SAMBox: A Smart Asynchronous Multi-Channel Black-Box for Wireless Data Broadcast

Xiaofeng Gao[1], Yan Shi[2], Jiaofei Zhong[3], Xuefei Zhang[3], and Weili Wu[3]

[1]Department of Computer Science and Engineering,
Shanghai Jiao Tong University, PRC. E-mail: gao-xf@cs.sjtu.edu.cn
[2]Department of Computer Science and Software Engineering,
University of Wisconsin - Platteville, USA. E-mail: shiy@uwplatt.edu
[3]Department of Computer Science, University of Texas at Dallas, USA.
E-mail: {fayjiao, xuefei.zhang, weiliwu}@utdallas.edu

Abstract

In this paper, we design a B$^+$-tree based mechanical auto-construction system for asynchronous multi-channel wireless data broadcast. We name it as ***SAMBox*** (***S****mart* ***A****synchronous* ***M****ulti-channel black* ***Box***), which is an integrated framework. Given the number of available channels and the data set to be broadcast, the framework can automatically construct an optimal broadcast program with best index tree structure, channel assignment, as well as index and data allocation, such that the overall performance of the data broadcast system will become optimal according to different client requirements. Theoretical analysis are also provided to prove the efficiency of SAMBox.

Key Words: Data Broadcast; B$^+$-Tree Indexing

1 Introduction

Recently, 4G networks together with smartphones make mobile computing within a wide area possible. Mobile users prefer to access public information like stock price, real-time traffic, and weather information through a wireless connection. *Wireless Data Broadcast* is an attractive solution to disseminate data efficiently because of its scalability and flexibility [6]. In data broadcast systems, data are broadcast periodically from a *Base Station* (BS) on some channels within a transmission range. Clients within this range can freely access the data.

Energy conservation and access efficiency are two main performance concerns for wireless data broadcast systems. A mobile device usually supports two operation modes, *active mode* and *doze mode*, to facilitate the energy conservation because of battery limitation. Two metrics have been proposed to measure the energy conservation and access efficiency respectively [8]: *Tuning time*: The total time when a client is in active mode; and *Access latency*: The time elapsed from the moment a client requests some data to the moment when all the requested data are retrieved.

To reduce tuning time, a good way is to use index. By listening to certain index information, clients will be able to compute the time they need to wait until the data requested arrives. They can then turn into doze mode during the waiting time and tune in right before the target data arrive. Several indexing techniques such as B$^+$-tree, alphabetic Huffman tree, exponential indexing and hashing table have been applied to the wireless data broadcast environment. To reduce access latency, an intuitive way is increasing the throughput of the broadcast. Using more channels for data broadcast can naturally increase the overall throughput. Hence, considering data broadcast in multi-channel wireless environment becomes a natural extension for research in this area.

However, both solutions proposed above bring new problems. In general, inserting indices will increase the length of a broadcast cycle and consequently make average access latency longer; on the other hand, indexing is critical to decrease tuning time. Therefore, a good balancing between access latency and tuning time should be found. Meanwhile, the introduction of multi-channel also poses new questions such as: how to design an index scheme for data on multiple channels; how to allocate channels to indices and data; and how to assign pointers and design index structures.

In this paper, we consider a global optimization for efficient data broadcast scheme with B$^+$-Tree based distributed index under asynchronous multi-channel wireless communication. We build a completely automatic framework named ***SAMBox***: ***S****mart* ***A****synchronous* ***M****ulti-channel black* ***Box***. It can auto-construct a complete data broadcast system under any wireless communication, and will compute the best index structure, channel allocation, as well as data allocation such that the system will be the most efficient broadcasting system to its clients.

The rest of the paper is organized as following: in Section 2 we introduce related works. In Section 3 we illustrate the system architecture for our design. In Section 4 we describe SAMBox construction with five processes. In Section 5 we discuss system performance theoretically. Finally, Section 6 gives a conclusion.

2 Related Work

Researches on wireless data broadcast mainly focus on data allocation and index designs. For data scheduling, Acharya et al. [1] proposed a "broadcast disk" scheme to group data by similar access probabilities. Vaidya et al. [16] optimized the average access latency under nonuniform data access distribution. Vlajic et al. [17] gave an optimized data broadcast scheme in systems of hierarchical cellular organization.

For indexing design, [10] presented a signature technique for information filtering. Imielinski et al. [7] proposed flexible index and hash-based index. They later customized distributed B$^+$-tree indexing [8]. Xu et al. presented an *exponential index* scheme in [20]. Chen et al. [5] and Shivakumar et al. [15] discussed how to construct an imbalanced index tree to minimize the average tuning time. Yao et al. [21] designed a hash function to facilitate skewed data access probabilities.

