existing index technologies only consider equal access probabilities of data items. However, in real-life applications, some data items may be more popular than others; that is, access patterns of clients are skewed. In this paper, we propose a skewed distributed indexing, *SDI*, which considers the access probabilities of data items and the replication of index nodes. The proposed algorithm traverses an index tree to determine whether an index node should be replicated by considering the access probability of its child node. In our experimental results, we have shown that our proposed algorithm outperforms the variant-fanout index tree and the distributed indexing.

(**Key Words:** data broadcast, mobile computing, power conservation, selective tuning, wireless network.)

[1]This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-90-2213-E-110-027 and by National Sun Yat-Sen University.

1 Introduction

In recent years, wireless communications have become very popular. The emergence of powerful portable computers, along with advances in wireless communication technologies, has made mobile computing a reality [3]. Although a wireless network with mobile clients is essentially a distributed system, there are some characteristic features that make the system unique and a fertile area of research [7], including asymmetry in communications, frequent disconnections, power limitations and screen size. Each of these features has an impact on how data can be effectively managed in a system with mobile clients [3].

For example, because of asymmetry in communications, there has been considerable interest in delivering information to distributed mobile clients via wireless broadcast [16], including wireless applications using palmtops to access airline schedules, stock activities, traffic conditions and weather information on the road. In the wireless environments, the communication bandwidth from servers to clients is much higher than that from clients to servers. Under such environments, with the limited bandwidth of the wireless channel, using the broadcast technique can serve with large numbers of mobile clients. That is, it is independent of the number of clients tuning to the channel, *i.e., scalability* [4, 5, 25]. By broadcasting the file periodically, mobile clients can specify predefined condition to filter out the data they wanted [1, 2, 27]. Microsoft's smart personal objects technology (SPOT), for example, utilizes the broadcast technique to provide wireless data services [30]. With a wide-area network based on the FM subcarrier technology, SPOT-based devices, *e.g.*, watches, can continuously retrieve timely information such as news, weather, sports, and stocks.

Because of power limits, power conservation is a key issue for the portable units (*e.g.,* palmtops). When a palmtop is listening to the channel, its CPU must be in the *active* mode to examine data packets. This is a waste of energy, since on average, only a very few data packets are of interest to the particular unit. It is definitely beneficial if the palmtop can slip into the *doze* mode most of the time and *wake up* only when the data of interest is expected to arrive [12, 13]. This method is called *selective tuning*. As a consequence, it is advantageous to use some special data organizations, such as tree-based, hash-based and signature-based data organizations, to broadcast data over the wireless channel. In

this way, those mobile units can be guided to the data of interest efficiently and only need to be actively listening to the broadcasting channel when the relevant information is present. As a result, those mobile units can save a lot of power energy while retrieving the relevant information, and lengthen their operating time without recharging. For a file being broadcast on a channel, the following two parameters are of concern [9, 10]: (1) Access time: The average time elapsed from the moment a client wants a record identified by a primary key, to the point when the required record is downloaded by the client. (2) Tuning time: The amount of time spent by a client listening to the channel. This will determine the power consumed by the client to retrieve the required data.

Over the past few years, there have been many strategies for reducing power consumption. For the uniform broadcast in which the same data item appears once in a broadcast cycle, the flexible indexing [13], the hashing-based schemes [13], the tree-based indexing [6, 8, 12], signature schemes [17, 18, 19, 20], the mixture of the index tree and the signature scheme [11], and the method using hashing and index tree techniques [31] have been proposed. A skewed index tree based on data popularity patterns was considered in [7]. In [14], the nonclustered index and multiple indexes were addressed. For energy efficient filtering of nonuniform broadcast in which data records are broadcast according to the access frequency, the studies in [25, 26, 28, 32] proposed indexing schemes. The above schemes considered that there is only one broadcast channel. However, data can be broadcast over multiple channels; therefore, [24] studied allocating index, [21] discussed arranging data, and [15, 23] focused on index and data allocation. The work in [21, 22, 25] concerned on the issue of fault tolerance. The research work on [33, 34] concerned on spatial indexes for supporting spatial queries on the wireless data broadcast.

Since on the wireless broadcast, the access time is affected by the size of the broadcast file, adding the index increases the access time, reducing the tuning time. If the size of the index is too large, the whole broadcast file increases largely, resulting in the increase of the access time. Moreover, if clients miss the corresponding index information to the requested data, the clients have to wait for the next cycle to follow index probes, even though the requested data is not being broadcast yet, *i.e.,* a *directory miss*. In a directory miss, the client cannot get the requested data in one broadcast cycle. Among the strategies for selective tuning,

Chen *et al.*'s variant-fanout (VF) index tree [7] takes the access probabilities of data items into consideration. More popular data may be frequently accessed by the 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.

However, VF assumes that data items are sorted according to access probabilities, and an index tree is constructed according to this sorted order. In real-life applications, the index tree should be constructed according to key values of the data items, not according to access probabilities. Then, clients can efficiently traverse the index tree to get the requested data according to its key value. Moreover, VF does not consider the replication issue of index nodes. That means that clients always have to wait for the next cycle to traverse the index tree to get the requested data, resulting in the increase of the access time. In [12], Imielinski *et al.* proposed the distributed indexing (DI) considering the replication of index nodes. However, DI does not consider the access probability of each data item in a broadcast cycle and always replicates the index nodes of the fixed level. Therefore, in this paper, we propose a skewed distributed indexing, SDI, considering the access probability of each data item and the replication of index nodes to reduce the probability of the directory miss of popular data.

The rest of this paper is organized as follows. In Section 2, we give a brief description of the VF index tree and the distributed indexing. In Section 3, we present our proposed skewed distributed indexing. In Section 4, we study the performance of the proposed algorithm, and make a comparison with the distributed indexing by simulation. Finally, a conclusion is presented in Section 5.

2 Background

In the wireless environments, a broadcast cycle consists of a collection of data items, which are cyclically broadcast on the wireless channel. Mobile clients listen to the wireless channel to retrieve the data item of interest. In this section, we first briefly describe the VF index tree [7], and then the distributed indexing [12], over the broadcast cycle.

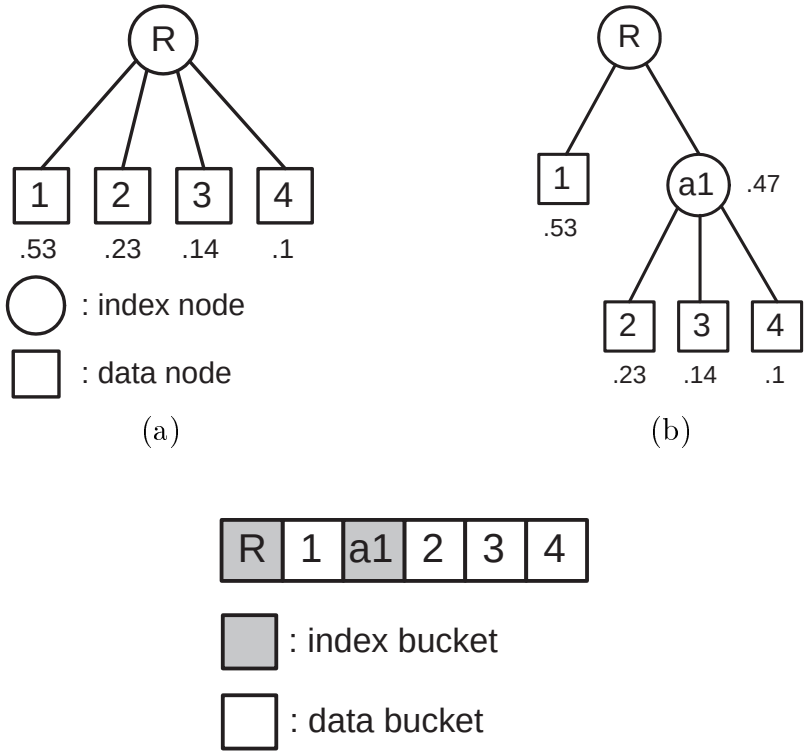


Figure 1: An example of a VF index tree: (a) the original tree; (b) the final VF tree; (c) the corresponding broadcast cycle.

