

8-2006

An energy-efficient and access latency optimized indexing scheme for wireless data broadcast

Yuxia YAO

Nanyang Technological University

Xueyan TANG

Nanyang Technological University

Ee Peng LIM


Singapore Management University, eplim@smu.edu.sg

Aixin SUN

Nanyang Technological University

DOI: <https://doi.org/10.1109/tkde.2006.118>

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the [Databases and Information Systems Commons](#), and the [Numerical Analysis and Scientific Computing Commons](#)

Citation

YAO, Yuxia; TANG, Xueyan; LIM, Ee Peng; and SUN, Aixin. An energy-efficient and access latency optimized indexing scheme for wireless data broadcast. (2006). *IEEE Transactions on Knowledge and Data Engineering*. 18, (8), 1111-1124. Research Collection School Of Information Systems.

Available at: https://ink.library.smu.edu.sg/sis_research/126

This Journal Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

An Energy-Efficient and Access Latency Optimized Indexing Scheme for Wireless Data Broadcast

Yuxia Yao, Xueyan Tang, *Member, IEEE*, Ee-Peng Lim, *Senior Member, IEEE*, and Aixin Sun, *Member, IEEE*

Abstract—Data broadcast is an attractive data dissemination method in mobile environments. To improve energy efficiency, existing air indexing schemes for data broadcast have focused on reducing tuning time only, i.e., the duration that a mobile client stays active in data accesses. On the other hand, existing broadcast scheduling schemes have aimed at reducing access latency through nonflat data broadcast to improve responsiveness only. Not much work has addressed the energy efficiency and responsiveness issues concurrently. This paper proposes an energy-efficient indexing scheme called MHash that optimizes tuning time and access latency in an integrated fashion. MHash reduces tuning time by means of hash-based indexing and enables nonflat data broadcast to reduce access latency. The design of hash function and the optimization of bandwidth allocation are investigated in depth to refine MHash. Experimental results show that, under skewed access distribution, MHash outperforms state-of-the-art air indexing schemes and achieves access latency close to optimal broadcast scheduling.

Index Terms—Wireless data broadcast, energy conservation, latency, indexing, scheduling, mobile computing.

1 INTRODUCTION

THE rapid development of wireless communication technology and battery-powered portable devices is making mobile information services increasingly popular. It is envisaged that a variety of information will be accessible through mobile devices (e.g., laptop computers, PDAs, and mobile phones) from anywhere at any time [5]. In mobile environments, a base station or a satellite is often deployed to disseminate data to mobile clients through wireless channels. *Data broadcast* is an attractive dissemination method in such context. First, the bandwidth resource of wireless networks is scarce. Broadcast allows simultaneous accesses to the data by a massive number of clients and is, thus, preferable to point-to-point delivery. Second, most wireless systems are asymmetric in that the downlink communication capacity from the base station to the clients is much higher than the uplink capacity in the opposite direction. In push-based broadcast, data are disseminated proactively and the clients simply filter and pick the data they want, thereby alleviating the load on the uplink channel. As a result of these advantages, data broadcast is receiving much attention from both academic and industrial communities [1], [2], [3], [22].

The limited battery capacity of mobile clients makes energy efficiency a critical issue in the design of broadcast systems [20]. Mobile clients can operate in two different modes: *active mode* and *doze mode*. They can retrieve data

from broadcast channels in the active mode only. However, the clients have much higher rates of energy consumption in the active mode than in the doze mode. Therefore, to save energy, it is desirable for mobile clients to switch to the doze mode as much as possible when waiting for the requested data. The performance of broadcast systems is often characterized by two metrics: *access latency* and *tuning time* [10], [22]. Access latency refers to how fast the client can access the requested data. It reflects the responsiveness of the system. Tuning time, on the other hand, refers to the duration for which the client stays active. It measures the energy consumed by the client in the active mode. A good broadcast system should achieve both low access latency and low tuning time.

The tuning time can be reduced by means of *air indexing* [10]. The basic idea is to interleave the index information with data in the broadcast schedule to assist the client in locating data. Following the links in the index structure, the client alternates between the active and doze modes until the requested data arrive. However, most existing air indexing schemes were designed for flat broadcast in which all data items are broadcast at the same frequency (see Section 2 for details) [7], [9], [10], [22], [23]. This sacrifices responsiveness when client accesses are not uniformly distributed among data items. To reduce average access latency under nonuniform access distribution, popular data items should be broadcast more frequently than unpopular items. This is known as *nonflat data broadcast*. However, most existing schemes of nonflat broadcast scheduling did not consider air indexing [4], [17], [19]. Without indexing, the client has to stay continuously active and monitor the broadcast channel until the requested data arrive. This consumes significant amount of battery power and sacrifices energy efficiency.

