An Extended Exponential Index Scheme for Multi-disk Broadcast in a Single Wireless Channel

Kiyoung Park and Sungwon Jung
Department of Computer Science and Engineering
Sogang University
{joy710, jungsung}@sogang.ac.kr

Abstract

In the wireless broadcast scheme, single channel index schemes are fit to flat broadcast that performs well when all the broadcasted data items are accessed with the same probability whereas the multi-disk broadcast scheme is proper when the data access distribution is skewed. The existing index schemes, however, cannot point the replicating data items in a broadcast cycle, so they are not efficient for the Multi-disk broadcast scheme. This paper proposes a Multi-disk Exponential Index (MDEI) which is a single channel index scheme fit to Multi-disk broadcast scheme. Because MDEI scheme organizes a separate index for each disk, it functions with multi-disk broadcast, resulting in a greater reduction of average access latency than that of other flat-broadcast index schemes when the data access distribution is skewed. The performance evaluation showed that MDEI has a good performance when data access distribution is skewed.

1. Introduction

Broadcast scheme has attracted much attention in the mobile environment for efficient data delivery because the downlink capacity is much greater than the uplink capacity and power of the mobile unit is limited. In the proposed scheme, the server continuously broadcasts data items, and the clients selectively receive the items they need. The performance of broadcast can be measured by the tuning time that is an amount of time spent in listening to the channel and by the access latency that is the time elapsed from the moment a client issues a query to the moment the answer is received by the client. To reduce the time required to get a data, the client’s access latency needs to be reduced. To conserve the power, the clients’ tuning time needs to be reduced.

The index lets clients know when the data item would be broadcasted in order that the tuning time and power may be reduced. Many index schemes have been proposed previously such as the tree-based index scheme [4] like (1,m) index and distributed index, the hash-based index scheme[5], the signature-based index scheme[6], and the exponential index scheme[1]. All of these schemes are fit to flat broadcast which has good performance when all the broadcasted data items are accessed with the same probability. Multi-disk broadcast scheme [2] is appropriate when the data access distribution is skewed. The existing index schemes, however, cannot point multiple data items in a broadcast cycle so they are not efficient in the Multi-disk broadcast scheme.

MHash scheme [3] has a better performance than the index scheme appropriate for flat broadcast when the data access distribution is skewed, because it uses a two-parameter hash function which is proper for non-flat broadcast. However, MHash scheme is not suitable for the multi-disk broadcast program for two reasons. First, there is no hash function that can point the exact bucket of multi-disk broadcast program for each of the data items broadcasted. Secondly, MHash limits the maximum replication number for broadcast data. That is, when the data access distribution is largely skewed, the multi-disk broadcast program may require a replication number larger than the maximum replication number of MHash scheme.

There is no proper index scheme for multi-disk broadcast over a single channel. Although there are some index schemes for multiple channels [7,8], they are not applicable to multi-disk broadcast over a single channel. In this paper, we propose a single channel index scheme Multi-disk Exponential Index (MDEI) which is applicable to multi-disk broadcast. MDEI index can reduce the average access latency in the multi-disk broadcast when data access distribution is skewed.
MDEI extends its exponential index[1] to point multiple replicating data items in the same broadcast cycle to function in the multi-disk broadcast. The original index scheme does not function in the multi-disk broadcast since it cannot point multiple replicating data items. MDEI scheme organizes a separate index for each disk so that it can function in multi-disk broadcast.

Among the many other index schemes, the exponential index was chosen to extend for two reasons. First, the effect of changing the number of data items between disks is smaller than the other index schemes, since its index table size is increased logarithmically. Second, the index table is broadcasted in a short period. As a result the initial probe time gets shorter and the average access latency can be reduced.

The rest of the paper is organized as follows: Section 2 describes the structure of MDEI. Section 3 presents MDEI data access protocol. Section 4 gives the performance analysis of MDEI scheme. Finally, the conclusion is given in Section 5.

2. MDEI structure

In this section, we show how to construct MDEI (Multi-disk Exponential Index). Figure 1 shows an example of broadcast data items. There are total 40 data items. Eight of them are more popular than the others. According to [2], we can construct the multi-disk broadcast program of Figure 2 if the popular data items are broadcasted two times as frequently as the others. The popular data items like D33, D35 are included in disk 1 and broadcasted twice in a broadcast cycle. The others are included in disk 2 and broadcasted once in a broadcast cycle. Note that data items in each disk are sorted in an increasing order of data item key.

