International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 5 Issue: 5 822 – 831

# Energy Efficient Varying Fanout Indexing Technique for Skewed Access Patterns in the Wireless Mobile Environments

Mani Dwivedi
MCA Department
AKG Engineering College
Ghaziabad, India
*dwivedimani@gmail.com*

Anuj Kumar Dwivedi
MCA Department
AKG Engineering College
Ghaziabad, India
*anujkdwivedi@gmail.com*

***Abstract*—** As we know, due to limited battery power, the most important issue in mobile computing is energy saving, which can be achieved through indexed data organization to broadcast data over wireless channels to a large no of mobile clients. In this paper, we explore the balanced and imbalanced index tree with varying fanout over skewed data. We purpose a varying fanout indexing technique with replication for data broadcast with skewed access pattern over a single wireless communication channel. We also show that replication can be performed at any level in varying fanout index tree, which increases the length of the overall broadcast cycle but reduces the directory miss. We compared our technique with the conventional as well as existing techniques. The performance results suggests the superiority of this technique over another replicated index technique i.e., fixed fanout index in all aspects. Our index technique also ensures correctness of results when larger size of broadcast file is used. From the performance analysis, the proposed indexing technique outperforms fixed fanout index technique.

***Keywords:*** *Varying Fanout, skewed data, replication, directory miss.*

*****

## I. INTRODUCTION

In recent years, the utilization of wireless technology devices has been growing at an exponential rate. For wireless data applications, the data dissemination methods are categorised between the two: point-to-point access and broadcast. In point-to-point access, a logical channel is established between the client and the server, where, queries are submitted to the uplink of server and returns the results to the client as in a wired network. In broadcast, data is transmitted simultaneously to all users who are residing in the broadcast area. It is to choice of a client to select the data it wants[1]. Data broadcasting is referred to as broadcasting which mainly transmits data, for example characters, shapes, still pictures, images, sounds etc., and differs from television broadcasting which mainly transmits videos or radio broadcasting which transmits sounds only [13]. But there is problem lies with data broadcast, when a mobile client retrieve a data item, it has to continuously monitor the broadcast channel until the data item of its interest arrives. This will consumes a lot of battery power. The limited battery capacity of mobile client's device makes power conservation a critical issue in the design of broadcast systems. It is important for mobile clients for energy saving will operate in two different modes**:** active mode and doze mode. The mobile clients can retrieve data from broadcast channels in the active mode only. However, the clients have much higher rates of battery consumption in the active mode than in the doze mode. The wireless devices can stay in the power saving mode or doze mode and tune into the broadcast channel only when the data items of interest to them arrive, hence lots of energy of these devices can be saved [1].The efficiency of the broadcast channels is estimated by two criteria used frequently :access latency and tuning time. The access latency refers to how fast. the client can access the requested data and tuning time refers to the duration for which the client stays active to receive the requested data items[13].

The performance of broadcast systems is always characterized by these two metrics. The tuning time can be reduced by means of air indexing. So by adding an index information to the broadcast file one can save mobile device power battery. Without indexing, the clients have to be continuously active and monitor the broadcast channel until the requested data item would arrive. This consumes significant amount of battery power and sacrifices energy efficiency. So the issue is to save battery power with minimized access latency during data broadcast in the single channel where data and index can broadcasted in the same channel.

## II. RELATED WORK

The broadcast disks in [7] takes into consideration for non-uniform data access distribution. In this approach, the several data items with similar access rates are grouped together to form logical disks. Each disk is assigned a relative broadcast frequency; more popular items are assigned higher frequencies. The broadcast schedule is then constructed by circularly picking up items from the disks based on their relative broadcast frequencies. Another indexing technique proposed in [9], a signature-based indexing method. Specifically, a broadcast cycle is divided into a number of frames. Each frame is preceded by a signature of its data item in the broadcast schedule. This allows the client to check whether a requested item is in the frame by investigating the signature only. However, this signature does not provide the arrival times of data items. Thus, when a match is found in a signature, the data items which are indexed by the signature have to be searched sequentially. Moreover, since a signature does not contain global information about the broadcast, data accesses require sequential scans of signatures. In[1] authors applied the tree-based index designed for traditional disk storage to wireless data broadcast. The index nodes in the tree are interleaved with data items in the broadcast schedule. Starting from the root index node, the client follows the links in the tree and tunes to selected index nodes to locate the requested item. The tree-based indexes are extended in [10,11] by constructing multiple index trees that share links. The resultant index structure allows searching to start at anywhere in the broadcast. Unfortunately, most tree-based

indexes are applicable to flat broadcast only because they require data items be ordered by their key values in the broadcast schedule. Besides from tree-based indexes, hash functions can also be used for indexing purpose to map data items to the slots in the broadcast schedule [8]. A salient feature of hash-based index is that it eliminates the need to broadcast index structures , since only a hash function is broadcast together with data. While the broadcast overhead of a tree based index structure normally increases with the number of data items, the broadcast overhead of a hash function is largely independent of the latter. Another energy efficient indexing scheme called MHash that optimizes tuning time and access latency in an integrated fashion [14].