• The authors are with the School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798.
E-mail: yaoyuxia@pmail.ntu.edu.sg,
[asxytang, aseplim, axsun]@ntu.edu.sg.

Different from existing work, our objective is to optimize responsiveness and energy efficiency in an integrated fashion. In this paper, we propose a novel indexing scheme called MHash for wireless data broadcast. MHash constructs the broadcast schedule using a two-argument hash function for indexing purpose. The two-argument nature of the hash function allows each data item to be mapped to an adjustable number of slots in the schedule, thereby enabling nonflat data broadcast. Under this framework, we further investigate the issues of generating hash functions that produce broadcast schedules free of unoccupied slots; improving access latency by properly spacing the broadcast instances of each data item in the schedule; and optimally allocating the bandwidth among data items. Experimental results show that under skewed access distribution, MHash outperforms state-of-the-art air indexing schemes and achieves access latency close to optimal broadcast scheduling.

The rest of this paper is organized as follows: Section 2 summarizes the related work. Section 3 presents the MHash indexing scheme and investigates a variety of issues to refine MHash. Section 4 describes the experimental setup and discusses the experimental results. Finally, Section 5 concludes the paper.

2 RELATED WORK

Data broadcast in mobile environments has received much attention in recent years. The simplest broadcast scheme is flat broadcast, in which all data items are scheduled in a round-robin manner. Although simple, flat broadcast shows poor access latency when data accesses are not uniformly distributed. Acharya et al. [4] proposed a scheme called broadcast disks that takes into consideration of nonuniform data access distribution. In this approach, the items with similar access rates are grouped together to form logical disks. Each disk is assigned a relative broadcast frequency: those with 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. There also exists work on optimal broadcast scheduling for nonuniform data accesses [19]. The average access latency is shown to be minimized when each item is allocated a bandwidth fraction proportional to the square root of its access probability and the broadcast instances of each item are equally spaced in the schedule. Recent work has studied time-critical broadcast scheduling [25]. However, none of the above work has considered air indexing. Without indexing, tuning time is as high as access latency, which would result in significant waste of energy.

To cater for limited battery power, some air indexing techniques have been proposed to assist the client in predicting the arrival time of requested data [7], [10], [12], [16], [22], [23]. Lee and Lee [12] proposed 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 items in the broadcast schedule. This allows the client to check whether a requested item is in the frame by examining the signature only. However, a signature does not provide the arrival times of data items. Thus, when a match is found in a signature, the data items

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. Imielinski et al. [10] 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. Chen et al. [7] and Shivakumar and Venkatasubramanian [16] further showed that under non-uniform data access distribution, the average tuning time can be reduced by an imbalanced index tree. In their approaches, popular items are placed closer to the root of the index tree and unpopular items are placed deeper in the tree. Xu et al. [22], [23] extended tree-based indexes 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. Nonflat broadcast schedules generally do not have this property. Thus, when applied to nonflat broadcast, the indexes can only be built locally for short segments of broadcast, where each segment holds a sequence of items with increasing key values. These segments have to be searched sequentially in data accesses [8], [22], [23], [27]. As a result, the effectiveness of indexing diminishes significantly.

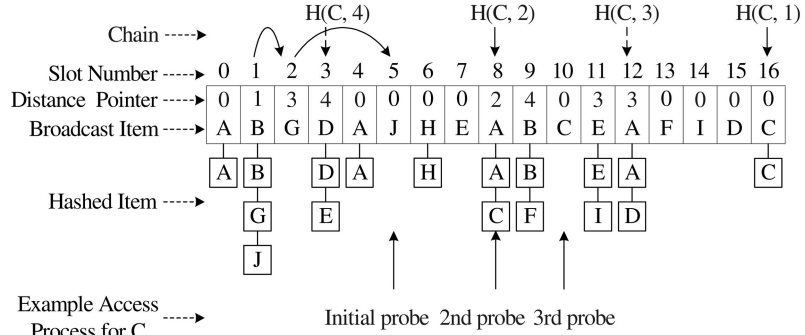
Besides tree-based indexes, hash functions can also be used for indexing purpose to map data items to the slots in the broadcast schedule [9]. A salient feature of hash-based index is that it eliminates the need to broadcast index structures (e.g., trees)—only a hash function is broadcast together with 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. Furthermore, hash-based indexes allow searching to start anywhere in the broadcast. In hash-based indexes, multiple items are likely to be hashed to the same slot. Imielinski et al. [9] resolved such collisions by pushing overflow items into succeeding slots and pushing forward the items originally hashed to the succeeding slots. This approach, similar to linear probing, introduces an extra tuning time of one slot to all the items hashed to the succeeding slots. Moreover, the indexing scheme proposed in [9] neither considered nonflat data broadcast nor addressed the problem of producing broadcast schedules free of unoccupied slots. In this paper, we propose a novel indexing scheme using a two-argument hash function. Our proposed scheme gracefully incorporates hash-based index with nonflat data broadcast. It naturally reduces both access latency and tuning time for popular items. We also propose a new chaining method for collision resolution to reduce the penalty in tuning time and investigate how to generate broadcast schedules without any unoccupied slot.

