The TopK Scheme for the Energy-Saving Data Organization in Broadcast-Based Wireless Environments*

Jun-Hong Shen, and Ye-In Chang
Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, R.O.C
E-mail: {*shenjh, changyi*}*@cse.nsysu.edu.tw*

Abstract

Wireless broadcasting is an efficient way to deliver information to mobile clients. Due to power limit for the portable units, how to design an energy-saving organization is a key issue. Imielinski *et al.* have proposed two hashing-based schemes, *Hashing A* and *Hashing B*, to save energy in the progress of getting data of interest. However, these two hashing-based schemes have the *directory miss* phenomenon. To improve the directory miss phenomenon further, in this paper, we propose the *TopK* scheme which is a multiple-hashing-function-based scheme. From our simulation study, we show that the performance of *TopK* is better than that of *Hashing B* in terms of the average access time and the average tuning time.

keywords: data broadcast, power conservation, selective tuning, wireless network.

1 Introduction

Due to the feature of asymmetry in communications, it is an efficient way to deliver information to mobile clients via wireless broadcast [1], *e.g.,* stock quotes and weather information. The main advantage of this method is that it is independent of the number of clients tuning to the channel, *i.e., scalability* [1, 8]. By broadcasting the file periodically, mobile clients can specify predefined condition to filter out the data they wanted [4].

Due to the feature 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 for examining data packets. This is a waste of energy, since on an 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 [4, 5], *i.e., selective tuning.* As a consequence, it is advantageous to use some special data organizations, say indexed (or hash-based or signature-based) data organizations, to broadcast data over wireless channels to guide mobile units to get the relevant information.

For a file being broadcasted on a channel, the following two parameters are of concern [4]: (1) Access time: The average waiting time for clients to get the required data (2) Tuning time: The amount of time spent by a portable unit listening to the channel, which will determine its power consumption.

There have been many strategies for reducing power consumption. For the uniform broadcast in which the same data record appears once in a broadcast cycle, the flexible indexing [4], the hashing-based schemes [4], the tree-based indexing [3, 5], signature schemes, the mixture of the index tree and the signature scheme, and the mixture of the hashing and the index tree scheme [10] have been proposed. A skewed index tree based on data popularity patterns was considered in [2]. For energy efficient filtering of nonuniform broadcast in which data records are broadcast according to the access frequency, [8] proposed indexing schemes. The above schemes considered that there is only one broadcast channel. However, broadcasting data can be over multiple channels; therefore, [7] focused on index and data allocation.

Since in the wireless broadcast, the access time is affected by the size of the file, adding the index increases the access time. If the size of the index is too

*This research was supported in part by the National Science Council of Republic of China under Grant No. NSC94-2213-E-110-003 and by National Sun Yat-Sen University.

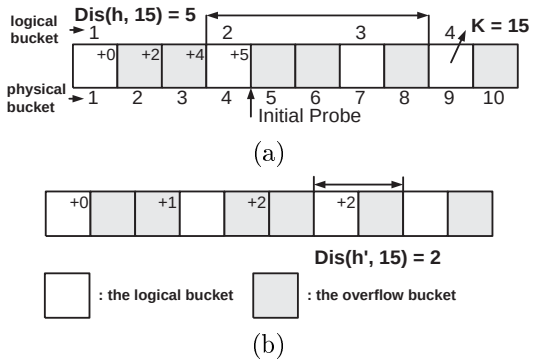


Figure 1: A comparison of *Hashing A* and *Hashing B* (for $K = 15$): (a) *Hashing A*; (b) *Hashing B*.

large, the whole broadcast file increases largely. In this case, using the hashing-based scheme is a better choice than using the index-based scheme. For power conservation, Imielinski *et al.* [4] have proposed two hashing-based schemes, *Hashing A* and *Hashing B*. *Hashing B* improves the *directory miss* in *Hashing A* by taking the minimum overflow into consideration, where the directory miss is that client's initial probe comes before the requested data item but after the bucket that has its corresponding offset.