It is noted that, in most databases, the access frequencies of different data items are usually different from one another [2]. Among the selective tuning strategies, in [3] constant fanout (CF) and variant fanout (VF) index tree takes the access probabilities of data items into consideration. More popular data item may be frequently accessed by the mobile clients than the less popular ones. This is known as skewed data access. However, VF assumes the sorted data items according to the access probabilities and index tree is constructed according to this sorted order. But in real life applications, the index tree should be constructed according to the key values of the data items, not according to access probabilities. The Alphabetic Huffman tree [4],[5] preserve leaf ordering on any input sequence used to construct them( similar to B+ Trees) ,i.e., left-to-right scan of the leaves of each tree will show the leaves ordering by their keys, and function as search trees. In order to minimize the tuning time, we consider two cases: one for fixed index fanout, and one for variant index fanout considering k-ary Alphabetic Huffman tree construction. For the case of fixed index fanout, in light of Alphabetic Huffman tree construction, we consider binary fanout. And for the case of variant index fanout we construct the k-ary Alphabetic Huffman tree which will produce varying fanout (between 2 and k). In VF, the replication of index nodes would not be considered. That means mobile 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 this paper we will consider the replication of index nodes at fixed level of tree in both fixed index fanout tree and variant fanout tree.

## III. BACKGROUND KNOWLWDGE

In the wireless communication environment, a broadcast cycle consists of a collection of data items which are broadcasted cyclically on the wireless channel. The mobile client in the broadcast area listen to the channel to retrieve the data item of their interest. This is known as selective tuning [1]. If the data is broadcasted without any index ,the mobile client will have to listen to the wireless channel, on the average, half of the total broadcasting time for the complete file. Hence by using proper indexing this selective tuning allows mobile clients to stay active only when the data of interest is present , thereby saving lot of battery consumption. In this section we will talk about the balanced and imbalanced index tree techniques.

### *A. Balanced Index Trees*

Most of the prior work is on symmetric balanced index tree with all leaves are in the same level and essentially the same fanouts for all index nodes. B+ tree indexing is a widely used indexing technique in traditional disk-based environments. It is also one of the first indexing techniques applied to wireless environments. The use of B+ tree indexing in wireless environments is very similar to that of traditional disk based environments [15]. Indices are organized in B+ tree structure to accelerate the search processes. In [1] two indexing schemes based on B+ tree data structure, (1,m) indexing and distributed indexing, are presented and assumed as balanced index tree. In distributed indexing, every broadcast data item is indexed on its primary key attribute. Indices are organized in a B+ tree structure.

### *B. Imbalanced index Trees*

In reality index trees may be imbalanced as the distance of leaf nodes from the root is not same or leaf nodes are not at the same level of the tree. Most examples of these trees are Huffman tree. It has been shown by experimental results that the use of imbalanced index trees will give considerable improvement in performance over the use of balanced trees, and such an advantage becomes even more prominent as the skewness of the data access increases [3]. The $B^+$ Tree based distributed indexing is a technique in which index is partially replicated introduced in [1] provides a method to multiplex it together with the corresponding data file on the broadcast channel. Two different states of Btree i.e. Fixed fanout tree and Varying fanout tree is well described in our previous work [16]. The Huffman tree construction considers the minimum frequency sum so that the index pointers of more popular data items are higher up in the tree than others. However, the Huffman tree constructed [4] not a search tree since users will need to know the encoding of a file before they can traverse the tree for the given file. We can have the index pointers are at different levels of the tree based on the popularity information, there is no way of traversing the tree to find a desired pointer by knowing only its key. Hence the only way to access a specific leaf is to know its Huffman code.

However this is a problem for mobile client since they only have the key of the file they are searching for. The mobile client cannot know the Huffman code of the desired file in advance since the code depends on the popularity patterns of other files being broadcasted at that time and may change over time. There exists a special class of Huffman tree known as Alphabetic Huffman tree [4] [5] which function as search tree and preserves the leaf ordering.

#### *B.1 Fixed Fanout Tree*

Firstly, we construct the fixed fanout Alphabetic Huffman index tree considering binary fanout [4,5] . For k=2 the fanout will be constant. Table1 is an example data set with 20 data items considered in [6], which will be continuously used for fixed and varying fanouts in the paper. The Alphabetic

**TABLE 1:Files and their popularity patterns**

| Key | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frequency | 23 | 4 | 12 | 10 | 17 | 31 | 15 | 21 | 29 | 19 | 7 | 12 | 16 | 14 | 20 | 48 | 11 | 22 | 18 | 8 |

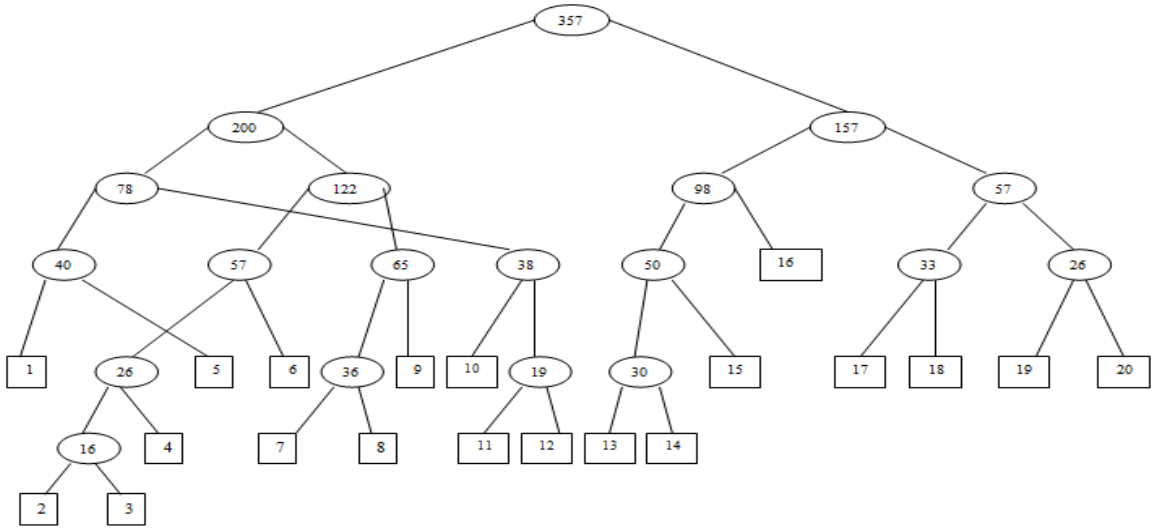


**Figure 1(a) The first step of constructing of Fixed Fanout Tree T**

Huffman Tree is constructed in two steps given by Hu & Tucker in [5] shown in Figure 1(a) and Figure 2(b).