Other related work on air indexing and broadcast scheduling includes air indexing of spatial data [26], [28], semantic data broadcast [11], adaptive broadcast scheduling

k	$H(k,1)$	$H(k,2)$	$H(k,3)$	$H(k,4)$
A	0	8	4	12
B	1	9	5	13
C	16	8	12	3
D	12	3	16	8
E	3	11	7	15
F	9	1	13	4
G	1	9	5	13
H	6	14	10	1
I	11	3	15	6
J	1	9	5	13

□ Hashed slot □ Cheating slot

(a)



(b)

Fig. 1. An example of MHash. (a) Hash table. (b) Broadcast cycle.

[13], channel allocation of data broadcast [14], and client caching under data broadcast [24], [21]. They complement our study in various aspects.

3 MHASH INDEXING SCHEME

This section presents our MHash indexing scheme. We start with an overview of MHash in Section 3.1. Section 3.2 shows how to produce broadcast schedules free of unoccupied slots. Section 3.3 discusses the spacing of broadcast instances. Finally, Section 3.4 analyzes the optimal bandwidth allocation among different data items under MHash indexing.

3.1 Overview

The key idea of MHash is to construct an energy-efficient index that allows different items to be broadcast with different frequencies. We consider a system that repeatedly broadcasts a set of data items in *cycles*. A broadcast cycle consists of a sequence of *slots*, each of which accommodates one item. All slots in the cycle are numbered sequentially: 0, 1, 2, ..., $N - 1$, where N is called the *cycle length*. MHash first maps each item to a given number of M slots. The item is then placed and broadcast in a subset of these slots. To reduce average access latency, popular data items are placed in more slots than unpopular items, thus enabling non-flat data broadcast. M is a tunable parameter representing the maximum allowable number of times each item can be broadcast in one cycle. We refer to M as the *replication bound*.

We use a two-argument hash function $H(k, l)$ to map a data item to a list of slots, where k is the key of the item and l is a sequential identifier. The function maps the key to the slots $H(k, 1), H(k, 2), \dots, H(k, M)$. If the item is to be broadcast $c \leq M$ times in a cycle, it is then placed and broadcast in the first c slots on the list, i.e., $H(k, 1), H(k, 2), \dots, H(k, c)$. They are called the *hashed slots* of the item and we say that the item is *hashed* to these slots. As will be discussed shortly, choosing a prefix of the list allows the tuning time to be further reduced by pruning in data accesses. The remaining slots, i.e., $H(k, c + 1), H(k, c + 2), \dots, H(k, M)$, are called the *cheating slots*, since the item would not actually be broadcast in these slots.

It is likely that multiple items are hashed to the same slot. This is known as *collision*. Meanwhile, there might be some slots such that no item is hashed to them. We refer to these slots as *empty slots*. In the following, we propose a chaining method to resolve collisions. In Section 3.2, we shall investigate, under our collision resolution method, how to construct hash functions that produce broadcast schedules without any unoccupied slot where no item is placed.

To resolve collisions, all items hashed to the same slot are sorted in decreasing order of access probability. The first item (i.e., the most frequently accessed one) is placed in the hashed slot. The remaining items are sequentially placed in subsequent empty slots. To facilitate data accesses, a distance pointer is recorded in each slot to refer to the next slot accommodating an item with the same hashed slot. The distance pointer of the last item is set to 0. The purpose of broadcasting the items hashed to the same slot in decreasing order of access probability is to reduce the average access latency.