2.1 The VF Index Tree

In [7], Chen *et al.* proposed the variant-fanout (VF) index tree for skewed access patterns over the wireless broadcast. Figure 1 illustrates the VF index tree of four data items. VF assumes that the broadcast data items (i) are sorted according to the descending order of their access probabilities ($Pr(i)$), $1 \leq i \leq 4$, $\sum_{i=1}^{4} Pr(i) = 1$. VF first attaches all data items to the root node, R, as shown in Figure 1-(a). After some evaluation, VF groups nodes with small access probabilities and moves them down to the next level. The evaluation function is $y(k) = (m-k-1)\sum_{i=1}^{k} Pr(i) - \sum_{i=k+1}^{m} Pr(i)$, $1 \leq k \leq m-2$, where k is the position of the child node and m is the degree of the root node. VF finds the maximal value of $y(k)$. If this value is less than or equal to zero, this grouping process is terminated. Otherwise, data items from $k+1$ to m are attached to a new index node, and their access probabilities are aggregated to this node. In Figure 1-(a), the maximal value of $y(k)$, $1 \leq k \leq 2$, is $y(1) = (4-1-1) \times 0.53 - (0.23 + 0.14 + 0.1) = 0.59$. Therefore,

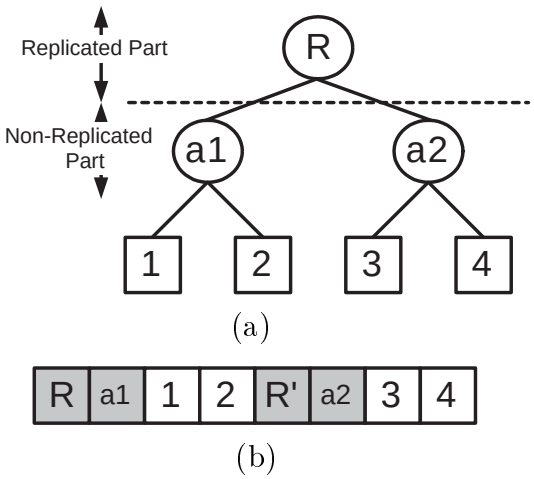


Figure 2: Distributed indexing: (a) an index tree ; (b) the corresponding broadcast cycle.

data items 2-4 are attached to new index node $a1$. Then, the same grouping process is proceeded on the tree rooted by $a1$. Finally, index node $a1$ is attached to the root node R according to descending order of $Pr(i)$, $1 \leq i \leq (k+1) = 2$, and the same grouping process is proceeded on the root node R. Figure 1-(b) shows the final VF tree of Figure 1-(a). The corresponding broadcast cycle of the VF tree is generated by traversing it in preorder, as shown in Figure 1-(c).

2.2 The Distributed Indexing

Now, we describe Imielinski *et al.*'s distributed indexing, DI [12]. Figure 2 illustrates the distributed indexing. An index tree for four data items is shown in Figure 2-(a). The algorithm divides the index tree into two parts: the *replicated part* and the *non-replicated part*. The algorithm replicates only the replicated part (R), and the number of times each node appears in that part equals the number of its children. Moreover, each index node in the replicated part has the *control index* used to direct clients to a proper branch (a higher-level index node) in the index tree. On the other hand, each node in the non-replicated part will appear only once in front of the set of data nodes it indexes. The distributed indexing traverses the index tree and allocates the index nodes and the data nodes to *buckets* in a broadcast cycle. The broadcast cycle of the distributed indexing according to Figure 2-(a) is shown in Figure 2-(b). Index node R is broadcast first. Next, the subtree rooted by index

node $a1$ is traversed in preorder, resulting in $< a1, 1, 2 >$. After that, since the root node R is in the replicated part, this node is broadcast again. Furthermore, traversal sequence $< a2, 3, 4 >$ of the subtree rooted by index node $a2$ is appended to the broadcast cycle. Each data bucket contains the offset to the nearest-replicated index bucket.

3 Skewed Distributed Indexing

The *VF* index tree does not consider the replication of index nodes, resulting in the directory miss when clients tune into the broadcast channel to retrieve data. This will increase the access time. The distributed indexing always determines the replicated part of an index tree by the fixed level. However, when some data nodes are more popular than the others, *i.e.*, data nodes with different access probabilities, the access time may be improved if we replicate the index nodes different times according to their different access probabilities. In this section, we first state the assumptions of our proposed algorithm, and then present the proposed algorithm, the *skewed distributed indexing*, *SDI*.

3.1 Assumptions

This paper focuses on the wireless environment. Some assumptions should be restricted in order to make our work feasible [5]. These assumptions include:

1. Data appears once in the whole broadcast file, *i.e.,* the uniform broadcast.
2. Data is read-only; there are no updates either by the clients or at the servers.
3. 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 one or more buckets, the *data bucket*.
4. Clients make no use of their upstream communications capability; that is, they provide no feedback to servers.
5. When a client switches to the public channel, it can retrieve buckets immediately. The delay for hardware and software preparation to begin monitoring the broadcast channel is short.

6. The server broadcasts buckets over a single channel. All clients retrieve buckets from this single channel.

7. The wireless channel is reliable; that is, clients receive *correct* data and do not miss their data.

3.2 The Proposed Algorithm

Now, we present our proposed algorithm, the *skewed distributed indexing*, which replicates the index nodes by considering the access probabilities of data nodes. Assume that the root node is of level zero, its children are of level one, and so on. In the proposed algorithm, the following variables are used:

1. n: the total number of data items in a broadcast cycle.

2. $Pr(i)$: the access probability of each data node i, $1 \leq i \leq n$, $\sum_{i=1}^{n} Pr(i) = 1$.

3. d: the degree of an index node.

4. h: the depth of an index tree.

5. l: the level of an index tree, $0 \leq l < h$.

6. μ_l: the threshold for each level l, *i.e.*, $\mu_l = \frac{1}{d^l}$.

The proposed algorithm is processed as follows.

1. Accumulate the access probabilities of the child nodes to their parent nodes in an index tree in a bottom up manner.

2. Set the root node of the index tree to be replicated.

3. Traverse the index tree in preorder.

   (a) If the access probability of an index node or a data node is greater than μ_l, set the parent node of the current node to be replicated.