**Step1:** Start with the initial sequence of data items $d_1$, $d_2$, $d_3$,.....,$d_n$, having leaf nodes in their given order. Combine the data nodes $d_i$ , $d_j$ such that sum of their frequencies is the minimum and there are no leaves between them and also $d_i$ and $d_j$ are the leftmost nodes among all candidates. A new sequence of data items $d_1, d_2,...,d_{i-1}, d_{(i, j)}$ ,$d_{i+2}$ ,...., $d_n$ where $d_{(i, j)}$ is an index node and other are still leaf nodes ; now combine some adjacent with minimum frequency in this new sequence and replace the combined pair with sum and so on. This will produce a tree T without alphabetic ordering of the data nodes as in Figure 1 where we record the frequencies of each index node inside the circle as index key values.

**Step 2:** Now record the level of each data node of T denoted as $L_i$ . Consider the root node level is 1. From bottom to the root, rearrange the pointers such that for each level the leftmost two nodes have the same parent, and then the next two and so on [6]. Therefore Alphabetic Huffman Tree T' is generated without changing the level of each node in T as shown in Figure 2(b).

We also extended this algorithm in the next section to construct *k*-ary(Variant Fanout) Huffman-Tree, by merging at most *k* nodes in step1, and combining up to *k* nodes with the same parent in step2. After generating the alphabetic Huffman tree *T'* in Fig 3, we cut *T'* at level *l*, and perform a depth first traversal. The index node above *l* is still called control index, and index nodes below *l* is search index.

The broadcast sequence of this fixed fanout tree is given in Figure 1(c), in which the control and the search indices are shaded as grey. In $B1^{[rep]}$ , [rep] indicates the number of replication of index node B1. Since in this broadcast sequence every control index node is replicated for fixed number of times (say for two times) in this example. The index node A is broadcasted first, then B1 and B2 is traversed. Since the root node A and its child nodes B1 and B2 are replicated, so they will appear twice in the broadcast sequence, or we can say for fixed number of times.

*B.2 Varying Fanout Tree*

In the modification of above Hu & Tucker algorithm , we allow at most k nodes to be combined into a single super-node during the passes in step1 , instead of two nodes in [5]. We also allow combining k leaf nodes if they are consecutive in the construction sequence, while the other conditions remain the same. Also, the second step remains the same except that we allow upto k nodes to be join together to have the same parent (i.e. from 2 to k nodesSince the conditions on combining nodes in the first step are modified minimally, we can still perform the reordering phase similar to that in [5]. In our example, the varying fanout number, k=3. So we can join 2 or 3 nodes to have the same parent in the index tree .The k-ary construction for the Table 1 is given in Figure 2.

## IV. REPLICATION IN VARYING FANOUT (RVF) INDEX TREE

In this paper, the replication of index nodes of index trees would have been considered for varying fanout. Note that the number of times a index node is replicated depends on the number of fanouts it has. In this section we will consider the same algorithm for the construction of k-ary Huffman Tree in previous section in which we will consider three data items $d_i$ , $d_j$ ,$d_k$ , such that they are consecutive and their frequency sum is minimum [5, 6]. The Alphabetic Huffman tree constructed.

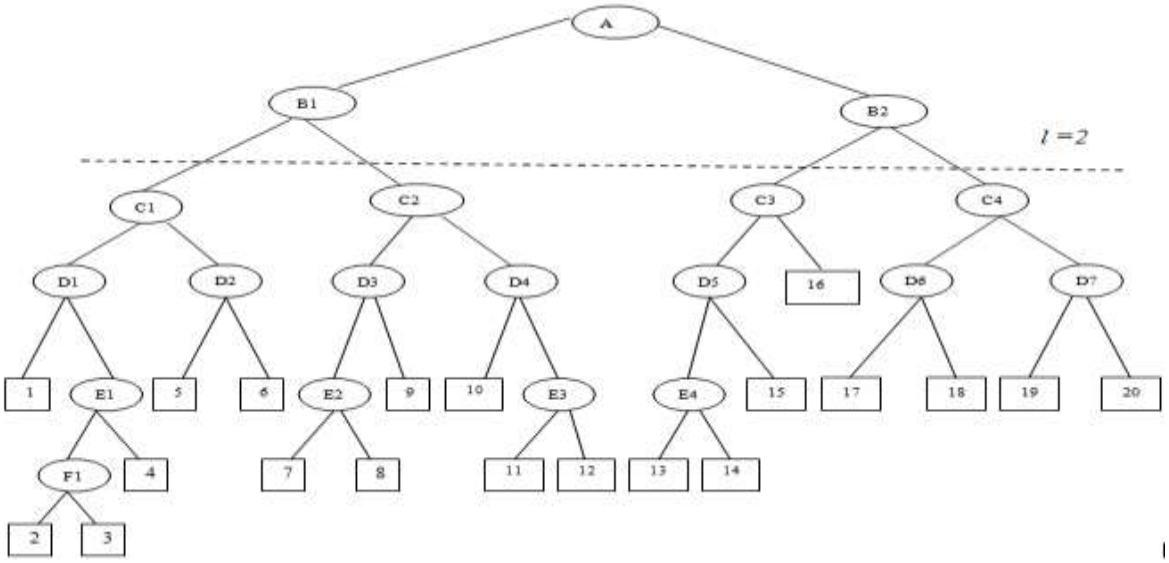


**Figure 1(b) The Final Huffman Tree T'**

| A[1] | B1[1] | C1 | D1 | 1 | E1 | F1 | 2 | 3 | 4 | D2 | 5 | 6 | B1[2] | C2 | D3 | E2 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| D4 | 10 | E3 | 11 | 12 | A[2] | B2[1] | C3 | D5 | E4 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| B2[2] | C4 | D6 | 17 | 18 | C4 | D7 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|

**Figure 1(c) Broadcast Sequence of Huffman Tree T'**

in this paper will be imbalanced index tree. The k-ary Alphabetic Huffman tree in Figure 3, which have varying fanout will be replicated.