Fig. 1 shows an example of MHash indexing. The data items, listed in decreasing order of access probability, are A to J . Suppose the broadcast cycle consists of 17 slots. Item A is broadcast four times in the cycle, items B to E are broadcast twice each, and the remaining items are broadcast once each. Suppose the replication bound $M = 4$. Fig. 1a shows a hypothetical hash table. Accordingly, the items hashed to each slot are shown below the slot in Fig. 1b. As can be seen, slots 2, 5, 7, 10, 13, 14, and 15 are empty slots where no item is hashed. Items B, G , and J are all hashed to slot 1. To resolve the collision, B (the most popular item among B, G , and J) is placed in its hashed slot 1 and G and J are placed in empty slots 2 and 5, respectively. Slot 1 has a distance pointer 1 since the next item hashed to slot 1 (i.e., G) is broadcast one slot away in slot 2. Slot 2 has a distance pointer 3, indicating the next item hashed to slot 1 (i.e., J) is broadcast three slots away in slot 5. The distance pointer of slot 5 is 0 since J is the last item hashed to slot 1. As shown in Fig. 1b, the distance pointers link the items hashed to the same slot in a chain. Similarly, items D and E are both hashed to slot 3. So, D is placed in its hashed slot 3. Since empty slot 5 has been occupied, E is placed in the following empty slot 7.

Algorithm 1 describes the algorithm for data accesses. We illustrate the access process with the example in Fig. 1. Note that item C is broadcast twice in the cycle at slots 10 and 16, which correspond to hashed slots $H(C, 2) = 8$ and $H(C, 1) = 16$, respectively.¹ Suppose the client tunes to slot 5 at the initial probe and would like to access item C , given its key. The client first calculates the M slot numbers where the key is mapped (step 1 of Algorithm 1). Following the hash table in Fig. 1a, the slot numbers where C is mapped are 16, 8, 12, and 3. They are the potential locations of the target data item. These slot numbers are then sorted in increasing order of their distances ahead of the initial probing slot (step 2). In our example, the sorted list of slot numbers is 8, 12, 16 and 3. Intuitively, the client should tune to all of these slots sequentially to look for item C . However, we note that the slot numbers can be short-listed for searching purposes. For example, since $H(C, 2) = 8$ precedes $H(C, 3) = 12$ and $H(C, 4) = 3$ on the sorted list, $H(C, 3)$ and $H(C, 4)$ can be pruned. This is because if the target item is broadcast at least twice in the cycle, $H(C, 2)$ is a hashed slot and the client would be able to retrieve item C from it (possibly by following the distance pointers). On the other hand, if the item is broadcast fewer than two times in the cycle, all slots $H(C, l)$, where $l \geq 2$ are cheating slots and the client would not be able to find item C from any of them. So, in either case, there is no need for the client to tune to slot $H(C, 3)$ or $H(C, 4)$ after checking $H(C, 2)$. In general, if the sorted list of slot numbers is $H(k, l_1), H(k, l_2), \dots, H(k, l_M)$ where k is the key, and l_1, l_2, \dots, l_M is a permutation of $1, 2, \dots, M$, the client can get rid of all slots $H(k, l_i)$ where $\min_{1 \leq j \leq i-1} l_j < l_i$ (step 3). As will be shown in Section 4, this pruning technique makes tuning time grow slowly (logarithmically) with M .

Algorithm 1 Data Access

Input: Key k

Output: Target data item with key k

- 1: Calculate slot numbers $H(k, 1), H(k, 2), \dots, H(k, m)$;
- 2: Sort the slot numbers in increasing order of their distances ahead of the initial probing slot: $H(k, l_1), H(k, l_2), \dots, H(k, l_M)$, where l_1, l_2, \dots, l_M is a permutation of $1, 2, \dots, M$;
- 3: Remove all slot numbers $H(k, l_i)$ where $\min_{1 \leq j \leq i-1} l_j < l_i$;
- 4: Put remaining slot numbers in a searching set Q ;
- 5: **repeat**
- 6: Tune to the nearest slot $q \in Q$ ahead;
- 7: **if** Bcast[q].data.key = k **then**
- 8: Probe success and return Bcast[q].data;
- 9: **else**
- 10: Read the distance pointer d of q ;
- 11: **end if**
- 12: **if** $d > 0$ **then**
- 13: Insert $q + d$ to Q ;
- 14: **end if**
- 15: Remove q from Q ;
- 16: **until** Q is empty;
- 17: Probe failure;

1. For convenience, we shall use the same letter to denote an item and its key.

The slot numbers left after pruning constitute a searching set Q (step 4). The client repeatedly tunes to the nearest slot $q \in Q$ ahead. If the target data item is not found in slot q , the client reads the distance pointer d from the slot. If $d > 0$, Q is updated by replacing q with $q + d$. If $d = 0$, q is removed from Q (i.e., the item broadcast in slot q is the last item hashed to its hashed slot). The process continues until the target data item is found or Q becomes empty² (steps 5 to 16). The latter case leads to a failure of data access, i.e., there does not exist any item with the requested key in the broadcast schedule (step 17).