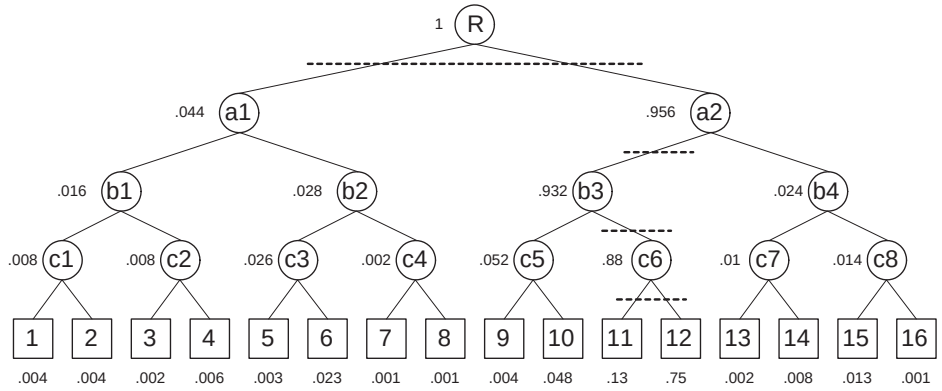
4. Call **procedure** *Mapping*(the *root* node), as shown in Figure 3.

```
1: procedure Mapping(v)
2:     if the parent node of v is set to be replicated then
3:         Put the corresponding control index into the parent node of v
4:         if the last broadcast node is not the same as the parent node of v then
5:             Broadcast the parent node of v
6:         end if
7:     end if
8:     Broadcast v
9:     for all c ∈ the children of v do
10:        Call procedure Mapping(c)
11:    end for
12: end procedure
```

Figure 3: Procedure *Mapping*

Take Figure 4 for example. The access probability ($Pr(i)$) of each data node i is labeled under it, and the sum of the access probabilities of all of data nodes could be normalized to equal 1. In Step 1, the access probabilities of the child nodes are accumulated to the corresponding parent nodes, as shown in Figure 4-(a). (Node that Figure 4-(b) also lists the access probability of each node.) When a directory miss of the root node (R) occurs, the client has to wait for the next cycle to traverse the index buckets. If the root node is replicated, the probability of the directory miss in this cycle could be reduced. Therefore, in Step 2, the root node in the index tree is set to be replicated. The dotted line under the root node in Figure 4-(a) represents that the root node is in the replicated part; that is, index node R will appear just before index nodes $a1$ and $a2$ in a broadcast cycle.

In Step 3, the algorithm traverses the index tree in preorder to determine the replicated index node. Since the access probability ($= 0.956$) of index node $a2$ is greater than μ_1 ($= 1/2^1 = 0.5$), the parent node (R) of index node $a2$ is set to be replicated before it. But in this case, index node R has already been set to be replicated in Step 2. When index node $b3$ is traversed, the parent node ($a2$) of index node $b3$ is set to be replicated before it, since the access probability ($= 0.932$) of $b3$ is greater than μ_2 ($= 1/2^2 = 0.25$). The dotted line between index nodes $a2$ and $b3$ in Figure 4-(a) depicts this replicated information. The final result of Step 3 is shown in Figure 4-(a). (Note that Figure 4-(b) lists μ_1, μ_2, μ_3 and μ_4 for the index tree shown in Figure 4-(a).)

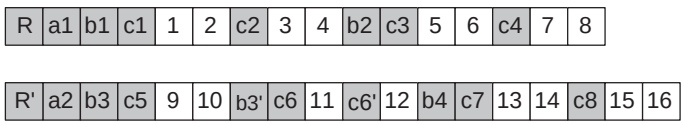


(a)

4th	*prob.*	μ_4	3rd	*prob.*	μ_3	2nd	*prob.*	μ_2	1st	*prob.*	μ_1
1	0.004		$c1$	0.008							
2	0.004					$b1$	0.016				
3	0.002		$c2$	0.008							
4	0.006								$a1$	0.044	
5	0.003		$c3$	0.026							
6	0.023					$b2$	0.028				
7	0.001		$c4$	0.002							
8	0.001	$\frac{1}{2^4}$			$\frac{1}{2^3}$			$\frac{1}{2^2}$			$\frac{1}{2^1}$
9	0.004		$c5$	0.052							
10	0.048					$b3$	0.932				
11	0.13		$c6$	0.88							
12	0.75								$a2$	0.956	
13	0.002		$c7$	0.01							
14	0.008					$b4$	0.024				
15	0.013		$c8$	0.014							
16	0.001										

* *prob.*: the access probability
* 1st: the first level
* 2nd: the second level
* 3rd: the third level
* 4th: the fourth level

(b)



(c)

Figure 4: Skewed distributed indexing: (a) an index tree; (b) the access probability of each node and the threshold of each level; (c) the corresponding broadcast cycle.

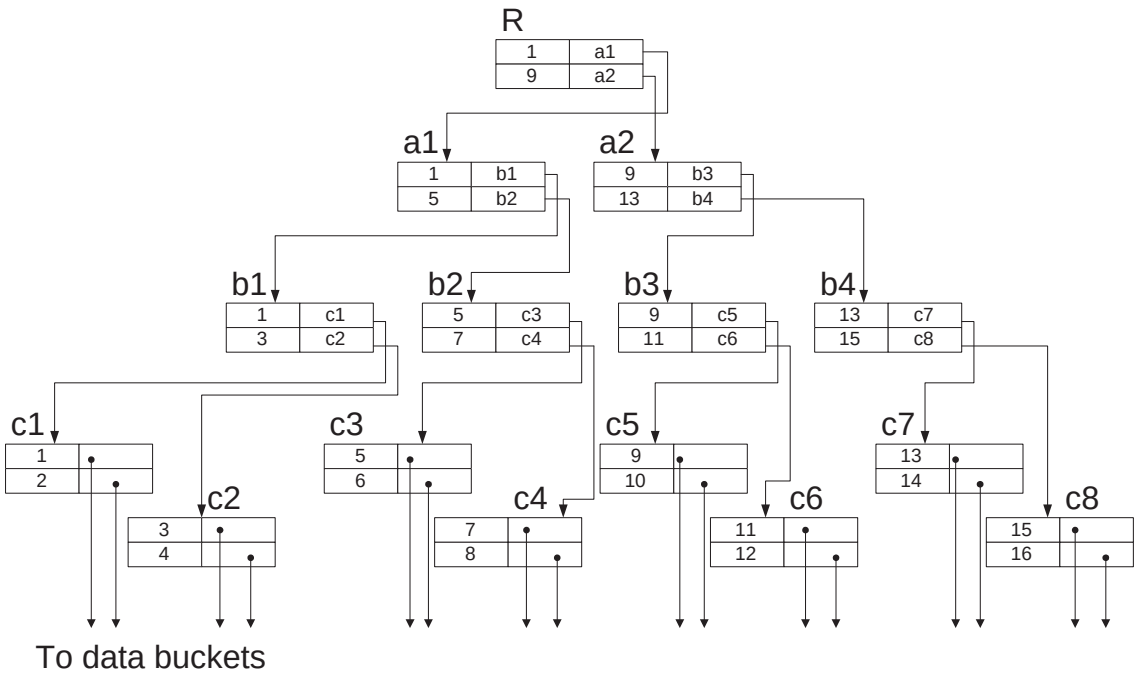


Figure 5: Tuples in each index node

After setting the replicated information, the algorithm calls **procedure** *Mapping*, as shown in Figure 3, with the parameter, the root node R, to generate the broadcast cycle. In **procedure** *Mapping*, the nodes of an index tree are allocated in preorder. When the node is allocated, if its parent node is set to be replicated and not the same as the last broadcast node, the parent node will be allocated before this node. Note that, in [7], Chen *et al.* mentioned that, every index node is always broadcast immediately before its child nodes so that the extra access time of other nodes, which is incurred due to the presence of this index node, is minimized. Therefore, **procedure** *Mapping* follows this principle. The broadcast cycle corresponding to the index tree in Figure 4-(a) is shown in Figure 4-(c), tuples in each index node are shown in Figure 5. Each tuple of index nodes is of form $< K, ptr >$, where K is a key value and ptr is an offset to the next index or data node. If the key value K of the requested data item is greater than or equal to the key value of the current tuple and less than the key value of the following tuple, clients should follow the ptr of the current tuple.