However, in Imielinski *et al.*'s two hashing-based schemes [4], if the differences between the minimum overflow and the other overflows are large extremely or the small overflows appear near the rear part of the broadcast file, both schemes have a poor performance. Therefore, in this paper, we propose the *TopK* scheme, which is a multiple-hashing-function-based scheme, to reduce the probability of the directory miss further. From our simulation study, we show that the performance of *TopK* is better than that of *Hashing B* in terms of the average access time and the average tuning time.

The rest of paper is organized as follows. In section 2, we give a brief survey of the hashing-based schemes. In section 3, we present our proposed *TopK* scheme. In section 4, we study the performance of *TopK* by simulation. Finally, a conclusion is given in section 5.

2 Background

In [4], Imielinski *et al.* have proposed two hashing schemes, *Hashing A* and *Hashing B*, to help clients get data of interest. The smallest logical unit of the broadcast in those schemes is called a *bucket* composed of packets, the physical unit of the broadcast.

Figure 1 shows *Hashing A* and *Hashing B*, in which all of the broadcast files are hashed by $h(K) = (K \ mod \ 4) + 1$ and have the same size. Each of the first 4 physical buckets (called the *designated* bucket) in *Hashing A* stores the offset to the logical bucket in the $h(K)$th physical bucket as shown in Figure 1-(a). For example, in Figure 1-(a), physical bucket 4 stores an offset, 5, to logical bucket 4, which contains the data item of key 15. The remaining buckets contain an offset to the beginning of the next cycle. Moreover, $Dis(h, K)$ is the difference between the address of the physical bucket in which K resides and the designated bucket for K, for a given hash function h. If clients tune into the broadcast channel in the range covered by $Dis(h, K)$, a directory miss occurs. Therefore, the smaller the $Dis(h, K)$ is, the lower the probability of the directory miss is, and the shorter the access time is. In fact, it is the same as that the smaller the total offset is, the shorter the access time is.

To reduce the probability of the directory miss, *Hashing B* modifies the hashing function $h(K)$ to $h'(K)$ as follows:

$$h'(K) = \begin{cases} h(K) & \text{if } h(K) = 1 \\ (h(K) - 1)(1 + MO) + 1 & \text{if } h(K) > 1, \end{cases}$$

where MO denotes the minimum number of overflows in the whole file. The $h'(K)th$ physical bucket contains the offset of the logical bucket $h(K)$. Figure 1-(b) shows *Hashing B*, where $MO = 1$. In Figure 1-(a) and Figure 1-(b), $Dis(h, 15)$ $(= 5) > Dis(h', 15)$ $(= 2)$; therefore, the probability of the directory miss of *Hashing A* is higher than that of *Hashing B*. Consequently, *Hashing B* has improved the performance of *Hashing A*.

3 The TopK Scheme

To reduce the probability of the directory miss in the hashing schemes, we propose the *TopK* scheme, which is a multiple-hashing-function-based scheme.

3.1 Assumptions

This paper focuses on the wireless environment. Some assumptions should be restricted in order to make our work feasible [1]. These assumptions include: (1) Data appears once in the whole broadcast file, and is broadcast over a reliably single channel. (2) Data will not updated during the current broadcast cycle. (3) When a client switches to the public channel, it can retrieve buckets immediately. (4) A query result contains only one bucket.

3.2 The Basic Idea

The basic idea of *TopK* is to use the cutlines to divide the broadcast file into several regions, which can have the different value of the minimum overflow. Therefore, each region can have the different hashing function to determine the positions of the designated buckets.

The basic steps of *TopK* are described as follows. Given a parameter, t ($t < N$), which determines the number of the cutlines, we then divide the whole file with N logical buckets into $(t+1)$ regions $(R_1, R_2, \ldots, R_{t+1})$ by considering the descending order of differences of overflows. For each region R_i $(1 \leq i \leq t+1)$, we design the related hashing function by using the value of MO_i in this region R_i, where MO_i is the value of the minimum overflow in the ith region. Since the value of MO_i in each region R_i can be different, we can have different hashing functions for those $(t+1)$ regions. (Note that in *Hashing B*, there is only one value of MO.)