Multi-channel broadcast enables further reduction of access latency. Prahakara et al. [13] presented a air cache method to allocate data on multi-channels based on their popularity factors. Yee et al. [23] gave a near-optimal approximation to minimize the average access latency. Ardizzoni et al. [3] proposed dynamic programming to optimally allocate skewed data on multi-channels with flat broadcast per channel. Saxena et al. [14] developed an on-line broadcast scheduling scheme to handle the volatility of the data. Chen et al. [4] discussed disseminating time-constrained service data through multi-channel broadcast.

Various index techniques have been developed for multi-channel broadcast, including both index design and index allocation. There are two popular categories of index allocation. One is to interleave index with data on multiple channels. Examples include [2], [12], and [22]. Another is to separate channels as index channels and data channels, like [9], [18], and [19].

3 System Model

Let D denote the set of t data items to be broadcast in a program, $D = \{d_1, d_2, \cdots, d_t\}$. For simplicity, we assume that the keys for D are consecutively increasing. P denotes the probability set for D, say, each p_i is the access frequency of d_i, and $\sum_{i=1}^{t} p_i = 1$. Let *bucket* denote the minimum logical unit in data transmission process. Each data item may have different sizes according to application constraints, and let s_i denote the size of d_i (measured by bucket), which can be intuitively viewed as the "length" of d_i on time axis. Each datum is recognized by their primary key.

A BS periodically broadcasts data set D on multi-channels. To reduce tuning time, a B$^+$-tree Index scheme is involved in SAMBox. We use a tree T to index every datum by its primary key, and define k as the maximum number of branch for each intermediate node in this tree. A full B$^+$-tree should be a k-ary balanced tree. Let L be the depth of T. According to [22], we consider *depth-first* index layouts. Similar as the *distributed index* method in [8], T will be "cut" at the l^{th} level: nodes from level 1 to level l are *replicated part*, while others from level $l+1$ to level L are *non-replicated part*. We also append *control index* to replicated parts to connect every branch together.

Data and indices are partitioned onto different channels to avoid data interleaving and speed up data retrieval process. Let N denote the number of available channels for a BS, in which m channels are used for broadcasting index and n channels for data items. $N = m + n$. Let $\mathbf{C}$ denote the channel groups for data items, $\mathbf{C} = \{C_1, C_2, \cdots, C_n\}$, and $\mathbf{I}$ for indices, $\mathbf{I} = \{I_1, I_2, \cdots, I_m\}$. Each C_i (I_i) contains the data (indices) broadcast on the i^{th} data (index) channel.

3.1 System Architecture

SAMBox system only needs the data information D with access probability P and length S, and available channel number N. It will decide:

(1) Which B$^+$-tree should be applied for index? At which level should it be cut to form a distributed index? (determine k and l);

(2) How many channels should be used to broadcast indices and data? (determine m and n);

(3) How to allocate indices and data onto N channels? (determine data and index allocation).

Figure 1 illustrates the whole system architecture for our design with 11 elements, 5 of which will be discussed in this paper.

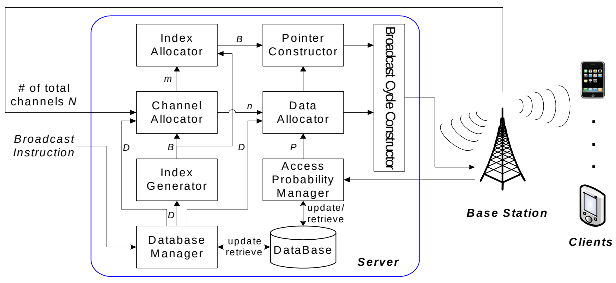

Figure 1: System Architecture for Data Broadcast

The flow of SAMBox is shown as follows. First, the server collects the information of data group and available channels for broadcast. Here data are retrieved from local DBMS by a *Database Manager*. Each data item has different client access probability, which are generated by *Access Probability Manager*, using statistics of historical records from BS. Next, according to data information, *Index Generator* will generate an efficient k-ary B$^+$-tree, and then generate a distributed index sequence cut at l^{th} level. After getting the length of index tree and data group, *Channel Allocator* will split N channels into two parts: index channels I of size m and data channels C of size n. After that, *Data Allocator* will assign D onto n channels with appropriate permutation. *Index allocator* will put the index group onto m channels. After each bucket has its fixed allocation, *Pointer Constructor* will generate values within each pointer figuring out the correct location it points. Finally, a *Broadcast Cycle Constructor* will merge index channels and data channels together, make a complete broadcast cycle and send it to BS. Data will continuously broadcast as radio waves, and clients can download their required data by accessing corresponding channels.