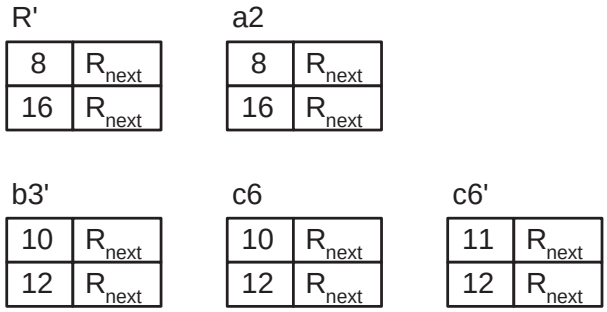


Figure 6: The control indexes in the replicated index nodes

3.3 Control Index

Each non-replicated index node and each data node have an address to its nearest replicated index node of the following broadcast [12]. This helps clients traverse a sequence of index probes. Each replicated index node has the control index, which can direct clients to go to the proper index node. Moreover, in the replicated index node, tuples of the control index are allocated first, and then the original index tuples. The first tuple in the control index contains key S of the latest broadcast data item. That is, if the value of key K of the requested data item is greater than that of S ($K > S$), the requested data item was already broadcast and the client has to wait for the beginning of the next broadcast cycle. The remaining tuples contain the largest key L that is covered by the current index node. If the value of key K of the requested data item is greater than that of L, the client will be directed to the higher level index node. If the value of key K of the requested data item is less than that of L and greater than that of S, the client will follow the entry of the current index node. The control indexes of the replicated index nodes in Figure 4 are shown in Figure 6, where R_{next} denotes the root index node in the next cycle.

3.4 Access Protocol

We now present the access protocol of the proposed algorithm. Assume that a data item with key K is requested. The access protocol is as follows [12].

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

2. Read the current bucket to get the offset of the nearest replicated index node which

contains the control index, and go into the doze mode.

3. Tune in to the broadcast channel receiving the nearest replicated index node.

   (a) If the control index indicates that the data item with key K was broadcast, go into the doze mode until the beginning of the next broadcast cycle.

   (b) If the control index does not have information about the data item with key K, go to the higher level index node that contains the control index.

   (c) If the control index indicates that the current index node can direct clients to get the data item with key K, proceed as in Step 4.

4. Follow the sequence of index probes to obtain the data items with key K.

Consider the broadcast cycle as shown in Figure 4-(c) for example. When tuning in at the beginning of data bucket 9, a client wants to retrieve data bucket 12. From data bucket 9, the client gets the offset to the nearest index bucket that has the control index, *i.e.,* index bucket $b3'$. After retrieving index bucket $b3'$, the client knows that data bucket 12 is covered by the current index bucket through the second tuple (as shown in Figure 6) of the control index. (Note that if an index node does not contain the control index to indicate the index range, it cannot direct clients to follow the remaining index probes.) Therefore, the client then gets the offset to index bucket $c6$, and finally retrieves data bucket 12. The traversal sequence of this example in our skewed distributed indexing is $< 9, b3', c6, 12 >$. The result of the distributed indexing for the same example is shown in Figure 7, and the traversal sequence in the distributed indexing is $< 9, R_{next}, a2, b3, c6, 12 >$. Since the access probability of data bucket 12 is high, reducing its probability of the directory miss will decrease the average access time and tuning time. In this case, the proposed algorithm can retrieve the data of a high access probability in the same cycle, shortening the access time and tuning time.

4 Performance

In this section, we study the performance of the proposed algorithm. We first compare our proposed algorithm with *VF* [7]. The three variations of the distributed indexing [8]

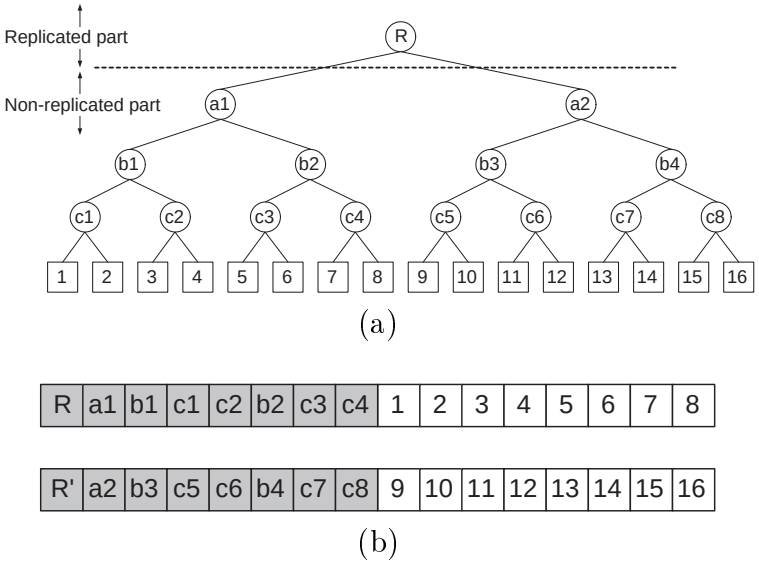


Figure 7: Distributed indexing: (a) an index tree; (b) the corresponding broadcast cycle.

have a little improvement on the tuning time, and the access time is the same as that in the original distributed indexing. Therefore, in the experimental results, we have only compared our proposed algorithm with the distributed indexing [12].

4.1 The System Model

The parameters used in our performance model are shown in Table 1. Given n, the total number of data items, we generate n data items with the access probability, $Pr(i)$, $1 \leq i \leq n$, based on the $Zipf$ distribution. The $Zipf$ distribution is typically used to model nonuniform access patterns. The $Zipf$ distribution can be expressed as $Pr(i) = \frac{(1/i)^\theta}{\sum_{j=1}^{n}(1/j)^\theta}$, $1 \leq i \leq n$, where θ is a parameter named access skew coefficient or $Zipf$ factor. 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 [7]. For example, when $\theta = 1$ and $n = 3$, we have $Pr(1) = \frac{6}{11}$, $Pr(2) = \frac{3}{11}$, and $Pr(3) = \frac{2}{11}$.

Since the size of an index node is smaller than that of the data page, parameter R_{DI} represents the ratio of the size of the data page to that of the index node. That is, if the size of an index node is 1, that of the data page is R_{DI}. In our simulations, we collect the

Table 1: Parameters

Parameter	Description
n	The total number of data items
R_{DI}	The ratio of the size of the data page to that of the index node
θ	The $Zipf$ factor

experimental results for 100 executions.