We cut the tree T'' at level *l* so that the index nodes above *l* is still called control index, and index nodes below *l* is search index. Now we will perform the depth first traversal of the replicated tree given in Fig 3(a) and hence the final broadcast sequence B generated in this example is given in Fig 3(b). The index node A is broadcasted first. Next the subtree rooted by B1 is traversed , then B2 is traversed in preorder. Since root node A and its child nodes B1 and B2 are in the replicated part, these nodes are broadcasted again. As root node A is having its two child nodes B1 and B2 will be replicated two times. But the index node B1 is having child nodes 1, C1 and 9, then it will replicated three times and B2 will also replicate three as it has its child nodes as C2, C3, C4.

The important feature of this k-ary index trees is that we may end up with a tree with smaller depth resulting in smaller broadcast sequence. It is important to note that in the k-ary construction, it may not be always possible (or optimal) to combine k nodes together. Therefore this k-ary Alphabetic Huffman tree will have a fanout that varies between 2 and k.This indexing technique would solve the problem of directory miss which would be occurring in previous VF because replication is not considered. Moreover this technique would result in reduced average access time.

After seeing the broadcast sequence of both fixed and varying fanout index tree in Figure 1(c) and Figure 3(b), we found that root node A is replicated two times both in fixed fanout and varying fanout. But the index node B1 of Figure 3(c) is replicated two and that of Figure 3(b)is three times. Likewise happen with index node B2 would replicate two times in fixed fanout index tree but three times in varying fanout index tree. Hence it is possible in varying fanout index tree the index nodes replication would vary at different levels of the tree. Now, by taking into account both the broadcast sequence, we found that the length of fixed fanout tree is larger than the varying fanout index tree. The summary of these findings is given in Table 2.

### *A. Control Index*

The bcast of every index tree will contain the control index, search index and data index. Each search index