In our example, the initial searching set Q includes $H(C, 2) = 8$ and $H(C, 1) = 16$. The client first tunes to slot 8 and finds that the item broadcast in slot 8 is not item C . It then reads the distance pointer 2 and replaces 8 with 10 in Q . Now, the updated Q includes slot numbers 10 and 16. The client tunes to slot 10 and retrieves item C . Thus, after the initial probe at slot 5, the client only tunes to slots 8 and 10 in the access process and can switch to the doze mode in the other slots.

As seen from the example, a collision in hashing introduces some penalty to tuning time. In general, the performance of MHash indexing improves with decreasing collision rate of the hash function.³ It is also intuitive that the tuning time increases with the number of cheating slots. Since an item that is broadcast c times in the cycle has $M - c$ cheating slots, more frequently broadcast items would have lower tuning times. Therefore, if popular items are broadcast more frequently than unpopular items (to reduce average access latency), MHash naturally leads to less tuning time for popular items. This salient feature helps reduce the average tuning time.

The following parameters are needed by the client in data accesses: the hash function H , the cycle length N , and the replication bound M . These parameters can be recorded in the header of each bucket, which is the smallest accessible unit of broadcast [10], [22]. The number of slots in a bucket depends on the size of a data item. If a bucket contains multiple slots, the accesses to the slots therein involve the same bucket access physically.

3.2 Hole-Free Hash Function

An empty slot may be left unoccupied in the broadcast schedule. Fig. 2b shows the broadcast cycles derived from an example hash table shown in Fig. 2a. Slots 1 and 6 are left unoccupied in the first cycle because no item is hashed to or pushed to these slots due to collisions in the preceding slots. Meanwhile, items E and G , which are hashed to slots 9 and 10, have to be pushed forward to the second cycle. The existence of such unoccupied slots (referred to as *holes*) not only wastes bandwidth but also complicates the computation of distance pointers in subsequent cycles (e.g., slot 2 has different pointer values in the two cycles in Fig. 2b). Therefore, it is desirable to look for hole-free hash functions that can produce broadcast schedules without unoccupied slots. In this section, we show that a hole-free hash function can be constructed by injecting an offset into an *arbitrary*

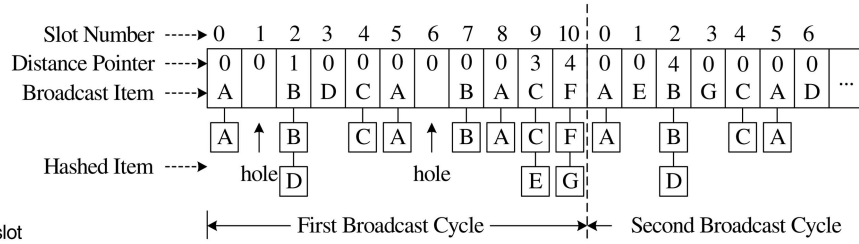
2. Note that the access process may extend to the next broadcast cycle if the initial probe is not at the beginning of a cycle.

3. The selection of a good hash function is application specific and is beyond the scope of this paper.

k	$H(k,1)$	$H(k,2)$	$H(k,3)$
A	0	5	8
B	2	7	4
C	4	9	6
D	2	7	4
E	9	4	0
F	10	5	1
G	10	5	1

□ Hashed slot □ Cheating slot

(a)

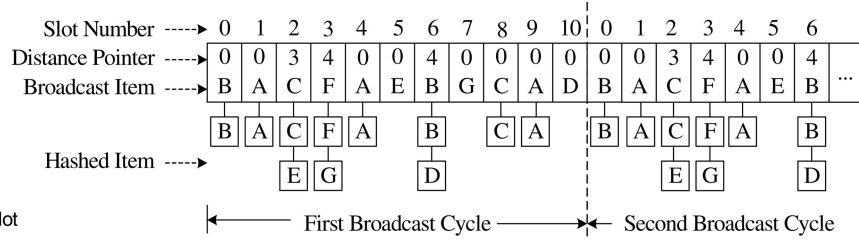


(b)

k	$H'(k,1)$	$H'(k,2)$	$H'(k,3)$
A	4	9	1
B	6	0	8
C	8	2	10
D	6	0	8
E	2	8	4
F	3	9	5
G	3	9	5