4.2 Analysis of Access Time and Tuning Time

For simplicity, we assume that a full index tree is built and that the clients tune in at the beginning of each bucket. Since the size of an index node is smaller than that of a data node, we assume that the ratio of the index node to the data node is $1 : R_{DI}$. That is, if the index node occupies one bucket in the broadcast cycle, the data node will occupy R_{DI} contiguous buckets. Each data bucket of R_{DI} contiguous buckets contains the offset to the nearest-replicated index node that is not broadcast yet. Therefore, if the initial probe of the clients is in the data bucket, they do not need to retrieve all R_{DI} contiguous buckets to get the offset to the nearest-replicated index node; that is, they retrieve only one data bucket. In the following discussion, we measure the access time and the tuning time in terms of buckets.

The control index in the replicated index nodes can direct the clients to reach the data node that does not pass over. If the clients miss the corresponding replicated index node, they have to wait for the next cycle to get the data bucket of interest. For the analysis of the access time, there are two cases: (1) The clients tune in before the corresponding nearest-replicated index node to the wanted data node; (2) the clients tune in after the corresponding nearest-replicated index node. In the first case, the clients can retrieve the data node of interest in the same cycle; in the second case, the clients have to wait for the next cycle to retrieve that data node.

Assume that $Distance(i, j)$ means the distance from the beginning of node i to the end of node j. For Case 1, the access time, $AT_{t,w}$, from the initial probe bucket, t, to the wanted data node, w, is $Distance(t, w)$, as shown in Figure 8. For Case 2, the access

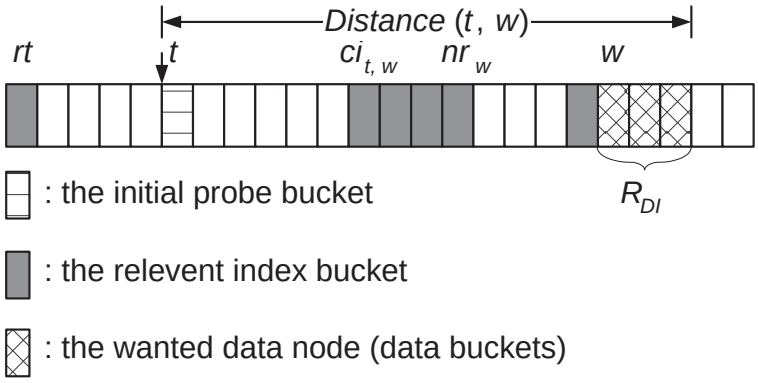


Figure 8: Case 1: The client tunes in before the corresponding nearest-replicated index node, nr_w, to the wanted data node, w.

time, $AT_{t,w}$, is $Distance(t, e) + Distance(b, w)$, where b represents the beginning bucket in a broadcast cycle, and e represents the end bucket, as shown in Figure 9. Since an index node occupies one bucket and a data node occupies R_{DI} contiguous buckets, the probabilities of the initial probes in the index node and the data node are different. If the initial probe is in the index node, the fraction of the average access time, $fAT_{t,w}$, for the wanted data node, w, is $AT_{t,w} \times \frac{1}{BC}$, where BC is the size of the broadcast cycle; otherwise, $fAT_{t,w} = AT_{t,w} \times \frac{R_{DI}}{BC}$. Assuming that $rIndex$ is the total number of the replicated index nodes, we have $BC = rIndex + \frac{d^{h-1} - 1}{d - 1} + d^{h-1} \times R_{DI}$, where d is the degree of an index node and h is the depth of an index tree. The second term represents the total number of index nodes (buckets) in the original index tree, and the third term represents the total number of data buckets. Let S_w represent the set of the initial probes that do not exceed the corresponding nearest-replicated index node, nr_w, to the wanted data node, w, and T_w represent the set of the initial probes that exceed it. The average access time, AAT_w, for the wanted data node, w, is $\sum_{t \in S_w} fAT_{t,w} + \sum_{t \in T_w} fAT_{t,w}$. Let D represent the set of all data nodes in the broadcast cycle. The average access time for the whole broadcast cycle is $\sum_{\forall w \in D} Pr(w) \times AAT_w$.

Assume that $Path(i, j)$ means the set of index nodes from node i to node j in an index tree, not including the data node. Let rt be the root node in the index tree, and nr_w be the nearest-replicated index node to the wanted data node, w. We have the set, U, of

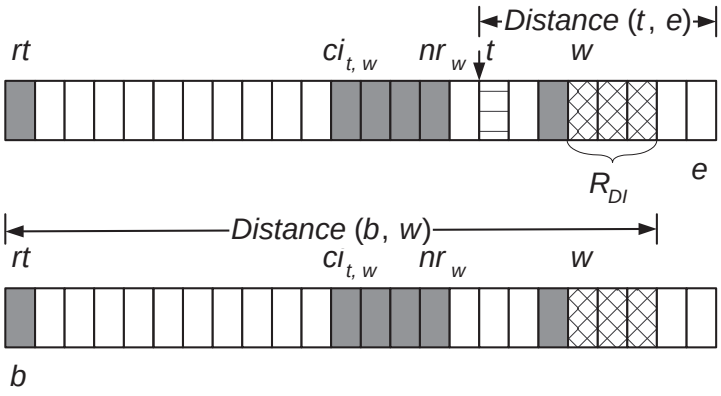


Figure 9: Case 2: The client tunes in after the corresponding nearest-replicated index node, nr_w, to the wanted data node, w.

the index nodes from the root node, rt, to the nearest-replicated index node, nr_w, to the wanted data node, w, *i.e.*, $U = Path(rt, nr_w)$. Let $ci_{t,w}$ be the closest-replicated index node in U that is not passed over in the broadcast cycle, from the initial probe bucket, t, to the wanted data node, w. Figure 8 depicts these defined variables. For the analysis of the tuning time, similar to the analysis of the access time, there are two cases. For Case 1, the tuning time, $TT_{t,w}$, from the initial probe bucket, t, to the wanted data node, w, is $1 + |Path(ci_{t,w}, w)| + R_{DI}$. For Case 2, the tuning time, $TT_{t,w}$, from the initial probe bucket, t, to the wanted data node, w, is $1 + h + R_{DI}$.

If the initial probe is in the index bucket, the fraction of the tuning time, $fTT_{t,w}$, for the wanted data node, w, is $TT_{t,w} \times \frac{1}{BC}$; otherwise, $fTT_{t,w} = TT_{t,w} \times \frac{R_{DI}}{BC}$. The average tuning time, ATT_w, for the wanted data node, w, is $\sum_{t \in S} fTT_{t,w} + \sum_{t \in T} fTT_{t,w}$. Therefore, the average tuning time for the whole broadcast cycle is $\sum_{\forall w \in D} Pr(w) \times ATT_w$.