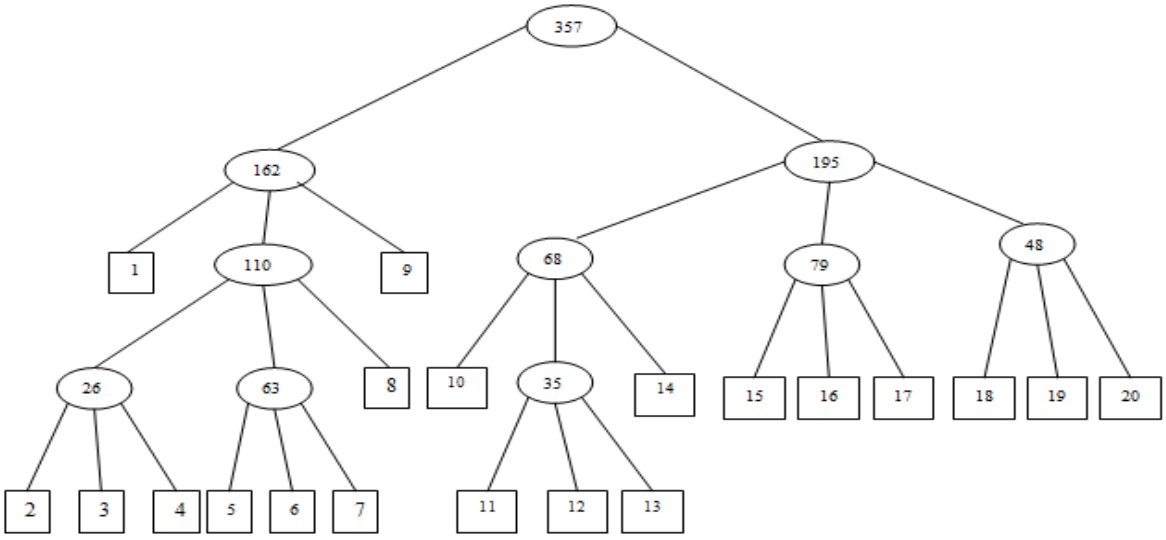


**Figure 2 The k-ary Alphabetic Huffman Tree T''**

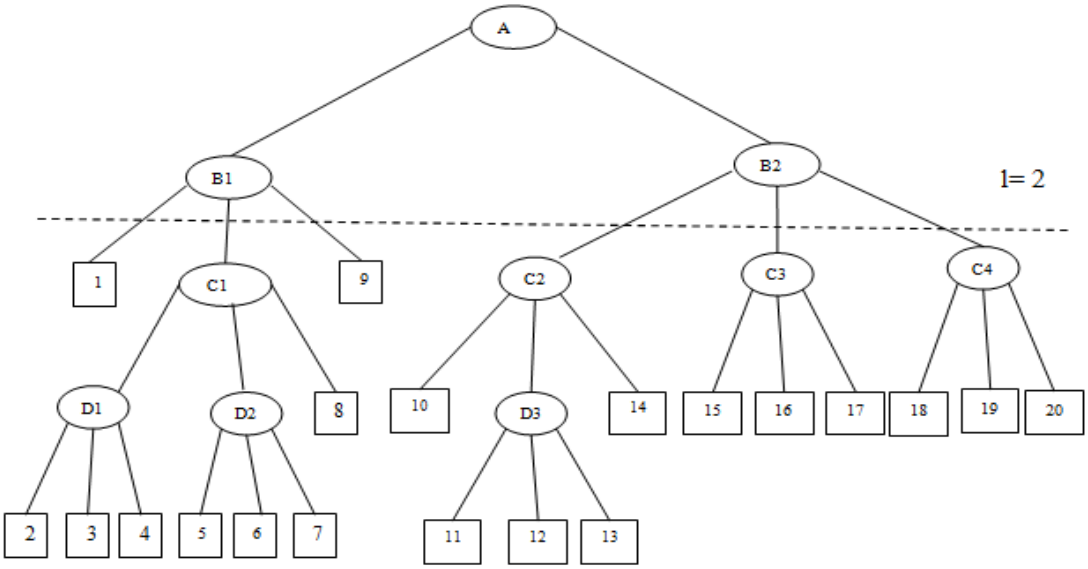


**Figure 3(a) The k-ary Alphabetic Huffman Index Tree T'''with replication at l=2.**

node and data index node have an address to its nearest replicated index node of following broadcast. Each control replicated index node has control in Figure 3(b), which can guide to the proper index node. The control indexes of replicated index nodes of Figure 3(b) are shown in Figure 4. The first entry in the control index $B2^{[1]}$ is {9, Begin}, given in Figure 4,which means that if the client searching for a data item with the key K≤9, then it has to wait till the beginning of the next broadcast cycle. The second entry {14, $B2^{[2]}$}implies that if a client is searching for a data item with key K> 14, then wait till the arrival of $B2^{[2]}$. So, this control index indicates that the subtree immediately after it, will direct to data node with the key values K in the range {9, 14}.

*B. Access Protocol*

Now, we describe the client access protocol for the proposed technique. Assume that the data item with the key K is requested. The access protocol is given as:

1. Tune into broadcast channel with the key K randomly.
2. let B= current bucket.
3. read B to get the offset of the nearest replicated index node.
4. go into doze mode.

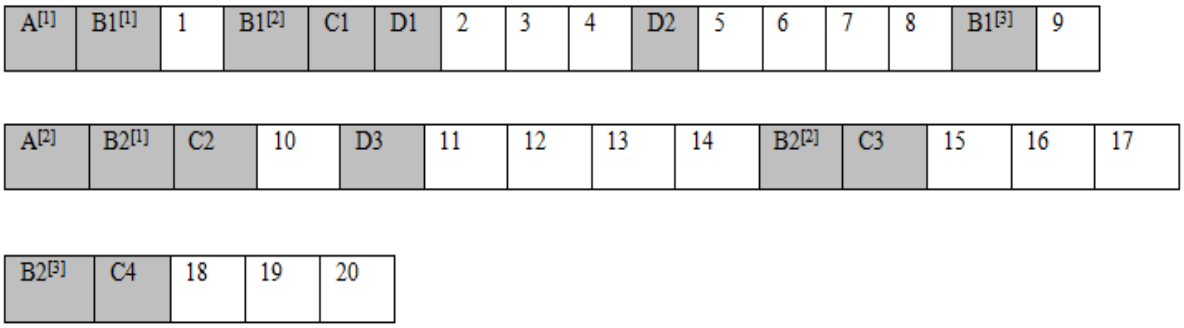


**Figure 3 (b) Broadcast sequence of k-ary Huffman tree T'''**

**TABLE 2 :Summary of Fixed and Varying Fanout (with replication) with Non Replicated Varying Fanout**

| Issues | Replicated Fixed Fanout | Replicated Varying Fanout (RVF) | Non replicated Varying Fanout |
|---|---|---|---|
| Replicated index nodes | Fixed | Vary | Zero |
| Depth of index tree | Deep | Less deep | Same as RVF |
| Length of broadcast sequence | Larger | Smaller | Smallest |
| Directory miss | Reduced | Reduced | Increased |

5. tune in again at nearest replicated index node.
6. assume B=control index bucket.
7. Read the index bucket B
8. if K ≤ B.Key , then
9. go into doze mode until the beginning of next broadcast cycle.
10. else if K=B.Key, then
11. read the offset to the actual data item go into doze mode and became active when the requested data bucket arrives
12. download the data item.
13. Else
14. go to higher level index node that contains the control index
15. repeat from step 7

## V. PERFORMANCE ANALYSIS

In this section, we study the performance of the proposed technique. We compare our proposed algorithm with fixed fanout index tree technique (Zhong et. al 2011). The parameters used in performance evaluation are given in Table 3. Let Pr(i),1 ≤ i ≤ n, is the access probability based on the Zipf distribution. The Zipf distribution is typically used to model non-uniform access patterns and can be expressed as

$$P(i) = \frac{\left(\frac{1}{i}\right)^{\theta}}{\sum_{j=1}^{n}\left(\frac{1}{j}\right)^{\theta}}, 1 \le i \le n, \quad \ldots(1)$$

where Θ is a parameter named access skew coefficient. When Θ=0, we have the uniform distribution. When the value of Θ increases, the access probabilities become more skewed(Chen et al, 2003). The height of a k-ary is given as

$$h = \lceil \log_k(k-1) + \log_k N - 1 \rceil \quad \ldots(2)$$

where k is the number of fanout in a tree and N is the total number of nodes in an index tree, which is given as

$$N = \frac{nk-1}{k-1} \quad \ldots(3)$$

Since in a k-ary tree h denotes the height of the tree and the tree constructed using Hu-Tucker algorithm is an unbalanced tree. The value of h tells that the data nodes starts from this level when traversing from the root node in an index tree. So the number of levels in an index tree is given by L. We will cut the tree at *l* by assuming (h-1)=*l*, because data nodes are at h in an unbalanced tree. Now, the number of nodes above *l*, that is in the replicated part are known to be control indices which is given by,