□ Hashed slot □ Cheating slot

(c)



(d)

Fig. 2. Generation of hole-free hash function. (a) Original hash table. (b) Holes in broadcast cycle. (c) New hash table. (d) Hole-free broadcast cycle.

hash function. For example, denote the hash function of Fig. 2a by $H(k, l)$. The holes in Fig. 2b would be eliminated if we use a new hash function

$$H'(k, l) = (H(k, l) - 7) \bmod 11,$$

where 11 is the cycle length. The new hash table is shown in Fig. 2c and the derived broadcast cycles are shown in Fig. 2d.

In general, consider n data items d_1, d_2, \dots, d_n to be broadcast c_1, c_2, \dots, c_n times respectively in a cycle of length $N = \sum_{i=1}^n c_i$. In MHash indexing, each item d_i has c_i hashed slots $H(d_i, 1), H(d_i, 2), \dots, H(d_i, c_i)$ between 0 and $N - 1$, and there are a total of N hashed slots for all items. Given a hash function H , let $f_i(H)$ be the number of items hashed to slot i , then $\sum_{j=0}^{N-1} f_j(H) = N$. Let $h_i(H)$ be the cumulative number of holes in slots 0 to i . Obviously, $h_0(H)$ depends on whether any item is hashed to slot 0, i.e.,

$$h_0(H) = \begin{cases} 0 & \text{if } f_0(H) > 0, \\ 1 & \text{if } f_0(H) = 0. \end{cases} \quad (1)$$

For each $1 \leq i \leq N - 1$, since, among the i slots from 0 to $(i - 1)$, $i - h_{i-1}(H)$ slots are occupied by data items, we have

$$\sum_{j=0}^{i-1} f_j(H) \geq i - h_{i-1}(H).$$

If the equality holds, i.e., $\sum_{j=0}^{i-1} f_j(H) = i - h_{i-1}(H)$, no item hashed to slots 0 to $(i - 1)$ is pushed forward to slot i for collision resolution. Otherwise, if $\sum_{j=0}^{i-1} f_j(H) > i - h_{i-1}(H)$, at least one item is pushed forward to slot i for collision

resolution. Note that slot i ($i > 0$) is a hole if and only if 1) no item is hashed to it (i.e., $f_i(H) = 0$), and 2) no item is pushed forward to slot i for collision resolution. Therefore, for each $1 \leq i \leq N - 1$,

$$h_i(H) = \begin{cases} h_{i-1}(H) & \text{if } f_i(H) > 0 \\ & \text{or } \sum_{j=0}^{i-1} f_j(H) > i - h_{i-1}(H), \\ h_{i-1}(H) + 1 & \text{if } f_i(H) = 0 \\ & \text{and } \sum_{j=0}^{i-1} f_j(H) = i - h_{i-1}(H). \end{cases} \quad (2)$$

Theorem 1 shows that a hole-free hash function can be constructed by injecting an offset into an arbitrary hash function.

Theorem 1. *Given any hash function $H(k, l)$, let b be the smallest index such that slots b to $N - 1$ are hole-free under H , then the new hash function $H'(k, l) = (H(k, l) - b) \bmod N$ is hole-free, i.e., for each $0 \leq i \leq N - 1$, $h_i(H') = 0$.*

Proof. See Appendix A for details. \square

3.3 Spacing between Broadcast Instances

If an item is broadcast multiple times in a cycle, its access latency, to a large extent, depends on how the slots broadcasting the item (called *broadcast instances*) are located in the cycle. Intuitively, if multiple broadcast instances are clustered in a short segment of the cycle, they would not reduce access latency much compared to a single broadcast instance. In general, the access latency of an item can be reduced by equalizing the spaces between successive broadcast instances of the item [19].

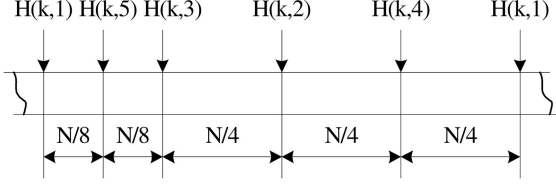


Fig. 3. Spacing between broadcast instances.