3.2 Bucket and Pointer Design

Bucket is the minimum logical unit for both indices and data. Each bucket should have a **head** segment and **payload** segment. The **head** segment contains the following information:

bucket_type	data or index bucket flag.
bucket_id	id of this bucket.
bucket_sq	sequence number in a *bcycle*.
bucket_length	length of the bucket.
bucket_optional	auxiliary space for future usage.

Intuitively, the minimum logical unit is one index node, so we set the size of payload be the information stored in an index. If a bucket is a data bucket, then payload stores information of such datum, else if it is an index bucket, then the payload stores several pointers on index tree to depict locations of its children. The size of a bucket is the size of an index node. Note that a datum may be split into several buckets.

In traditional disk storage, an index contains several pointers, each of which points to the address of the target index/data item as its children. The pointer only consumes several bits to record the location of target. Different from that, in wireless data broadcast system, the "address" for an index (data item) is a "location" on the time axis. Thus, previous literature set this pointer as an OFFSET from current moment, which guides clients to tune out for OFFSET time and tune in again at the moment when target data appears. In our paper, a pointer should contain the following items for asynchronous multi-channel environment:

pointer_key	for client to find search direction.
target_id	id of the bucket pointed to.
target_channel	channel id where the target resides.
target_sq	target sequence number in a *bcycle*.
target_blength	length of the target channel.

Therefore, a pointer is a five-tuple containing all necessary information to find a bucket. A client can follow the pointer to find its requested data. Each index bucket contains several pointers, depending on the B$^+$-tree design. Figure 2 is an example bucket storing an index A in a 3-ary B$^+$-tree.

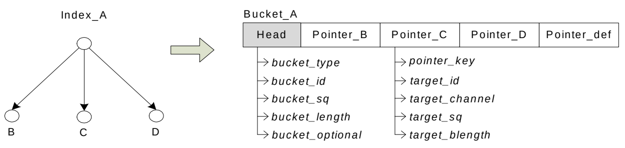

Figure 2: An Index Bucket in a 3-ary B$^+$-Tree

4 System Design

4.1 Distributed B$^+$-Tree Index Generator

We adopt the main idea in [8] to generate a k-ary B$^+$-Tree cut at the l^{th} level, with a different pointer structure. Fig. 3 is an example with $k = 3$ and $l = 2$. We denote each node as B_i^j, means the j^{th} index on the i^{th} level of T. Nodes above l are the *replicated part*, named as *control index*, while nodes below l are the *non-replicated part*, named as *search index*. L is the height of T. Then we transform the index search tree into an index sequence with three steps:

(1) Cut T at the l^{th} level. On level $l+1$, there are at most k^l non-replicated subtrees, rooted at B_{l+1}^1, $B_{l+1}^2, \cdots, B_{l+1}^R$, where R is the total number of nodes on the $l+1^{th}$ level of T. We use $\triangle_i$ to represent each subtree. For instance, in Fig. 3, $\triangle_1$ is a subtree rooted at B_3^1, with three children B_4^1, B_4^2, and B_4^3.

(2) Traverse T. Let $\text{PATH}(B_i^j)$ be a path from root B_1^1 to B_i^j ($\{B_1^1, \cdots, B_i^j\} \backslash \{B_i^j\}$), and $\text{LCA}(B_i^j, B_k^l)$ be the *least common ancestor* of B_i^j and B_k^l. Let V_i be a *distributed path* for $\triangle_i$. Then V_1=$\text{PATH}(B_{l+1}^1)$, V_i=$\text{PATH}(B_{l+1}^i) \backslash \text{PATH}(B_{l+1}^{i-1})$+$\text{LCA}(B_{l+1}^i, B_{l+1}^{i-1})$. E.g., in Fig. 3, V_4=$\{B_1^1, B_2^2\}$ for B_3^4. Next, we generate the distributed index sequence, denoted as $\mathbb{B}$ from B_1^1. We traverse each V_i, followed by a depth-first traversal of subtree $\triangle_i$, defined by $\text{DFT}(\triangle_i)$. Thus, $\mathbb{B} = \{V_1, \text{DFT}(\triangle_1), V_2, \text{DFT}(\triangle_2), \cdots, V_R, \text{DFT}(\triangle_R)\}$.

Fig. 4 is an example of $\mathbb{B}$ for index tree in Fig. 3. Here $t = 81$, $k = 3$, $l = 2$. There are totally 48 indices, 12 as control indices and 36 as search indices.