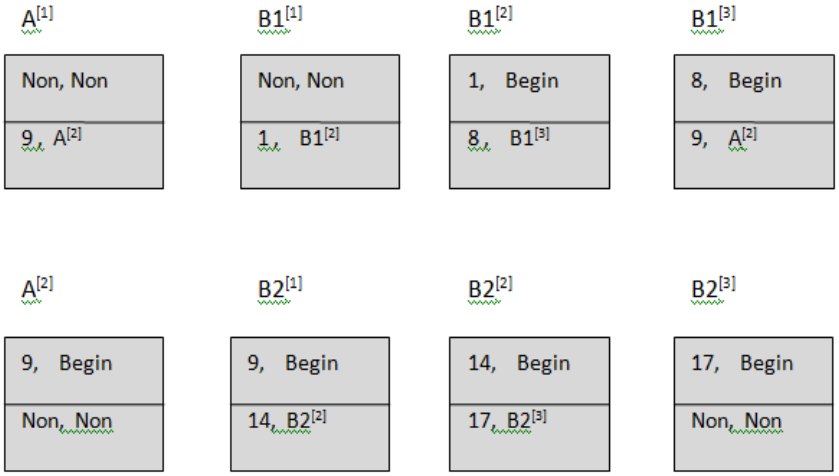


**Figure 4 The Control Indexes**

$$rInd = \left\lfloor \frac{1}{k-1} \sum_{d=2}^{k} \left( \sum_{i=1}^{l} d^{(l-i)} \right) \right\rfloor \quad \ldots(4)$$

The total number of index nodes I, is obtained by subtracting the number of data items from the total number of nodes. Also, the index nodes below *l* are known as C is obtained by subtracting the number of replicated nodes above above *l* from total number of internal nodes. An index bucket may have different size from a data bucket, so r is the ratio of data bucket size to index bucket size. So the length of bcast in data bucket units,

$$BC = \frac{total_{indices}}{r} + n + C + rInd \quad \ldots (5)$$

Note that, RP denote the average of replicated part and NRP denotes the average length of non replicated part. Hence, we have

$$RP = \frac{number\_of\_control\_indices}{S}, \quad \ldots (6)$$

**and**

$$NRP = \frac{number\ of\ search\ indices + number\ of\ data\ items}{S} \quad \ldots(7)$$

**TABLE 3: Parameters List**

| Parameters | Description |
|---|---|
| N | Total number of data items |
| T | An index tree |
| N | Total number of nodes in an index tree |
| K | Maximum number of fanouts in a tree T |
| L | Level of T |
| *L* | Level to cut T for replication |
| I | Total number of index nodes in a T |
| Rind | Total number of replicated index nodes in a T |
| S | Total number of subtrees at *l*+1 on T |
| RP | Length of replicated part of a T |
| C | Total number of nodes below *l* |
| NRP | Length of non replicated part of a T |
| BC | Length of Bcast on a channel |
| R | Ratio of data bucket size to that of index bucket size |
| Θ | Zipf factor |

*A. Average Access Time*

The access time is the sum of bcast wait and the prob wait. For the analysis of the access time, there are two cases: (1) The clients tune in at i[th] node before the corresponding nearest-replicated index node to the wanted data node, w is shown in Figure5(a); (2) the clients tune in after i[th] corresponding nearest-replicated index node as shown in Figure 5(b). In the first case, the clients can retrieve the data node of interest in the same cycle; but in the second case, the clients have to wait for the next cycle to retrieve that wanted data node. Then the average access time is

$$AAT = \frac{1}{BC}\left( \sum_{i=1}^{S} \left( \sum_{w=1}^{s-2} \frac{RP_i + NRP_i}{2} + RP_w + \frac{NRP_w}{2} \right) \cdot (RP_i + NRP_i) + \sum_{i=1}^{S} (RP_i + NRP_i) \cdot NRP_i \right) \cdot P(i) \quad \ldots(8)$$

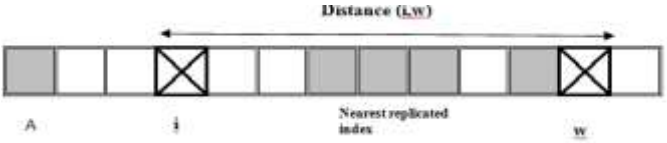


**Figure 5(a) Tune in Before Corresponding Nearest Replicated Index**

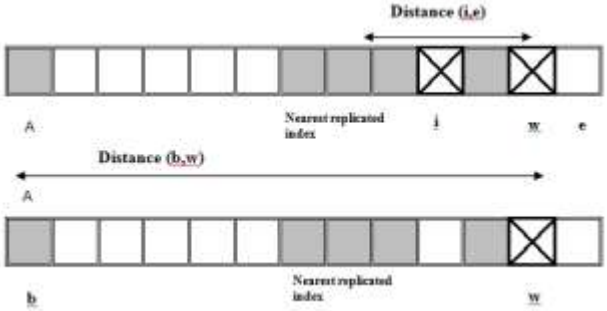


**Figure 5(b) Tune in After Corresponding Nearest Replicated Index**

*B. Average Tuning time*

The tuning time primarily depends on the number of levels L of the index tree which is equal to the number of probes by a client. There are two steps for determining the tuning time. Firstly the client tunes into the broadcast channel and search for the right index and then, the client searches for the index that directs the client to the required data and goes into doze mode and becomes active when data appears and download the data. In the first step, there may be any one of the following cases: (1) client tunes into a control index (in the replicated part), (2) the first visited bucket is a data bucket, and (3) the first visited bucket is a search index(in the non replicated part).

$$ATT = \frac{2\times RP+(2+r)\times n+3\times NRP}{r\times BC} + \sum_{i=1}^{n}\left(\frac{l}{2r}+\frac{1}{r}\left(\frac{L_i-l}{k}\right)+dsize\right)\times P(i)$$

where ***dsize*** is the size of each data item.