3.3 The TopK Scheme

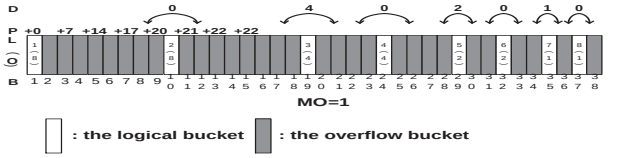
For our illustrations clearly, we let O_i be the number of overflows which follow the related logical bucket and $D[i] = |O_{i+1} - O_i|$, $1 \leq i < N$. Moreover, we let c_i denote the logical bucket of the ith cutline and $c_0 = 1$.

Formally, the algorithm of this scheme is shown in Figure 2. From lines 01–05, the differences of overflows between two adjacent logical buckets for N logical buckets are calculated out. (Note that D stores the differences of overflows for N logical buckets and DI stores the corresponding indices of D.) And then we sort those differences in a descending order (line 06). (Note that in the SortI procedure, when the swap of $D[i], D[j]$ occurs, the corresponding $DI[i], DI[j]$ must also be swapped.) Finally, we find t cutlines by considering the descending order of the differences of overflows (lines 07–11), where $CS[i]$ stores the position of the ith cutline. (Note that CS may be in disorder after a for loop, because the algorithm of *TopK* does not determine the cutlines in a certain direction. Therefore, CS must be sorted in line 09.)

Let's use one example to illustrate this scheme. Figure 3 shows an example of *Hashing B*, where $O_1 = O_2 = 8$, $O_3 = O_4 = 4$, $O_5 = O_6 = 2$, $O_7 = O_8 = 1$, $MO = 1$, $h(K) = (K \ mod \ 8) + 1$, $TShift$ $(= \sum_{i=1}^{8} Shift_i = 0 + 7 + 14 + 17 + 20 + 21 + 22 + 22) = 123$, MO is the minimum number of overflows in the whole file,

```
01  for i := 1 to (N − 1) do
02  begin
03    DI[i] := i;
04    D[i] := |O_{i+1} − O_i|;
05  end;
06  SortI(D, DI);        (* in a descending order *)
07  for i:= 1 to t do
08    CS[i] := DI[i] + 1;
09  Sort(CS);        (* in an ascending order *)
10  for i := 1 to t do
11    c_i := CS[i];
```

Figure 2: The algorithm of *TopK*



D: the difference of overflows between two adjacent logical buckets
P: the offset to the designated logical bucket
L: the logical bucket number
O: the number of overflows which follow the related logical bucket
B: the physical bucket number

Figure 3: An example of *Hashing B*

$TShift$ is the total summation of $Shift_i$ and $Shift_i$ is the offset to logical bucket i.

For the same input, Figure 4 shows the result of *TopK*, where $t = 2$ and $TShift = 5$. Following the algorithm from lines 01–05, differences of overflows are $D[1] = 0, D[2] = 4, D[3] = 0, D[4] = 2, D[5] = 0, D[6] = 1$ and $D[7] = 0$, and the corresponding indices of D are $DI[1] = 1, DI[2] = 2, DI[3] = 3, DI[4] = 4, DI[5] = 5, DI[6] = 6$ and $DI[7] = 7$. (Note that after sorting D in a descending order, we have $D[1] = 4, D[2] = 2, D[3] = 1, D[4] = 0, D[5] = 0, D[6] = 0, D[7] = 0$ and $DI[1] = 2, DI[2] = 4, DI[3] = 6, DI[4] = 7, DI[5] = 1, DI[6] = 5, DI[7] = 3$.) From lines 07–09, we then have $CS[1] = 3$ and $CS[2] = 5$. Accordingly, from lines 10–11, it turns out that $c_1 = 3$ and $c_2 = 5$.