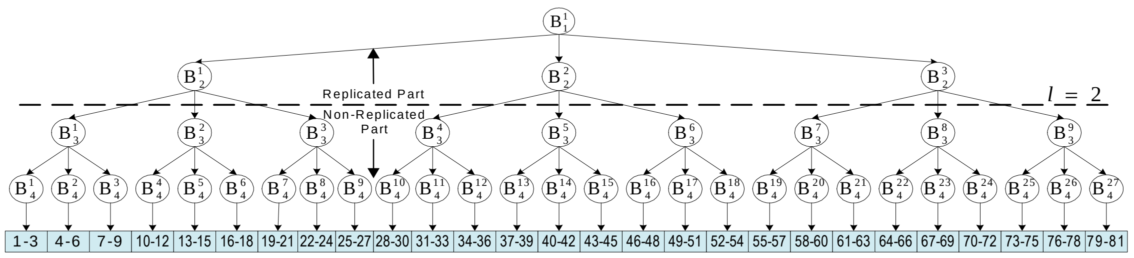

Figure 3: An Example of a 3-ary B$^+$-tree cut at the 2^{rd} level

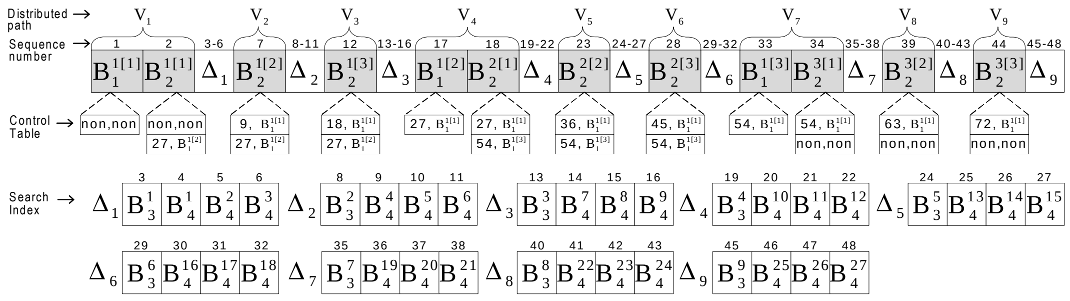

Figure 4: An Example of $\mathbb{B}$ with Control Tables

In Fig. 4 the number above each block is the sequence number. Block $\boxed{\triangle_i}$ is a depth-first traverse for $\triangle_i$, illustrated in search index sequence. V_i is the distributed path for $\triangle_i$, represented as grey blocks. Since each control index appears k times, we use $B_i^{j[1]}, \cdots, B_i^{j[k]}$ to represent each appearance of B_i^j.

(3) Append control tables. For each $B_i^{j[x]}$ in V_g, Let $\{B_1^{1[x_1]}, B_2^{j_2[x_2]}, \cdots, B_{i-1}^{j_{i-1}[x_{i-1}]}\}$ be the index sequence in $\text{PATH}(B_i^{j[x]})$, where each x_r is the x_r^{th} appearance for $B_r^{j_r}$ along $\mathbb{B}$ $(1 \leq x_i \leq k)$. Let $\text{MAX}(B_i^j)$ denote the maximum key value of data items indexed by B_i^j, and similarly we can define $\text{MAX}(\triangle_g)$. Then $B_i^{j[x]}$ has a control table as follows:

$$\begin{array}{ll}
\big(\text{MAX}(\triangle_{g-1}), B_1^{1[1]}\big), & \leftarrow 1^{st} \text{ entry} \\
\big(\text{MAX}(B_2^{j_2}), B_1^{1[(x_1+1)\%k]}\big), & \leftarrow 2^{rd} \text{ entry} \\
\cdots, & \cdots \\
\big(\text{MAX}(B_r^{j_r}), B_{r-1}^{j_{r-1}[(x_{r-1}+1)\%k]}\big), & \leftarrow r^{th} \text{ entry} \\
\cdots, & \cdots \\
\big(\text{MAX}(B_i^{j}), B_{i-1}^{j_{i-1}[(x_{i-1}+1)\%k]}\big). & \leftarrow i^{th} \text{ entry}
\end{array}$$

Each entry gives faster direction for clients to find datum which does not belong to current branch. For more explanation please refer [8]. After finishing index generation, we need to set up each bucket head. For index $B_i^{j[x]} \in \mathbb{B}$, we can set every exponent except *bucket_sq* (we will assign it after index allocation).