Figure 1. A set of broadcast data items

All the buckets in the broadcast program have one or more data items, the control information, and an index table. For simplicity, however, each bucket is assumed to accommodate the control information, an index table, and only one data item. The structure of the index table is the same as that of the exponential index. The index table consists of tuples of \( \{\text{distInt, maxKey}\} \), where \( \text{distInt} \) means a distance from the current bucket (measured in the unit of buckets), and \( \text{maxKey} \) means the maximum key value of the bucket which is away as far as \( \text{distInt} \)-buckets. The \( \text{distInt} \) increases exponentially: the 1st entry of \( \text{distInt} \) is 1, the 2nd entry is 3, the 3rd entry is 7, and the \( i \)th entry is \( 2^{i-1} \). The contents of the index table, however, are different from those of the original exponential index. MDEI organizes a separate index for each disk to function with multi-disk broadcast. Hence, the index table broadcasted in disk 1 is organized with the data items broadcasted in disk 1. The same index table applies to separate index for disk 2, separate index for disk \( k \). For example, the index table of bucket 1 presents D35 which is 1 bucket away from the start, D43 is 3 buckets away, and D68 is 7 buckets away in disk 1.

It is impossible to identify the location of the required data item only with the index table, because the broadcast program is interleaved with other disks and the index table does not know how the other disks are interleaved. Therefore, each bucket has additional information called control information and the contents presented in Table 1. Usage of this information will be explained in Section 3 with data access protocol for clients.

<table>
<thead>
<tr>
<th>Table 1. Control information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notation</td>
</tr>
<tr>
<td>numDsk</td>
</tr>
<tr>
<td>dskNo</td>
</tr>
<tr>
<td>mcLen</td>
</tr>
<tr>
<td>dskLen</td>
</tr>
<tr>
<td>bktNo</td>
</tr>
</tbody>
</table>

3. MDEI data access protocol

This section presents the access protocol of MDEI scheme. Section 3.1 informally describes MDEI data access protocol by using an example. Section 3.2 presents the tuning position calculating method while accessing a data.

3.1 Informal description of MDEI data access protocol

Suppose that the client issues a query for data item D33 just before the bucket 1 is broadcasted. The client reads the index table in bucket 1. Since D44 falls between the second maxKey D43 and the third maxKey D68, D44 must be in the buckets four through seven away from the current position if D44 is broadcasted in disk 1. Therefore, the client needs to tune at the fourth bucket of disk 1. It is bucket 13 which is calculated with control information. The calculation method will be explained in Section 3.2. Therefore, the client stores...
13 as the next tuning position in disk 1 and turns into doze mode until disk 2 is broadcasted.

Disk 2 starts at bucket 5. It also can be calculated with control information and the method will be explained in Section 3.2. The client examines the data item D44 in bucket 5. As the required data cannot be found in bucket 5, the client continues to check the index table in bucket 5. Since D44 falls between the third maxKey D41 and the fourth maxKey D51, D44 must be in the buckets that are eight through fifteen away if D44 is broadcasted in disk 2. Therefore, the client needs to tune at bucket 17 which is the eighth bucket, and stores it.

Next tuning positions of disk 1 and disk 2 are bucket 13 and bucket 17, respectively. The client tunes into the channel at bucket 13. Since bucket 13 does not have the required data item, the client finds out there is no bucket that contains required data item. Hence, the target item is not in disk 1. So the client stays in doze mode until bucket 17 which is the next tuning position of disk 2. The target item is not found in the bucket, but the key is found in the first entry of the index table. The client receives the data item D44 in the next bucket.

To get the target data item in this way, the client should be able to calculate the next tuning position with the index table and control information. Section 3.2 explains how to calculate the tuning position.

### 3.2 Tuning position calculation method

To calculate the next tuning position, the offset to the next tuning position should be obtained from the index table and then the offset needs to be converted into real distance. These two offsets are defined as follows:

**Definition 1.** *Logical_offset* is the offset to the next tuning position obtained from the index table directly. The index table does not consider the other interleaved disks and neither does the *logical_offset*.

**Definition 2.** *Physical_offset* is the real offset or distance to the next tuning position. It can be calculated from the *logical_offset* and control information.

As the *logical_offset* is the offset that does not consider the other interleaved disks, it should be converted into a *physical_offset*. The *physical_offset* can be calculated from the *logical_offset* as follows:

\[
\text{physical\_offset} = q \times \text{mcLen} - \text{bktNo} + r + 1
\]

where

\[
q = \left\lfloor \frac{\text{logical\_offset}}{\text{dskLen}} - \text{bktNo} \right\rfloor
\]

\[
r = \begin{cases} 
\frac{\text{logical\_offset}}{\text{dskLen}} & \text{if } q=0 \\
(\text{logical\_offset} - \text{bktNo} - 1) \mod \text{dskLen} & \text{if } q>0
\end{cases}
\]

In the equation above, *q* means the number of minor cycles that should be passed from the current position to proceed as far as the *logical_offset*. *r* means number of buckets that should be passed after passing a minor cycle. Therefore, to move to the position, *q* minor cycles and *r* buckets should be passed. Since *q* and *r* are calculated from the *logical_offset* which is obtained from the index table and since the variables (mcLen, bktNo, dskLen) of the equations above can be obtained from the control information, the client can get all the information needed to calculate the *physical_offset* within the bucket.
Based on the tuning position calculation method, Figure 3 shows the formal description of MDEI data access protocol. The variable $tuningPos_i$ represents the next tuning position of $i$th disk.