## VI. RESULT ANALYSIS

In this section, results are used to evaluate the performance of RVF over fixed fanout technique. The performance metrics (AAT and ATT) are implemented using JDK1.6 on Intel(R) Atom(TM) CPU N455 computer with 2GB memory, and Windows XP operating system.

The proposed technique is tested for the database of 5000 data items (Zhong et al, 2011), each of which has different sizes say from 1KB to 4KB, and multiple clients sending their request for different set of data items. The access probability of each data item must satisfies the zipf distribution and considers the skew factor $\Theta = 0.95$. The size of each data bucket is set to 1 KB then the size of each index bucket is 0.1KB. We also assume the same scenario (Zhong et al, 2011) for deriving the results that is set up r = 10. Another important factor is the number of fanout k which would vary in the result.

By considering k = 3, which would vary k from 2 to 3 for varying fanout and k = 2 for fixed fanout index tree, we vary the number of data items from 1000 to 5000 to compare bcast, average access latency and average tuning time of varying fanout technique and fixed fanout technique. Firstly, we evaluated the broadcast cycle length (bcast) of varying fanout index tree and fixed fanout index tree. We consider the bucket size ratio r for analyzing bcast length, AAL and ATT. Since both the trees are unbalanced, but their fanouts are different, so these two approaches have different bcast length after index and data allocation.The calculated bcast length of both the techniques with the increasing number of data items is given in Table 4.

**TABLE 4:The Length of Bcast for Increasing Value of n**

| No. of data items (n) | Varying Fanout | Fixed Fanout |
|---|---|---|
| 1000 | 2007.9 | 2661 |
| 2000 | 3557.9 | 5324.2 |
| 3000 | 5959.3 | 8550.6 |
| 4000 | 7509.3 | 10650.91 |
| 5000 | 9059.12 | 15003.40 |

In Figure 6, the bcast of fixed fanout technique is always longer than varying fanout technique. Since the number of fanouts are fixed in the fixed fanout technique, so it has much more index nodes than varying fanout technique, even when we use the same data set and cut the tree at same level. The average access time for increasing data items is shown in Table 5. Now from Figure 7, we can see that the varying fanout technique has much shorter average access latency than fixed fanout technique, while the average access latency of both the techniques gradually increases as the number of data items increases.

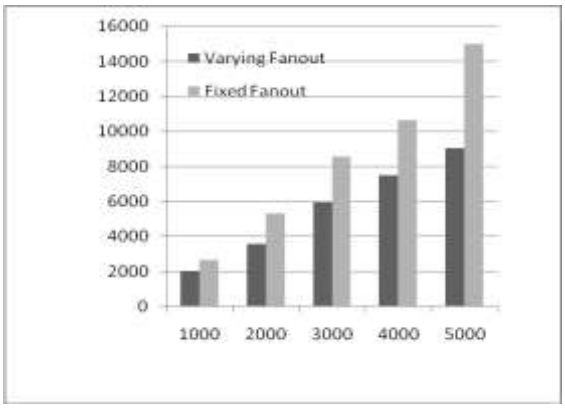


**Figure 6 Bcast Length with the Increasing Value of n**

Since the average tuning time is dependent on the number of levels in the index tree, we firstly calculate the total number of nodes in the tree and then find out the number of levels. And putting the value of r and BC, we get the average tuning time of varying fanout technique which is given in Table 6. In Figure 8 varying fanout technique needs less average tuning time than fixed fanout technique which means that varying fanout technique is more energy efficient. Also, as the number of data items increasing, the average access latency and average tuning time gap between varying fanout and fixed fanout also increases.

**TABLE 5 :The Average Access Time**

| No. of data items (n) | Varying Fanout | Fixed Fanout |
|---|---|---|
| 1000 | 1884.52 | 2842.96 |
| 2000 | 3338.44 | 5692.42 |
| 3000 | 5600.11 | 9484.12 |
| 4000 | 7055.22 | 11391.35 |
| 5000 | 8508.67 | 17067.08 |

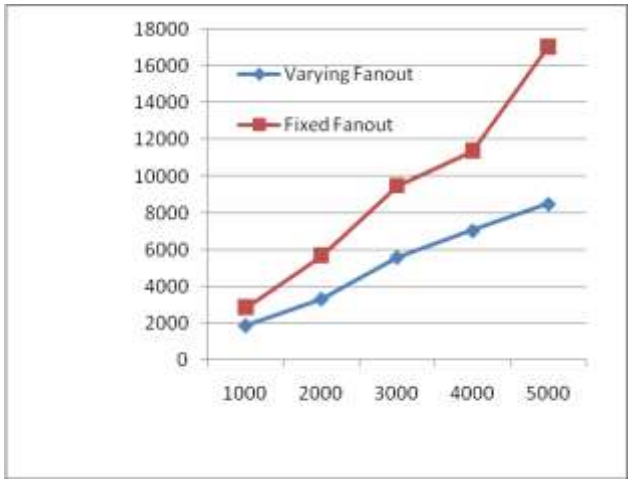


**Figure 7 AAT with the Increasing Number of Data Items.**

**TABLE 6: The Average Tuning Time**

| No. of data items (n) | Varying Fanout | Fixed Fanout |
|---|---|---|
| 1000 | 51.02 | 76.00 |
| 2000 | 101.02 | 150.95 |
| 3000 | 150.40 | 225.49 |
| 4000 | 200.49 | 300.51 |
| 5000 | 250.52 | 375.52 |