4.2 Global Optimal Channel Allocator

If we have N available channels, to avoid interleaving of data and indices, we divide N into m index channels and n data channels. Clients first access index channels to find the direction of required datum, and then access data channel to download data. Here we face a contradiction: if we assign more channels for indices, then access latency for index searching will reduce, while the access latency for data downloading will increase. To find a good balance, we need a global optimization method to determine m and n. Based on our observation, we find that there are three parameters affecting the performance of the data broadcast system: the k-ary structure of the index tree k, the cut level of the distributed index scheme l and the number of index channels m. Thus we define a client-oriented objective function for our optimization as

$$F(k, l, m) = a \cdot E(AL) + b \cdot E(TT) \tag{1}$$

where a and b can be adjusted by system users, and $E(AL)$ and $E(TT)$ are the average expected access latency and tuning time. F is a function of k, l, and m. By computing minimum F we can find the corresponding (k, l, m) tuple as the optimal solution (We will explain the calculation in Sec. 5). Thus we can setup m and n accordingly.

4.3 Dynamic Weight Data Allocator

We design an algorithm named *Dynamic Weight-Schedule*, using similar idea as [9]. The main difference is we use a dynamic threshold other than $1/n$ to guarantee accuracy after each iteration. The detailed description is shown in Alg. 1.

In Alg. 1, C_1 contains minimum number of data items, with highest access probability, while C_n contains maximum number of data items, with lowest ac-

Algorithm 1 Dynamic Weight-Schedule

```
   Input: $D$, $P$, $\mathbf{C}$;
   Output: Data Permutation for each $C_i \in \mathbf{C}$.
1: Sort $d_i \in D$ by $\frac{p_i}{s_i}$ in descending order. Reorder $D$ as $D = \{d'_1, d'_2, \cdots, d'_t\}$.
2: Set $ave=0$, $j=1$, $p=1$, $thre=\frac{1}{n}$, $C_i=\emptyset$, $i\in[1,n]$.
3: for $i = 1$ to $t$ do
4:     if $ave \le thre$ then
5:         $ave = ave + p'_i$; $C_j = C_j \cup \{d'_i\}$;
6:     else
7:         $p-=ave$; $ave=0$; $thre=\frac{p}{n-j}$; $j++$; $i--$;
8:     end if
9: end for
```

cess probability. Thus the higher the access probability is, the more times this data item can be repeated within a certain time. Let *bcast* denote the broadcast sequence of one round of data on C_i. After data allocation, we can set up head for each data bucket.

4.4 Wrap-Around Index Allocator

After getting $\mathbb{B}$, an *Index Allocator* allocate the index buckets onto m index channels. It equally splits $\mathbb{B}$ into m parts, and separately allocates each part onto one channel based on McNaughton's wrap-around rule [11], which is proven to be an optimal solution for scheduling problem. This algorithm is described in Alg. 2, in which *ave* is the average length of each I_i. We also set *bucket_sq* value for each index bucket.

Algorithm 2 McNaughton's Wrap-Around Method

```
   Input: $\mathbb{B}$, $\mathbf{I}$;
   Output: Index Permutation for each $I_i \in \mathbf{I}$.
1: Set $ave = \left\lceil \frac{|\mathbb{B}|}{m} \right\rceil$, $I_i = \emptyset$, $\forall 1 \le i \le n$.
2: for $i = 1$ to $m$ do
3:     for $j = 1$ to $ave$ do
4:         $x = (i-1) \times ave + j$; $I_i = I_i \cup \{B_x\}$; $B_x.bucket\_sq = j$;
5:     end for
6: end for
```

Alg. 2 is simple and yet can reduce the access latency dramatically by $\frac{1}{m}$. We also need to assign value for pointers inside each index after index allocation.

4.5 Broadcast Cycle Generator

Since data can only be updated between successive broadcasts, we need to adjust the length of data on each channel such that all channels for one program have the same length. Similar as [19], we use *least common multiple* (LCM) among all channels as the common length for each C_i. Data on C_i will repeat corresponding times due to this length. We name this length as *bcycle*. Define function $\textsc{lcm}(\cdot)$ the LCM of all inputs. Now, we have that $bcycle = \textsc{lcm}(|C_1|, \cdots, |C_n|, \lceil |\mathbb{B}|/m \rceil)$. For asynchronous environment, LCM is used as a *global cut* for consistency.

5 Performance Analysis

5.1 Evaluation for Access Latency

Let us first consider the access latency for one index channel and n data channels $(C_1, \cdots, C_n)$. All indices on the index channel can be divided into $\mathbb{B}_1, \cdots, \mathbb{B}_R$ blocks, where $\mathbb{B}_i = \{V_i, \textsc{dft}(\triangle_i)\}$, for $1 \le i \le R$. We denote $P_1, \cdots, P_R$ as their access probabilities. The access probability of one block can be computed by summing the access frequencies of all data its leaf index nodes point to, that is, $P_i = \sum_{j\in\mathbb{B}_i} p_j, i = 1, \cdots, R$. Let v be the average length of V_i; u the average length of $\mathbb{B}_i$, $u = |\mathbb{B}|/R$. The length of each C_j is denoted as $|C_j|, j = 1, \cdots, n$.