Therefore, given $t = 2$, we have divided the whole file into 3 regions. The region before c_1 is R_1 and $MO_1 = 8$. The region starting from c_1 and ending before c_2 is R_2 and $MO_2 = 4$. The region starting from c_2 to the end of the file is R_3 and $MO_3 = 1$. Then, we use MO_1 $(= 8)$ to determine which physical bucket to store $Shift$, when $1 < h_1(K) \leq c_1$ $(= 3)$. MO_2 $(= 4)$ and MO_3 $(= 1)$ are used, when $c_1 < h_1(K) \leq c_2$ $(= 5)$ and $c_2 < h_1(K) \leq 8$, respectively.

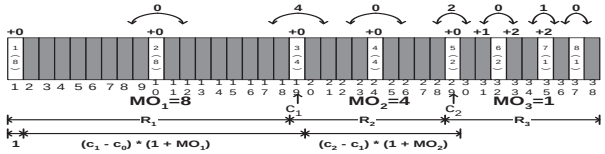


Figure 4: An example of *TopK* ($c_1 = 3$ and $c_2 = 5$)

Table 1: The hashing functions in *TopK*

Hashing Function	Condition
$h_1(K) = (K \ mod \ N) + 1$	$h_1(K) = 1$
$= (K \ mod \ 8) + 1;$	
$h_2(K) = (h_1(K) - c_0) \times (1 + MO_1) + 1$	$1 < h_1(K) \leq 3$
$= (h_1(K) - 1) \times (1 + 8) + 1$	
$= (h_1(K) - 1) \times 9 + 1;$	
$h_3(K) = (c_1 - c_0) \times (1 + MO_1) +$	$3 < h_1(K) \leq 5$
$(h_1(K) - c_1) \times (1 + MO_2) + 1$	
$= (3 - 1) \times (1 + 8) +$	
$(h_1(K) - 3) \times (1 + 4) + 1$	
$= 18 + (h_1(K) - 3) \times 5 + 1;$	
$h_4(K) = (c_1 - c_0) \times (1 + MO_1) +$	$5 < h_1(K) \leq 8$
$(c_2 - c_1) \times (1 + MO_2) +$	
$(h_1(K) - c_2) \times (1 + MO_3) + 1$	
$= (3 - 1) \times (1 + 8) + (5 - 3) \times (1 + 4) +$	
$(h_1(K) - 5) \times (1 + 1) + 1$	
$= 18 + 10 + (h_1(K) - 5) \times 2 + 1;$	

* $c_0 = 1$

Table 2: The $(t+2)$ hashing functions for $(t+1)$ regions and t cutlines

$h_1(K) = (K \ mod \ N) + 1;$	if $h_1(K) = 1$
$h_2(K) =$	if $1 < h_1(K) \leq c_1$
$(h_1(K) - c_0) \times (1 + MO_1) + 1;$	
$h_3(K) = (c_1 - c_0) \times (1 + MO_1) +$	if $c_1 < h_1(K) \leq c_2$
$(h_1(K) - c_1) \times (1 + MO_2) + 1;$	
...	
$h_i(K) =$	if $c_{i-2} < h_1(K) \leq c_{i-1}$
$\sum_{x=1}^{i-2} ((c_x - c_{x-1}) \times (1 + MO_x)) +$	
$(h_1(K) - c_{i-2}) \times (1 + MO_{i-1}) + 1;$	
...	
$h_{t+1}(K) =$	if $c_{t-1} < h_1(K) \leq c_t$
$\sum_{x=1}^{t-1} ((c_x - c_{x-1}) \times (1 + MO_x)) +$	
$(h_1(K) - c_{t-1}) \times (1 + MO_t) + 1;$	
$h_{t+2}(K) =$	if $c_t < h_1(K) \leq N$
$\sum_{x=1}^{t} ((c_x - c_{x-1}) \times (1 + MO_x)) +$	
$(h_1(K) - c_t) \times (1 + MO_{t+1}) + 1;$	