In this section, we construct hash functions that approximately equalize the spaces between broadcast instances in MHash indexing. Recall that each item is broadcast in the slots whose numbers are a prefix of the list $H(k, 1), H(k, 2), \dots, H(k, M)$, where k is the item key. Our idea is to let any prefix of length 2^m ($1 \leq m \leq M$) be a uniform partition of the broadcast cycle. When $m = 1$, the prefix consists of $H(k, 1)$ and $H(k, 2)$. To produce a uniform partition, the $H(k, 2)$ -to- $H(k, 1)$ space must be set at $\frac{1}{2}N$, where N is the cycle length. When $m = 2$, the prefix consists of $H(k, 1), H(k, 2), H(k, 3)$, and $H(k, 4)$. To produce a uniform partition, the $H(k, l)$ -to- $H(k, 1)$ spaces ($2 \leq l \leq 4$) must constitute a set $\{\frac{1}{4}N, \frac{1}{2}N, \frac{3}{4}N\}$. Since the $H(k, 2)$ -to- $H(k, 1)$ space is set at $\frac{1}{2}N$, the $H(k, l)$ -to- $H(k, 1)$ spaces ($3 \leq l \leq 4$) must constitute a set $\{\frac{1}{4}N, \frac{3}{4}N\}$. We propose to set the $H(k, 3)$ -to- $H(k, 1)$ space at $\frac{1}{4}N$, and the $H(k, 4)$ -to- $H(k, 1)$ space at $\frac{3}{4}N$. In general, for any m , the $H(k, l)$ -to- $H(k, 1)$ spaces ($2 \leq l \leq 2^m$) must constitute a set $\{\frac{i}{2^m}N \mid 1 \leq i \leq 2^m - 1\}$. It follows that the $H(k, l)$ -to- $H(k, 1)$ spaces ($2^{m-1} + 1 \leq l \leq 2^m$) must constitute a set $\{\frac{2^i - 1}{2^m}N \mid 1 \leq i \leq 2^{m-1}\}$. For each $1 \leq i \leq 2^{m-1}$, we propose to set the $H(k, 2^{m-1} + i)$ -to- $H(k, 1)$ space at $\frac{2^i - 1}{2^m}N$. Therefore, given $H(k, 1)$, we have

$$H(k, l) = \left(H(k, 1) + \frac{2l - 2^{\lceil \log_2 l \rceil} - 1}{2^{\lceil \log_2 l \rceil}} N \right) \bmod N. \quad (3)$$

It can be inferred that a hash function satisfying (3) has the following property: for any prefix of the list $H(k, 1), H(k, 2), \dots, H(k, M)$, the spaces between neighboring slots in the cycle differ by at most a factor of 2. For example, a prefix of five slots have the spaces $0, \frac{1}{2}N, \frac{1}{4}N, \frac{3}{4}N$, and $\frac{1}{8}N$ with respect to $H(k, 1)$. Thus, as shown in Fig. 3, the spaces between neighboring slots in the cycle are $\frac{1}{8}N, \frac{1}{8}N, \frac{1}{4}N, \frac{1}{4}N$, and $\frac{1}{4}N$. Our experimental results, not reported in this paper due to space limitations, show that hash functions satisfying (3) lead to much lower access latency compared to randomly chosen two-argument hash functions.

Note that the techniques proposed in this section and Section 3.2 are orthogonal. To construct a hole-free hash function that satisfies (3), we can first pick an arbitrary one-argument hash function $H(k, 1)$, then extend it to a two-argument function according to (3), and finally inject an offset to make it hole-free based on Theorem 1.

3.4 Bandwidth Allocation

So far, we have assumed the number of times each item should be broadcast in a cycle is given. In this section, we discuss the bandwidth allocation problem in MHash indexing. Since the objective of nonflat data broadcast is to reduce access latency, we consider average access latency as the performance metric in bandwidth allocation. Given

n items d_1, d_2, \dots, d_n , let p_i be the access probability⁴ of d_i , where $\sum_{i=1}^n p_i = 1$. Without loss of generality, assume that $p_1 \leq p_2 \leq \dots \leq p_n$. Let r_i be the fraction of bandwidth allocated to d_i , where $\sum_{i=1}^n r_i = 1$. If the broadcast instances of each item are equally spaced, the space between neighboring broadcast instances of d_i is proportional to $\frac{1}{r_i}$ and, thus, the average access latency of d_i is proportional to $\frac{1}{2r_i}$. Therefore, the overall access latency is proportional to

$$\sum_{i=1}^n p_i \cdot \left(\frac{1}{2r_i} \right) = \frac{1}{2} \sum_{i=1}^n \frac{p_i}{r_i}.$$

It has been proven that the latency is minimized when $r_i \propto \sqrt{p_i}$, i.e.,

$$r_i = \frac{\sqrt{p_i}}{\sum_{j=1}^n \sqrt{p_j}}$$

for each item d_i [19]. However, this solution is not directly applicable to the MHash bandwidth allocation problem due to the following constraint. Note that, with MHash indexing, each item is broadcast at least once and at most M times per cycle, where M is the replication bound. Thus, the bandwidth fractions allocated to the items can differ by, at most, a factor of M . The bandwidth allocation problem in MHash indexing is formally defined as follows:

Definition 1. (MHash Bandwidth Allocation Problem).