Access latency consists of two parts: waiting time to find the index pointer to the data, and the time used to locate the data and download it. We use $E_I(AL)$ and $E_D(AL)$ to represent their means respectively.

Lemma 1. *The average index access latency is*
$$E_I(AL) = \sum_{i=1}^{R} \left(\sum_{w=1}^{R-1} \left(wu + \tfrac{v}{2}\right) \cdot P_{(i+w)\%R} \cdot \tfrac{1}{R} + \tfrac{v}{2} \cdot P_i \right).$$

Proof. Consider a client that first tunes in the index channel at block $\mathbb{B}_i$. It then waits w blocks (not including $\mathbb{B}_i$) to eventually reach the index containing the pointer to the datum it requires at $\mathbb{B}_{i+w}$. Fig. 5 is an example to illustrate the whole process.

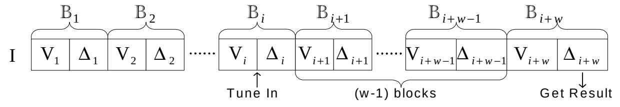

Figure 5: An Example for a Client's Activity

When $w \ge 1$, this period can be divided into three phases: 1) the client tunes in block $\mathbb{B}_i$, taking an average of $u/2$; 2) it waits through $(w-1)$ complete blocks, taking $(w-1)u$; and 3) it finds the pointer to the datum, which only exists in $\triangle$, so the average waiting is $v + (u - v)/2$. The mean of this period is:

$$E_I(AL|_{b=i,d=w}) = \frac{u}{2} + (w-1)u + (v + \frac{u-v}{2}) = wu + \frac{v}{2}$$

$w = 0$ is possible only when the client tunes in during the broadcast of V_i, and the data pointer is right in the following $\triangle_i$. In such a case, this period has only phases 1) and 3), and its mean becomes:

$$E_I(AL|_{b=i,d=0}) = \frac{v}{2} + \frac{u-v}{2} = \frac{u}{2}$$

Based on the above two equations and law of total expectation, we can derive the average access latency for finding the pointer to the requested datum as:

$$\begin{aligned} E_I(AL) &= \sum_{i=1}^{R}\sum_{w=0}^{R-1} E(AL|_{b=i,d=w}) \cdot P(b=i, d=w) \\ &= \sum_{i=1}^{R}\Bigg(\sum_{w=1}^{R-1} E(AL|b{=}i, d{=}w)P(b{=}i, d{=}w) \\ &\qquad +E(AL|b{=}i, d{=}0)P(b{=}i, d{=}0)\ \Bigg) \\ &= \sum_{i=1}^{R}\left(\sum_{w=1}^{R-1}\left(wu + \frac{v}{2}\right)P_{(i+w)\%R}\frac{1}{R} + \frac{v}{2}P_i\right)\square \end{aligned}$$

Lemma 2. *If data are broadcast on n channels, then the average data access latency* $E_D(AL) = \sum_{j=1}^{n}\sum_{i\in C_j}\left(\frac{|C_j|}{2} + s_i\right)\cdot p_i$.

Proof. After a client finds the pointer to the datum d_i required, it will hop to the corresponding data channel C_j, wait for d_i to come and download it. The average waiting time is $|C_j|/2$, and the downloading time is s_i. Hence, according to the law of total expectation, we have the above conclusion. $\square$

In Lemma 1, we consider one time unit as the time needed to broadcast one index, while in Lemma 2 we consider one time unit as the time needed to broadcast a datum whose $s_i = 1$. However, these two sizes are not equivalent. The size of an index is decided by the size of head and the number of pointers in it. A pointer contains five elements in our design, which takes around 0.1KB. A head also contains five elements, so we can set its size the same as a pointer. Thus, totally an index takes $(k+2)/10$ KB. On the other hand, the size of a datum (s_i) is usually measured by KB. Therefore, we have $E(AL) = (k+2)/10 \cdot E_I(AL) + E_D(AL)$.

If there are m index channels, the number of blocks w a client needs to wait after tuning in can be reduced to $\frac{R}{m}$. Then we have Theorem 1.

Theorem 1. *Average access latency for SAMBox is* $\frac{k+2}{10}\sum_{i=1}^{R}\left(\sum_{w=1}^{\lceil\frac{R}{m}\rceil-1}\left(wu+\frac{v}{2}\right)\frac{P_{(i+w)\%R}}{R}+\frac{vP_i}{2}\right)+\sum_{j=1}^{n}\sum_{i\in C_j}\left(\frac{|C_j|}{2}+s_i\right)p_i$.