Table 1 lists all hashing functions for *TopK* used in Figure 4. The part $(1 + MO_1)$ of $h_2(K)$ means that in this region $(1 < h_1(K) \leq c_1)$, we store the value of $Shift$ for every $(1 + MO_1)$ buckets. (Note that we must add one more 1 in each h_i, since the physical bucket is numbered from 1.) In $h_3(K)$, the first part $(c_1 - c_0) \times (1 + MO_1) + 1$ (as shown in Figure 4) is the total buckets from the beginning of the broadcast cycle to the designated bucket $(= 19)$ of c_1 $(= 3)$, and the second part $(h_1(K) - c_1) \times (1 + MO_2)$ is used to store the value of $Shift$ for every $(1 + MO_2)$ buckets after the designated bucket of c_1 and ending at the designated bucket $(= 29)$ of c_2 $(= 5)$. In $h_4(K)$, the first part $(c_1 - c_0) \times (1 + MO_1) + (c_2 - c_1) \times (1 + MO_2) + 1$ is the total buckets from the beginning of the broadcast cycle to the designated bucket of c_2, and the second part $(h_1(K) - c_2) \times (1 + MO_3)$ is used to store the value of $Shift$ for every $(1 + MO_3)$ buckets after the designated bucket of c_2. In general, for N logical buckets and t cutlines, there are $(t+2)$ hashing functions for $(t+1)$ regions, which are listed in Table 2.

In Figure 4, there are 2 cutlines $(t = 2)$, and the related 4 hashing functions are listed in Table 1. For example, in Figure 4, $h_1(K) = 2$ falls in the range of $1 < h_1(K) \leq 3$; therefore, the physical bucket which stores $Shift_2$ to logical bucket 2 is $(2-1) \times 9 + 1 = 10$ (by $h_2(K)$). For logical bucket 4, we use $h_3(K)$ to calculate the physical bucket which stores $Shift_4$ and the result is $18 + (4 - 3) \times 5 + 1 = 24$. Obviously, the comparison of *Hashing B* and *TopK* shows that $TShift$ $(= 5)$ of *TopK* is less than that $(= 123)$ of *Hashing B*, so we can conclude that *TopK* performs better than *Hashing B*.

3.4 Access Protocol

When tuning into the channel for searching the data item of key K, a client retrieves the corresponding hashing function from the current bucket according to key K [4]. The client then waits for the designated bucket of key K. After getting the designated bucket, the client has the offset to reach the requested data.

4 Performance

In this section, we study the performance of the proposed *TopK* scheme. We first make a comparison of *TopK* and *Hashing B* [4], and then make a comparison of *TopK* and a tree-based approach, *SL* [3].

4.1 Generation of Overflows

First, given LB, the amount of logical buckets, we generate the overflow patterns for LB logical buckets by the uniform distribution with the range between O_{min} and O_{Max}. O_{min} and O_{Max} denote the minimum size of overflows and the maximum one, respectively. Based on the result of the above generation of the overflow patterns for LB logical buckets, we then consider four kinds of distributions of those LB overflow patterns, including the increasing, decreasing, convex and concave distributions. Obviously, we have the hashing function $h(K) = (K \ mod \ LB) + 1$, and the amount of total physical buckets, B, equals $LB + \sum_{K=1}^{LB} O_K$.

The parameter, t, is given to determine the number of cutlines. Since the number $(= t + 2)$ of hashing functions is more than that $(= 2)$ of *Hashing B*, the bucket size of our *TopK* is somewhat larger than that

of *Hashing B*. Therefore, as compared to *Hashing B*, the average access time and the average tuning time of *TopK* are multiplied by the factor f ($= \frac{DH+\frac{(t+2)}{2}}{DH+1}$). ($DH$ denotes the ratio of the data part to the part of storing hashing functions in a bucket.)