Assume that an index node and a data node are allocated to one bucket, respectively. Following the analysis of the access time and tuning time mentioned above, Table 2 summarizes the average access time and the average tuning time for the whole broadcast cycle in Figures 7-(b) and 4-(c). In Table 2, the percentage in parentheses represents the improvement percentage of the corresponding algorithm as the distributed indexing is the baseline. It is clear that our proposed skewed distributed indexing could provide the better performance on the average access time and the average tuning time for the whole broadcast

Table 2: Comparison of the average access time and the average tuning time

Algorithm	Average access time	Average tuning time
DI	28.357	5.97
SDI	20.383 (28%)	5.428 (9%)

* *DI*: the distributed indexing
* *SDI*: the skewed distributed indexing

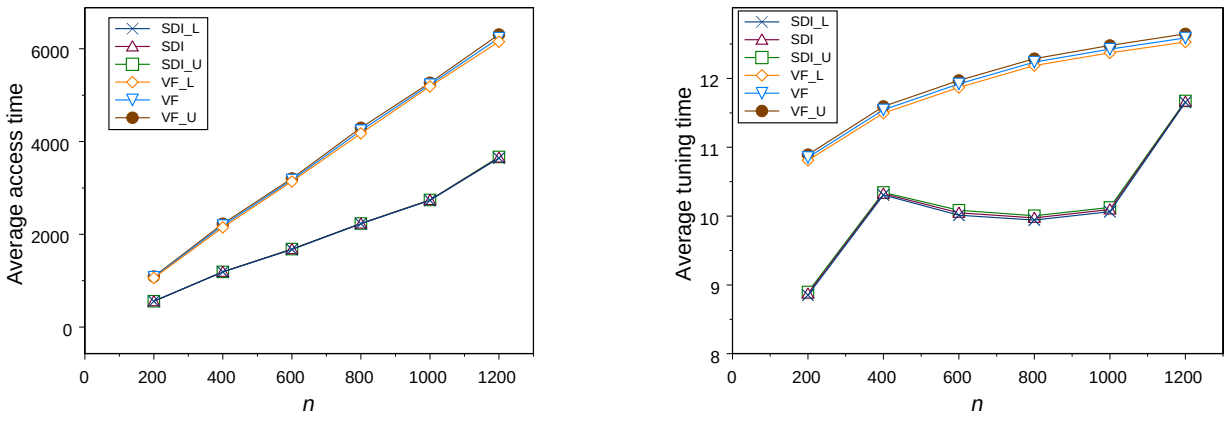
cycle than the distributed indexing.

4.3 Simulation Results: SDI *vs.* VF

Since the data items for *VF* should be arranged in the descending order of access probabilities, we generate n data items with their access probabilities of the descending order based on the *Zipf* distribution. For our proposed algorithm, we build an index tree of at most degree d for these n data items. Our proposed algorithm then traverses the index tree to determine which index node should be replicated according to access probabilities of its child nodes. For *VF*, it dynamically adjusts an index tree according to access probabilities of index or data nodes.

To provide a fairly statistic basis for performance comparison between our proposed algorithm and the compared one, 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 [29]. 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 [29]. 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 *SDI* is less than the lower confidence limit of these for the compared algorithm, we can conclude that *SDI* has a statistically significant better performance than the compared algorithm.

Furthermore, to prove statistically significant differences of the experimental results



(a) (b)

Figure 10: A comparison of SDI and VF with increasing the number of data items: (a) the average access time; (b) the average tuning time.

between our proposed algorithm and the compared one, we present an analysis of variance (ANOVA). ANOVA is a method of testing the equality of population means by analyzing sample variances [29]. If sample means that are close in value result in an F test statistic that is close to 1, we conclude that there is no significant difference among the sample means. On the other hand, if the value of F is excessively large, then we reject the clam of equal means. In our experimental results, we use a 5% significant level; that is, the corresponding critical value of F, $F_{0.05}$, is 3.84 [29]. If the observed value of F is greater than the value of $F_{0.05}$, we conclude that there is sufficient evidence to reject the clam of equal means of our proposed algorithm and the compared one. That is, there is a significant difference between the mean of our proposed algorithm and that of the compared one. Otherwise, we fail to reject the clam.

For the first experiment, we increase the total number of data items, n, from 200 to 1200 under $d = 4$ and $R_{DI} = 5$. Moreover, for each fixed value of n, we randomly pick 100 samples under $\theta = [0.8..1]$; that is, the experimental result is an average of these 100 samples. Figure 10-(a) shows the experimental result of the average access time, and Table 3 lists its corresponding details of the figures. In Figure 10-(a), the x-axis represents the number of data items in one broadcast cycle, and the y-axis represents the average access time in terms of buckets. In Figure 10-(a), *SDI_L* and *SDI_U* represent the lower confidence

Table 3: The average access time for the cases of increasing the number of data items

n	*SDI_L*	*SDI*	*SDI_U*	*VF_L*	*VF*	*VF_U*
200	557.416	557.745	558.073	1059.342	1074.641	1089.941
400	1187.448	1189.519	1191.589	2148.377	2188.314	2228.25
600	1674.432	1677.7	1680.969	3138.693	3171.228	3203.764
800	2226.986	2230.329	2233.672	4176.4	4236.891	4297.381
1000	2732.492	2737.169	2741.847	5187.916	5228.216	5268.517
1200	3644.478	3655.89	3667.303	6153.217	6231.154	6309.09

Table 4: The values of F of the average access time for the cases of increasing the number of data items

n	200	400	600	800	1000	1200
F	4382.951	2396.352	8014.341	4214.206	14482.474	4106.362

limit and the upper confidence limit for *SDI* under the 95% confidence level, respectively. That is, the values of *SDI_L* and *SDI_U* indicate the confidence intervals of the values of *SDI*. *VF_L* and *VF_U* represent the lower confidence limit and the upper confidence limit for *VF* under the 95% confidence level, respectively. That is, the values of *VF_L* and *VF_U* indicate the confidence intervals of the values of *VF*. As the value of n increases, the average access time of both our proposed *SDI* and *VF* increases. We can observe that our proposed *SDI* outperforms *VF* in terms of the average access time under all of the cases. This is because *VF* does not replicate index nodes resulting in the directory miss. Moreover, the values of *SDI_U* are always less than those of *VF_L*. Furthermore, all the corresponding values of F shown in Table 4 are extremely greater than the value of $F_{0.05}(= 3.84)$. Therefore, we can conclude that *SDI* has a statistically significant shorter access time than *VF*.

Figure 10-(b) shows the experimental result of the average tuning time, and Table 5 lists its corresponding details of the figures. In Figure 10-(b), the x-axis represents the number of data items in one broadcast cycle, and the y-axis represents the average tuning time in terms of buckets. We can observe that the average tuning time of our proposed algorithm is shorter than that of *VF*. Moreover, the values of *SDI_U* are always less than those of *VF_L*.

Table 5: The average tuning time for the cases of increasing the number of data items

n	*SDI_L*	*SDI*	*SDI_U*	*VF_L*	*VF*	*VF_U*
200	8.851	8.874	8.897	10.812	10.852	10.892
400	10.306	10.325	10.344	11.495	11.544	11.594
600	10.012	10.048	10.084	11.869	11.92	11.971
800	9.942	9.974	10.006	12.187	12.237	12.287
1000	10.065	10.096	10.126	12.372	12.426	12.48
1200	11.652	11.662	11.673	12.53	12.589	12.647

Table 6: The values of F of the average tuning time for the cases of increasing the number of data items

n	200	400	600	800	1000	1200
F	7033.342	2037.391	3407.423	5548.328	5409.219	932.173

Furthermore, all the corresponding values of F shown in Table 6 are extremely greater than the value of $F_{0.05}(= 3.84)$. Therefore, we can conclude that *SDI* has a statistically significant shorter tuning time than *VF*.

For the second experiment, we increase the *Zipf* factor θ from 0.6 to 0.9 under $d = 4$ and $R_{DI} = 5$. For each fixed value of θ, we randomly pick 100 samples under $n = [1000..1100]$; that is, the experimental result is an average of these 100 samples. As the value of θ increases, access patterns become more skewed. Figure 11-(a) shows the experimental result of the average access time, and Table 7 lists its corresponding details of the figures. In Figure 11-(a), the x-axis represents the value of the *Zipf* factor θ, and the y-axis represents the average access time. We can observe that the average access time of our proposed algorithm is shorter than that of *VF*. Moreover, the values of *SDI_U* are always less than those of *VF_L* under the 95% confidence level. Furthermore, all the corresponding values of F shown in Table 8 are extremely greater than the value of $F_{0.05}(= 3.84)$. Therefore, we can conclude that *SDI* has a statistically significant shorter access time than *VF*.