5.2 Evaluation for Tuning Time

Theorem 2. *The average tuning time for SAMBox is* $\frac{k+2}{10}\left(\sum_{i=1}^{R}\frac{u+|\triangle_i|}{|\mathbb{B}|} + L - \frac{l}{2} + 1\right) + 1 + \sum_{i=1}^{t} s_i p_i$

Proof. The tuning time for one data retrieval includes:
1. Client tunes in an index channel with 1 unit time.
2. If the first visited index is not a control index, the client may need to hop twice to find the right starting index. This has a probability of $\sum_{i=1}^{R}|\triangle_i|/|\mathbb{B}|$. Thus totally it takes $2\cdot\sum_{i=1}^{R}|\triangle_i|/|\mathbb{B}|$. If the first visited index is a control index, then the client only needs to hop one time, which takes $1\cdot\sum_{i=1}^{R}(u-|\triangle_i|)/|\mathbb{B}|$.
3. Then, the client searches for the pointer to the datum on index channels. The average number of visited index buckets here is $\frac{l}{2} + (L-l) = L - \frac{l}{2}$.
4. The client tunes in the target data channel.
5. The client downloads the datum with average downloading time $\sum_{i=1}^{t} s_i p_i$.

To summarize, we can get the average tuning time

$$E(TT) = \frac{k+2}{10}\Big(\sum_{i=1}^{R}\frac{u+|\triangle_i|}{|\mathbb{B}|} + L - \frac{l}{2}\Big) + \frac{k+12}{10} + \sum_{i=1}^{t} s_i p_i.\square$$

5.3 Global Optimization

To minimize $F(k, l, m)$, we need to find the best (k, l, m) tuple. There are several existing algorithms, such as powell method, genetic algorithm or simulated annealing. In our system, we notice that the total number of possible (k, l) pairs is limited given a data set. Since the number of pointers at the bottom level of an index tree should equal to the size of data, the scale of k should be bounded by t. Assume an index tree has at least three levels. Since any internal node of an B$^+$-tree should have at least two children, k is bounded as $2 \leqslant k \leqslant \sqrt[3]{t}$. The cut level l is bounded between 1 and $L - 1$, and L is decided by t and k. Thus, we can get the total number of possible (k, l) pairs once the data set is decided. In such a case, using enumeration to find the optimal (k, l, m) should be more time-efficient than most of the optimization algorithms. Table 6 is a sample optimization result of (k, l, m) given 10 data sets of sizes 1,000 to 7,000, and available channels from 6 to 30. Server can setup the system parameter according to this table.

6 Conclusion

In this paper, we design an efficient automatic system named ***SAMBox***: ***S****mart* ***A****synchronous* ***M****ulti-channel black* ***Box*** for wireless data broadcast. A server only needs to setup some parameters, input the data sets and the available channel numbers, then SAMBox will automatically generate a distributed index sequence and allocate data items and index sequences onto available channels to guarantee the best performance. We also provide theoretical analysis.

Table 1: Optimal (k, l, m) for different N and t

$N\backslash t$	1000	2000	3000	4000	5000	6000	7000
6	(4,4,2)	(3,6,2)	(4,5,2)	(3,1,2)	(4,1,2)	(4,1,2)	(4,1,2)
8	(3,6,3)	(3,6,3)	(4,5,3)	(4,5,3)	(3,7,3)	(4,1,2)	(4,1,2)
10	(3,6,3)	(3,6,3)	(4,5,3)	(4,5,3)	(3,7,3)	(3,7,3)	(4,1,2)
12	(2,9,5)	(3,6,4)	(3,7,4)	(3,7,4)	(3,7,4)	(3,7,4)	(3,8,4)
14	(2,9,5)	(3,6,4)	(3,7,4)	(3,7,4)	(3,7,4)	(3,7,5)	(3,8,5)
16	(2,9,5)	(3,6,5)	(3,7,5)	(3,7,5)	(3,7,5)	(3,7,4)	(3,8,5)
18	(2,9,6)	(3,6,5)	(3,7,5)	(3,7,5)	(3,7,5)	(3,7,5)	(3,8,5)
20	(2,9,6)	(3,6,6)	(3,7,6)	(3,7,6)	(3,7,5)	(3,7,6)	(3,8,6)
22	(2,9,8)	(3,6,7)	(3,7,6)	(3,7,6)	(3,7,6)	(3,7,6)	(3,8,6)
24	(2,9,7)	(3,6,6)	(3,7,6)	(3,7,7)	(3,7,7)	(3,7,7)	(3,8,7)
26	(2,9,8)	(3,6,7)	(3,7,7)	(3,7,7)	(3,7,7)	(3,7,7)	(3,8,7)
28	(2,9,8)	(3,6,7)	(3,7,7)	(3,7,8)	(3,7,7)	(3,7,7)	(3,8,7)
30	(2,9,8)	(3,6,8)	(3,7,7)	(3,7,7)	(3,7,7)	(3,7,7)	(3,8,7)

Acknowledgement: This work is supported by the National Natural Science Foundation of China (grant number 61033002).