4.2 Performance Analysis of Access Time and Tuning Time

In our simulation, we assume that the whole logical buckets ($= LB$) are requested. (Note that since the distribution of the overflows for *TopK* and *Hashing B* is the same, we do not consider the access of the overflow buckets in our experiment.) The access time AT_K "per key" K, $1 \leq K \leq LB$, is as follows [4].

$$\begin{aligned} AT_K &= \frac{Dis(h,K)}{B} \times \left(B + \frac{1}{2} \times Dis(h,K)\right) + \\ &\quad \left(1 - \frac{Dis(h,K)}{B}\right) \times \left(\frac{B - Dis(h,K)}{2} + Dis(h,K)\right) \\ &= \frac{Dis(h,K)}{B} \times \left(B + \frac{1}{2} \times Dis(h,K)\right) + \\ &\quad \left(1 - \frac{Dis(h,K)}{B}\right) \times \frac{B + Dis(h,K)}{2}, \end{aligned} \quad (1)$$

where $\frac{B-Dis(h,K)}{2}$ is the mean probing position outside the displacement area. In equation (1), the first term is for the case of the directory miss, and the second one is for the case of the non-directory miss. The average access time for the broadcast is $\frac{\sum_{K=1}^{LB} AT_K}{LB}$ [4].

On the other hand, the tuning time TT_K "per key" K, $1 \leq K \leq LB$, is calculated as follows [9].

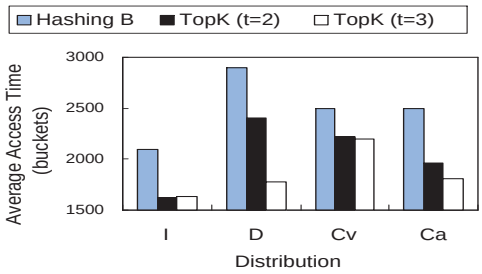
$$TT_K = 3 \times \frac{Pos(K)}{B} + 4 \times \left(1 - \frac{Pos(K)}{B}\right),$$

where $Pos(K)$ is the difference between the beginning of the broadcast cycle and the physical bucket containing the offset to the logical bucket which contains key K. The average tuning time for the broadcast is $\frac{\sum_{K=1}^{LB} TT_K}{LB}$.

4.3 Simulation Results: TopK *vs.* Hashing B

In this comparison, we let $LB = 100$, $O_{min} = 1$, $O_{Max} = 50$, and $DC = 50$, resulting in $B = 2608$. Moreover, we give $t = 2$ and $t = 3$, resulting in $f = 1.01961$ and $f = 1.02941$, respectively. The simulation results discussed later are the average of 100 cases. We will consider $t = 2$ and $t = 3$ for those four kinds of overflow distributions.

Figure 5 shows the average access time of *TopK* and *Hashing B*. In Figure 5, under these four cases of the distributions, the average access time of *TopK* is shorter than that of *Hashing B* no matter $t = 2$ or $t = 3$. This is because the minimum overflow in *Hashing B* is always a constant (called MO_f) for a given



I: the increasing distribution Cv: the convex distribution
D: the decreasing distribution Ca: the concave distribution

Figure 5: A comparison of the average access time of *TopK* and *Hashing B*

overflow pattern no matter under what kinds of distributions. However, in *TopK*, since the values of the minimum overflows are dependent on the positions of the cutlines, they can be larger than MO_f. Moreover, the average access time is affected by $Dis(h,K)$ and $Dis(h,K)$ is affected by the value of the minimum overflow in the region where $h(K)$ locates. Therefore, *TopK* has the adaptability to reduce the average access time.

In the case of the decreasing distribution, there is the largest difference of the average access time between *Hashing B* and *TopK*. This is because *TopK* uses the cutlines to reduce the average access time. Moreover, *Hashing B* has the worst performance in this case. Therefore, in the case of the decreasing distribution, there is the largest difference of the average access time between *Hashing B* and *TopK*.

Note that when the number of cutlines is increased, the size of a bucket is increased as well. In general, as long as the benefit resulting from the increase of the number of cutlines exceeds the cost resulting from the increase of the size of a bucket, the increase of the number of cutlines is beneficial on reducing the average access time. As shown in Figure 5, while in the case of the increasing distribution, the average access time is increased from $t = 2$ to $t = 3$ in *TopK*. In addition, we observe that the average access time of *TopK* is always shorter than that of *Hashing B* in any kind of these four cases of different distributions, no matter $t = 2$ or $t = 3$.