![MDEI data access protocol](image)

**Figure 3. MDEI data access protocol**

4. Experiments

We compare our index scheme MDEI with MHash and Exponential index schemes. Table 2 summarizes the experiment settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>number of data items</td>
<td>5000</td>
</tr>
<tr>
<td>bktSize</td>
<td>size of a bucket</td>
<td>1 KB</td>
</tr>
<tr>
<td>dataSize</td>
<td>size of a data item</td>
<td>128 bytes</td>
</tr>
<tr>
<td>indexSize</td>
<td>size of an index entry</td>
<td>4 bytes</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Zipf parameter</td>
<td>1.0</td>
</tr>
</tbody>
</table>

We use the Zipf distribution in our simulation which is typically used to model skewed data access pattern [9]. The Zipf parameter $\theta$ represents the skewness of data access pattern. As the Zipf parameter $\theta$ increase from 0.0 to 1.5, the skewness of data access patterns is also increased. Figure 4 and Figure 5 show the average access latency and the average tuning time respectively as we vary data access pattern (i.e. Zipf Parameter $\theta$). Figure 4 shows that the average latency of MDEI is shown to be a lot smaller than those of MHash and Exponential methods. The access latency of MDEI is one fourth of that of MHash when the zipf parameter is 1.5. This is because MHash limited the replication number of data items whereas MDEI dynamically increases the replication number of data items depending on the skewness of the data items.

![Figure 4. Average access latency versus data access distribution](image)

![Figure 5. Average tuning time versus data access distribution](image)

When the data access pattern is not skewed, the average tuning time of MDEI is a little higher than that of MHash. However, as the data access pattern is getting more skewed in Figure 5, the average tuning time of MDEI is getting close to that of MHash and is shown to be a lot better than that of Exponential methods. Since we developed MDEI scheme for the skewed data access patterns, this performance tendency is acceptable.

Figure 6 and Figure 7 show the average access latency and the average tuning time respectively depending on number of data items. The average access latency and the average tuning time increase as the number of data items increases. Compared with Exponential and MHash schemes, MDEI has the smallest average access latency, but its average tuning
time is not better than MHash scheme. The best performance on the average access latency is achieved because MDEI is constructed on multi-disk program which is very good to reduce the access latency.

Figure 6. Average access latency versus number of data items

Figure 7. Average tuning time versus number of data items

However, the above reason makes mobile clients tune into all the disks, resulting in a little higher tuning time than MHash. The difference of average tuning time between MDEI and MHash is not larger than one bucket time. On the other hand, the difference of access latency is almost 100 buckets time when the number of data items is 10000. This result shows that the overall performance of MDEI scheme is better than the other two schemes as the number of data items increases.

Figure 8. Average access latency versus data item size

Figure 9. Average tuning time versus data item size

Figure 8 and Figure 9 show the average access latency and the average tuning time respectively depending on the size of a data item. The size of data item varies from 16 bytes to 256 bytes. The average access latency increases as the data item size increases. This is because the broadcast cycle becomes longer as the data item size increases. Although its average tuning time is a little worse than that of MHash, the average access latency of MDEI becomes much better than those of MHash and Exponential schemes as the size of data item increases.

5. Conclusions

We proposed MDEI (Multi-disk Exponential Index), which is a single channel index scheme that functions with multi-disks. MDEI is the only multi-disk based index scheme that effectively reduces the average access time when the data access probability is skewed. As the data access pattern is getting skewed,
MDEI gives the better performance than MHash and Exponential index schemes. The performance analysis shows that MDEI scheme gives the average access latency much lower than MHash and Exponential index schemes. It also shows that the average tuning time of MDEI is getting close to that of MHash as the data access pattern is getting skewed.

Acknowledgement

This research was supported in part by grant No.10016775 under the supervision of Korea Automotive Technology Institute from the next-generation new growth engine program of Ministry Of Commerce and Industry and Energy in Korea and in part by grant No.R01-2006-000-10536-0 from the Basic Research Program of the Korea Science & Engineering Foundation

6. References