References

[1] S. Acharya, R. Alonso, M. J. Franklin and S. B. Zdonik, Broadcast disks: Data management for asymmetric communications environments, *SIGMOD Conference*, pp.199-210, 1995.

[2] D. Amarmend, M. Aritsugi and Y. Kanamori, An air index for data access over multiple wireless broadcast channels, *ICDE'06*, pp.135, 2006.

[3] E. Ardizzoni, A. A. Bertossi, S. Ramaprasad, R. Rizzi and M. V. S. Shashanka, Optimal skewed data allocation on multiple channels with flat broadcast per channel, *IEEE Trans. Comput.*, 54(5):558-572, 2005.

[4] C.-C. Chen, C. Lee and S.-C. Wang, On optimal scheduling for time-constrained services in multichannel data dissemination systems, *Inf. Syst.*, 34(1):164-177, 2009.

[5] M. Chen, K. Wu and P. Yu, Optimizing index allocation for sequential data broadcast in wireless mobile computing, *TKDE*, 15(1):161-173, 2003.

[6] T. Imielinski and B. R. Badrinath, Mobile wireless computing: Challenges in data management, *Communications of ACM*, 37:18-28, 1994.

[7] T. Imielinski, S. Viswanathan and B. R. Badrinath, Power efficient filtering of data on air, *EDBT'94*, pp.245-258, 1994.

[8] T. Imielinski, S. Viswanathan and B. Badrinath, Data on air: Organization and access, *TKDE*, 9(3):353-372, 1997.

[9] S. Jung, B. Lee and S. Pramanik, A tree-structured index allocation method with replication over multiple broadcast channels in wireless environments, *TKDE*, 17(3):311-325, 2005.

[10] W.-C. Lee and D. L. Lee, Using signature techniques for information filtering in wireless and mobile environments, *Distrib. Parallel Databases*, 4(3):205-227, 1996.

[11] J. Leung, L. Kelly and J. H. Anderson, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, 2004.

[12] S.-C. Lo and A. Chen, Optimal index and data allocation in multiple broadcast channels, *ICDE'00*, pp.293-302, 2000.

[13] K. Prabhakara, K. Hua and J. Oh, Multi-level multi-channel air cache designs for broadcasting in a mobile environment, *ICDE'00*, 167-176, 2000.

[14] N. Saxena and M. C. Pinotti, On-line balanced k-channel data allocation with hybrid schedule per channel, *MDM'05*, pp.239-246, 2005.

[15] N. Shivakumar and S. Venkatasubramanian, Efficient indexing for broadcast based wireless systems, *Mob. Netw. Appl.*, 1(4):433-446, 1996.

[16] N. H. Vaidya and S. Hameed, Scheduling data broadcast in asymmetric communication environments, *Wireless Networks* 5:171-182, 1996.

[17] N. Vlajic, C. Charalambous, D. Makrakis, Wireless data broadcast in systems of hierarchical cellular organization, *ICC'03*, pp.1863-1869, 2003.

[18] A. B. Waluyo, B. Srinivasan and D. Taniar, Global indexing scheme for location-dependent queries in multi channels mobile broadcast environment, *AINA'05*, pp.1011-1016, 2005.

[19] S. Wang and H.-L. Chen, Tmbt: An efficient index allocation method for multi-channel data broadcast, *AINA'07*, pp.236-242, 2007.

[20] J. Xu, W.-C. Lee, X. Tang, Q. Gao and S. Li, An error-resilient and tunable distributed indexing scheme for wireless data broadcast, *TKDE*, 18(3):392-404, 2006.

[21] Y. Yao, X. Tang, E.-P. Lim and A. Sun, An energy-efficient and access latency optimized indexing scheme for wireless data broadcast, *TKDE*, 18(8):1111-1124, 2006.

[22] W. G. Yee and S. B. Navathe, Efficient data access to multi-channel broadcast programs, *CIKM'03*, pp.153-160, 2003.

[23] W. G. Yee, S. B. Navathe, E. Omiecinski and C. Jermaine, Efficient data allocation over multiple channels at broadcast servers, *IEEE Trans. Comput.*, 51(10):1231-1236, 2002.