Figure 6 shows the comparison of the average tuning time. Based on the analysis of the tuning time, we observe that as the value of $Pos(K)$ is increased, the tuning time is decreased. Since the positions of the logical buckets are fixed under a case of distributions, the less the displacement between the designated bucket and the logical bucket is, the larger the

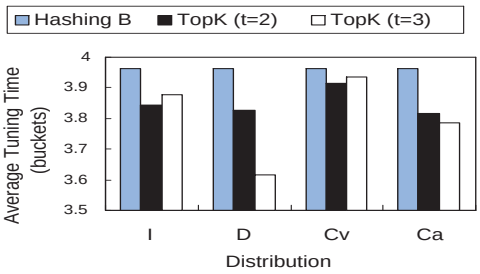


Figure 6: A comparison of the average tuning time of *TopK* and *Hashing B*

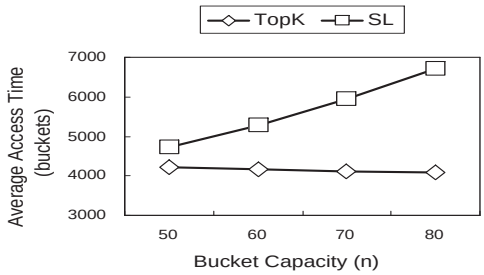


Figure 7: A comparison of the average access time of *TopK* and *SL* under the uniform distribution

value of $Pos(K)$ is, and the shorter the tuning time is. Therefore, the average tuning time of *TopK* is shorter than that of *Hashing B*, no matter $t = 2$ or $t = 3$.

Note that in fact, we have tried another three multiple-hashing-function-based schemes in [9] to determine the positions of cutlines. However, from our simulation study, we have observed that *TopK* outperforms the other schemes.

4.4 Simulation Results: TopK *vs.* SL

In this simulation, we assume that the whole keys are requested; that is, the whole buckets are accessed. We assume that the number of real data items of a bucket in *TopK* is the same as that in *SL*, and that the storage size of each hashing function is equal to that of a data item. While comparing *TopK* with *SL*, we multiply the access time of *TopK* by $\frac{n+(t+2)}{n}$ to normalize, where n is the capacity of a bucket.

In *SL*, we consider a balanced index tree and assume that each node has the same number of children. The index tree of *SL* has two parts: (1) The replicated part constitutes the top r levels of the index tree; (2) the non-replicated part consists of the rest levels [3]. In the broadcast cycle, the number of appearance times of each node in (1) equals the number of its children. On the other hand, the remaining index nodes will appear only once in the broadcast cycle. We assume that the clients uniformly tune into the broadcast channel. Since the number of the replicated levels of the index tree affects the access time, we use the optimal value of r from [3, 5]. The formula for calculating the average access time for *SL* is from [3, 5].

Figure 7 shows the comparison of the average access time of *TopK* and *SL*, where $LB = 150$, $t = 3$, $O_{min} = 40$, $O_{Max} = 50$, B is between 6795 and 6921, and the average of Bs is 6852. In Figure 7, we can observe that the average access time of *TopK* is shorter than that of *SL* with increasing the capacity of a bucket.

5 Conclusion

In this paper, we have proposed the *TopK* scheme to improve the directory miss phenomenon in hashing schemes over wireless broadcast, which increases the access time and the tuning time. In our simulation study, we have considered four distributions of overflows, including the increasing, decreasing, convex and concave distributions. From our experimental results, we have shown that *TopK* performs better than *Hashing B* in terms of the average access time and the average tuning time in each of these four distributions of overflows.

References

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] J. H. Shen, "The Multiple-Hashing-Function-Based Schemes for Energy-Saving Data Organization in the Wireless Broadcast," *Master Thesis, Dept. of Computer Science and Eng., National Sun Yat-Sen University,* June 2001.

[10] 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.