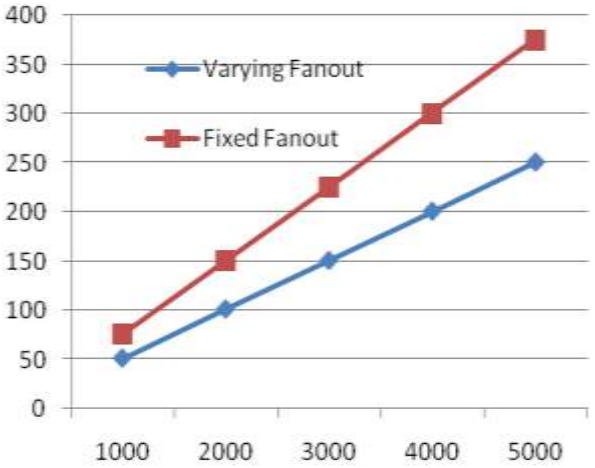


**Figure 8 ATT with the Increasing Number of Data Items**

## VII. CONCLUSION

In this paper, we proposed a varying fanout indexing technique with replication for data broadcast with skewed access pattern over a single wireless communication channel. Our proposed technique takes the frequency of each data item into consideration i.e. skewed pattern and generates the alphabetic Huffman tree. Then replication is performed in this tree at any fixed level . This varying fanout tree will generate a tree with smaller depth, which results a smaller broadcast sequence. We also found that the number of times a index node is replicated, would also vary. The length of overall broadcast cycle is increased but it reduces the directory miss because the root node is more times replicated than the non replication. Finally, the comparison of results between the existing technique i.e., fixed fanout and the replicated varying fanout tree have been performed. The major advantages of this replicated varying fanout (RVF) index tree technique are: (1)The problem of directory miss has been reduced significantly in this technique because the root node is more times replicated than in the non replication. (2) This proposed technique also improved the performance metrics: (a) average access time; (b) average tuning time. From our results, we have shown that the proposed technique (RVF) needs the shorter average access time and average tuning time than the fixed fanout indexing technique. So, the result reveals that replicated varying fanout (RVF) index tree is more energy efficient and also responses much faster than the existing techniques.

## VIII. REFERENCES

[1] .T. Imielinski, S. Viswanathan, and B. R. Badrinath. "Energy efficient indexing on air". In Proceedings of the

International Conference on SIGMOD, pages 25–36, 1994.

[2] A. Dan, D.M. Dias, and P.S. Yu, "The effect of Skewed Data Access on Buffer Hits and Data Contention in a Data Sharing Environment", Proc. 16th Large Dtabases Conf., pp.419-431, Aug.1990.

[3] Chen , M.S., Wu, K.L, " Optimizing index allocation for sequential data broadcast in wireless mobile computing", IEEE Transactions on Knowledge and Data Engineering, 15(1), pp. 161-173, 2003.

[4] Shivakumar, N., Venkatasbramanian, S.," Energy efficient indexing for Information Dissemination in wireless Systems", ACM Journal of Wireless and Nomadic Application ,1996.

[5] Hu, T.C., Tucker, A.C.,"Optimal computer search trees and variable-length alphabetic codes" ,SIAMJ. Applied Mathematics, 21(1), pp. 514-532, 1971.

[6] Jiaofei Zhong ,Weili Wu and Yan Shi, " Energy Efficient Tree Based Indexing Schemes for Information Retrieval in Wireless Data Broadcast", DASFAA 2011, Part II, LNCS 6588, pp.335-351, 2011.

[7] S. Acharya, R. Alonso, M.J. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments," Proc. ACM SIGMOD '95, pp. 199-210, May 1995.

[8] T. Imielinski, S. Viswanathan, and B.R. Badrinath, "Power Efficient Filtering of Data on Air," Proc. Fourth Int'l Conf. Extending Database Technology, pp. 245-258, Mar. 1994.

[9] W.-C. Lee and D.L. Lee, "Using Signature Techniques for Information Filtering in Wireless and Mobile Environments," Distributed and Parallel Databases, vol. 4, no. 3, pp. 205-227, July 1996.

[10] J. Xu, W.-C. Lee, and X. Tang, "Exponential Index: A Parameterized Distributed Indexing Scheme for Data on Air," Proc. ACM/ USENIX MobiSys, pp. 153-164, June 2004.

[11] 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. Knowledge and Data Eng., vol. 18, no. 3, pp. 92-404, Mar. 2006.

[12] J.Shen, Y.Chang," A skewed distributed indexing for skewed access patterns on the wireless broadcast", The Journal of Systems and Software 80(2007), Elsevier Inc., pp. 711-723, October,2006.

[13] J.Xu, DL Lee,Q Hu and WC Le, " Data Broadcast", Chapter11, Handbook of wireless networks and mobile computing, 2002, pp.243-265.

[14] Y. Yao, X. Tang, EP Lim and A. Sun," An energy efficient and access latency optimized indexing Scheme for Wireless Data broadcast", IEEE Trans. Knowledge and Data Eng.,vol. 18 no.8, pp. 1111-1124. August 2006.

[15] X.Xang and A. Bougettaya,"Broadcast-Based Data Access in Wireless Environments", EDBT 2002, LNCS 2287, pp. 553-571, 2002.

[16] Mani Pandey and Vikas Goel, "The Replication in Varying Fanout Indexing Technique for Skewed Access Patterns in the Wireless Mobile Environments" International Journal of Computer Applications, Vol. 51, No. 4, August 2012.