Figure 11-(b) shows the corresponding average tuning time, and Table 9 lists its details of the figures. In Figure 11-(b), the x-axis represents the value of the *Zipf* factor θ, and the y-axis represents the average tuning time. We can observe that the average tuning

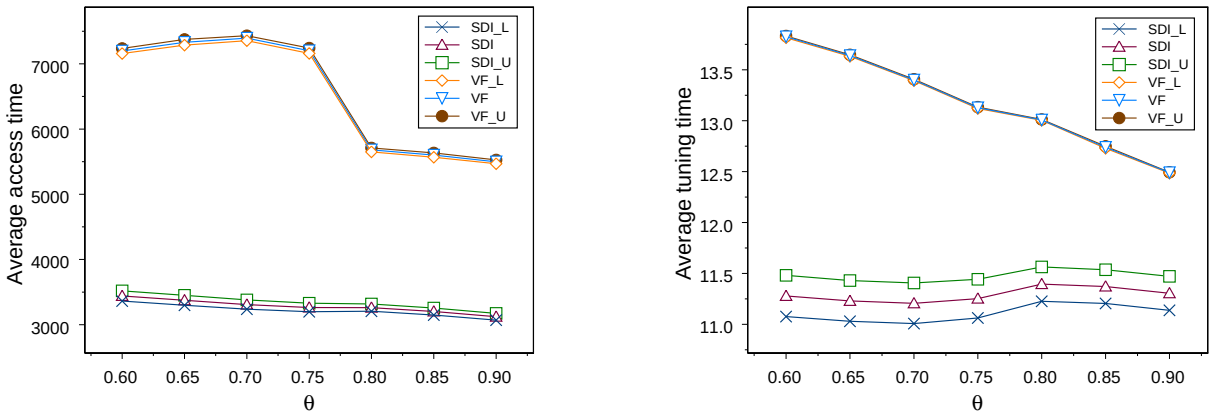


Figure 11: A comparison of SDI and VF with increasing the value of θ: (a) the average access time; (b) the average tuning time.

Table 7: The average access time for the cases of increasing the value of θ

θ	*SDI_L*	*SDI*	*SDI_U*	*VF_L*	*VF*	*VF_U*
0.6	3363.575	3440.777	3517.979	7158.904	7197.816	7236.728
0.65	3297.607	3374.157	3450.707	7287.141	7330.658	7374.175
0.7	3238.029	3308.802	3379.574	7354.934	7393.653	7432.372
0.75	3198.392	3264.053	3329.715	7160.913	7201.639	7242.365
0.8	3206.383	3262.205	3318.026	5650.65	5680.901	5711.153
0.85	3147.798	3201.831	3255.864	5568.705	5601.166	5633.627
0.9	3070.17	3121.827	3173.484	5467.566	5497.455	5527.344

Table 8: The values of F of the average access time for the cases of increasing the value of θ

θ	0.6	0.65	0.7	0.75	0.8	0.85	0.9
F	7254.973	7755.937	9849.723	9976.879	5574.966	5566.040	6086.928

Table 9: The average tuning time for the cases of increasing the value of θ

θ	*SDI_L*	*SDI*	*SDI_U*	*VF_L*	*VF*	*VF_U*
0.6	11.077	11.279	11.481	13.815	13.824	13.832
0.65	11.03	11.23	11.43	13.632	13.64	13.647
0.7	11.008	11.207	11.406	13.394	13.4	13.407
0.75	11.063	11.253	11.443	13.12	13.127	13.134
0.8	11.226	11.395	11.564	13.003	13.007	13.012
0.85	11.205	11.371	11.537	12.729	12.739	12.748
0.9	11.138	11.305	11.472	12.485	12.49	12.494

Table 10: The values of F of the average tuning time for the cases of increasing the value of θ

θ	0.6	0.65	0.7	0.75	0.8	0.85	0.9
F	606.838	556.844	466.452	373.748	350.674	259.078	193.273

time of our proposed algorithm is shorter than that of *VF* under $\theta = 0.6$–0.9. Moreover, the values of *SDI_U* are always less than those of *VF_L* under the 95% confidence level. Furthermore, all the corresponding values of F shown in Table 10 are extremely greater than the value of $F_{0.05}(= 3.84)$. Therefore, we can conclude that *SDI* has a statistically significant shorter tuning time than *VF* under $\theta = 0.6$–0.9.

4.4 Simulation Results: SDI *vs.* DI

After generating n data items with the access probability $Pr(i)$, $1 \leq i \leq n$, based on the *Zipf* distribution, we randomly pick a permutation of these data items, and build an index tree of at most degree d for them. The access probability of each index node in this index tree is equal to the sum of the access probabilities of its child nodes. For our proposed algorithm, the replicated index nodes are determined according to the access probabilities of their child nodes by traversing the index tree. For the distributed indexing, the replicated index nodes are determined by the replicated level in the index tree. Therefore, parameter r is used to determine the replicated level in an index tree in the distributed indexing. The optimum value of r for achieving the best access time is defined as $r = \lfloor \frac{1}{2} \times log_d(n \times (d -$

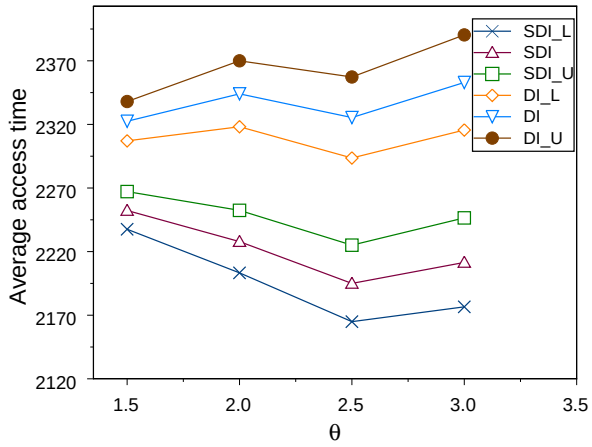


Figure 12: A comparison of the average access time of SDI and DI with increasing the value of θ.

$1) + \frac{d^{h+1}}{d-1}) - 1)\rfloor + 1$ [12].

According to [13], the tuning time of such tree-based indexes depends on the level of an index tree, and is bounded by $\lceil log_d n \rceil + 2 + R_{DI}$. Therefore, there is a limited improvement on the tuning time among the tree-based indexes, and in our simulation results, we will not show the comparison of the tuning time.

For the first simulation experiment, we increase the value of θ, the *Zipf* factor, from 1.5 to 3.0 under $d = 6$, $n = 1000$ and $R_{DI} = 5$. Figure 12 shows the experimental results of the average access time. In Figure 12, the x-axis represents the value of the *Zipf* factor θ, and the y-axis represents the average access time. As the value of θ increases, the access patterns become very skewed. We can observe that the average access time of our proposed algorithm is shorter than that of *DI*. This is because our proposed algorithm replicates the index nodes with a higher access probability, instead of always replicating the index nodes with the fixed level. In this way, our proposed algorithm increases the probability of accessing data items with a higher access probability in one broadcast cycle, reducing the average access time. Moreover, the values of *SDI_U* are always less than those of *DI_L* under the 95% confidence level. Furthermore, all the corresponding values of F shown in Table 11 are greater than the value of $F_{0.05}(= 3.84)$. Therefore, we can conclude that *SDI* has a statistically significant shorter access time than *DI*.

Table 11: The values of F of the average access time for the cases of increasing the value of θ

θ	1.5	2.0	2.5	3.0
F	41.252	40.664	34.059	29.375

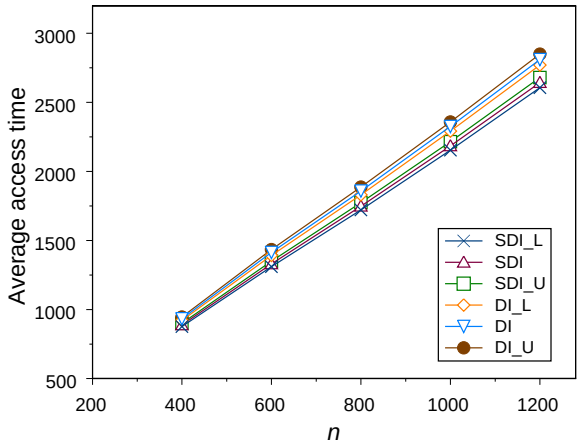


Figure 13: A comparison of the average access time of SDI and DI with increasing the value of n.

For the second simulation experiment, we vary the number of data items, n, from 400 to 1200 under $d = 6$, $R_{DI} = 5$, and $\theta = 3$. Figure 13 shows the experimental results of the average access time. In Figure 13, the x-axis represents the number of data items, n, and the y-axis represents the average access time. Table 12 lists its corresponding details of the figures. As the value of n increases, the average time of both algorithms increases. We can observe that the average access time of our proposed algorithm is shorter than that of *DI*. The reason is the same as mentioned before. Moreover, the values of *SDI_U* are always less than those of *DI_L* under the 95% confidence level. Furthermore, all the corresponding values of F shown in Table 13 are greater than the value of $F_{0.05}(= 3.84)$. Therefore, we can conclude that *SDI* has a statistically significant shorter access time than *DI*.

Table 12: The average access time for the cases of increasing the value of n

n	*SDI_L*	*SDI*	*SDI_U*	*DI_L*	*DI*	*DI_U*
400	876.974	888.549	900.124	921.329	933.52	945.712
600	1309.834	1329.838	1349.841	1391.753	1413.155	1434.558
800	1720.864	1747.377	1773.889	1830.106	1858.266	1886.425
1000	2153.898	2184.653	2215.408	2291.697	2324.698	2357.699
1200	2605.614	2642.7	2679.785	2769.944	2809.572	2849.201

Table 13: The values of F of the average access time for the cases of increasing the value of n

n	400	600	800	1000	1200
F	27.491	31.073	31.579	37.026	36.315

5 Conclusion

In this paper, we proposed the skewed distributed indexing for data broadcast with skewed access patterns over the single channel on wireless environments. Our proposed algorithm takes the access probability of each data item into consideration. Our proposed algorithm replicates the index nodes with a higher access probability, instead of always replicating the index nodes with the fixed level as in the distributed indexing. From our simulation results, we have shown that the proposed algorithm needs the shorter average access time than *VF* and the distributed indexing in many cases. 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, and S. Zdonik, "Disseminating Updates on Broadcast Disks," *Proc. of the 22rd VLDB Conf.*, pp. 354–365, 1996.

[2] S. Acharya, M. Franklin, and S. Zdonik, "Prefetching from a Broadcast Disk," *Proc. of the 12th IEEE Int. Conf. on Data Eng.*, pp. 276–285, 1996.

[3] D. Barbará, "Mobile Computing and Database—A Survey," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 11, No. 1, pp. 108–117, Jan./Feb. 1999.

[4] Y. I. Chang and S. Y. Chiu, "A Hybrid Approach to Query Sets Broadcasting Scheduling for Multiple Channels in Mobile Information," *Journal of Information Science and Eng.*, Vol. 18, No. 5, pp. 641–666, Sept. 2002.

[5] Y. I. Chang and C. N. Yang, "A Complementary Approach to Data Broadcasting in Mobile Information Systems," *Data and Knowledge Eng.*, Vol. 40, No. 2, pp. 181–194, Feb. 2002.

[6] Y. C. Chehadeh, A. R. Hurson, and L. L. Miller, "Energy-Efficient Indexing on a Broadcast Channel in a Mobile Database Access System," *Proc. of IEEE Int. Conf. on Information Technology: Coding and Computing*, pp. 368–374, 2000.

[7] 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.

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

[9] A. Datta, J. K. A. Celik, and D. E. VanderMeer, "Adaptive Broadcast Protocols to Support Power Conservant Retrieval by Mobile Users," *Proc. of the 13th IEEE Int. Conf. on Data Eng.*, pp. 124–133, 1997.

[10] A. Datta, D. E. VanderMeer, A. Celik, and V. Kumar, "Broadcast Protocols to Support Efficient Retrieval from Databases by Mobile Users," *ACM Trans. on Database Systems*, Vol. 24, No. 1, pp. 1–79, March 1999.

[11] 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.

[12] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Energy Efficient Indexing on Air," *Proc. of ACM-SIGMOD Int. Conf. on Data Management*, pp. 25–36, 1994.

[13] 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.

[14] 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.

[15] 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.

[16] K. Y. Lam, E. Chan, and J. C. H. Yuen, "Approaches for Broadcasting Temporal Data in Mobile Computing Systems," *Journal of Systems and Software*, Vol. 51, No. 3, pp. 175–189, May 2000.

[17] D. L. Lee and W. C. Lee, "On Signature Caching of Wireless Broadcast and Filtering Services," *Proc. of the 2nd Int. Mobile Computing Workshop*, pp. 15–24, 1996.

[18] W. C. Lee and D. L. Lee, "Information Filtering in Wireless and Mobile Environments," *Proc. of the 15th IEEE Int. Conf. on Computers and Comm.*, pp. 508–514, 1996.

[19] W. C. Lee and D. L. Lee, "Using Signature and Caching Techniques for Information Filtering in Wireless and Mobile Environments," *Distributed and Parallel Databases*, Vol. 4, No. 3, pp. 205–227, July 1996.

[20] 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.

[21] H. V. Leong and A. Si, "Data Broadcasting Strategies over Multiple Unreliable Wireless Channels," *Proc. of the 4th Int. Conf. on Information and Knowledge Management*, pp. 96–104, 1995.

[22] 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.

[23] S. C. Lo and A. L. P. Chen, "Optimal Index and Data Allocation in Multiple Broadcast Channels," *Proc. of the 16th IEEE Int. Conf. on Data Eng.*, pp. 293–302, 2000.

[24] 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.

[25] 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.

[26] K. L. Tan and J. X. Yu, "Energy Efficient Filtering of Nonuniform Broadcast," *Proc. of the 16th IEEE Int. Conf. on Distributed Computing Systems*, pp. 520–527, 1996.

[27] K. L. Tan and J. X. Yu, "Generating Broadcast Programs that Support Range Queries," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 10, No. 4, pp. 668–672, July/Aug. 1998.

[28] 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.

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

[30] 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.

[31] 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.

[32] 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.

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

[34] 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.