Given the access probabilities $p_1 \leq p_2 \leq \dots \leq p_n$ of data items d_1, d_2, \dots, d_n , respectively, and the replication bound M , the objective of the MHash bandwidth allocation problem is to find an allocation $R = (r_1, r_2, \dots, r_n)$, where $0 < r_i < 1$, $\sum_{i=1}^n r_i = 1$ and $\forall i, j, \frac{r_i}{r_j} \leq M$, such that $T = \frac{1}{2} \sum_{i=1}^n \frac{p_i}{r_i}$ is minimized. Here, $\forall i, j, \frac{r_i}{r_j} \leq M$ is called the differentiation constraint.

If $\frac{p_n}{p_1} \leq M^2$, the settings of

$$r_i = \frac{\sqrt{p_i}}{\sum_{j=1}^n \sqrt{p_j}} \quad (i = 1, 2, \dots, n)$$

satisfy the differentiation constraint because $\frac{r_i}{r_j} \leq M$ for any i and j . Since these settings are the optimal bandwidth allocation in the absence of the differentiation constraint [19], they are also the optimal solution to the MHash bandwidth allocation problem. If $\frac{p_n}{p_1} > M^2$, the optimal bandwidth allocation in MHash indexing is less obvious. We start by showing that more frequently accessed items must be allocated higher bandwidth in the optimal allocation.

Theorem 2. The optimal solution $R = (r_1, r_2, \dots, r_n)$ to the MHash bandwidth allocation problem satisfies

$$r_1 \leq r_2 \leq r_3 \leq \dots \leq r_n.$$

Proof. See Appendix B for details. \square

Theorem 3 shows that if $\frac{p_n}{p_1} > M^2$, $\frac{r_n}{r_1}$ must be M in the optimal bandwidth allocation.

4. The server can estimate the access probabilities by a number of methods [15], which are beyond the scope of this paper.

Theorem 3. If $\frac{p_n}{p_1} > M^2$, the optimal solution $R = (r_1, r_2, \dots, r_n)$ to the MHash bandwidth allocation problem satisfies $\frac{r_n}{r_1} = M$.

Proof. See Appendix C for details. \square

Theorems 2 and 3 imply that if $\frac{p_n}{p_1} > M^2$, the optimal bandwidth allocation must take one of the following two forms:

$$(A) \ r_1 = r_2 = \dots = r_{i-1} < r_i = r_{i+1} = \dots = r_n$$

for some $1 < i \leq n$ (we shall call i the *single separator*), where $\frac{r_n}{r_1} = M$, or

$$(B) \ r_1 = \dots = r_{i-1} < r_i \leq r_{i+1} \leq \dots \leq r_{j-1} \leq r_j < r_{j+1} = \dots = r_n$$

for some $1 < i \leq j < n$ (we shall call i and j the *lower and upper separators*, respectively), where $\frac{r_n}{r_1} = M$.

Given the single separator i , the allocation in form \mathcal{A} is given by

$$r_1 = r_2 = \dots = r_{i-1} = \frac{1}{i-1 + M(n-i+1)},$$

and

$$r_i = r_{i+1} = \dots = r_n = \frac{M}{i-1 + M(n-i+1)}.$$

If we denote the associated T value by $T_{\mathcal{A}}(i)$, the best allocation in form \mathcal{A} is then given by $\min_{1 < i \leq n} T_{\mathcal{A}}(i)$.

For the allocations in form \mathcal{B} , Theorem 4 presents some properties of the lower and upper separators.

Theorem 4. If $\frac{p_n}{p_1} > M^2$ and the optimal solution $R = (r_1, r_2, \dots, r_n)$ to the MHash bandwidth allocation problem has form \mathcal{B} , the lower and upper separators i and j satisfy 1) $\frac{p_i}{p_1} < M^2$; and 2) $\frac{p_{j+1}}{p_{i-1}} \geq M^2$.

Proof. See Appendix D for details. \square

Given the lower and upper separators i and j , let $r_i + r_{i+1} + \dots + r_j = X$. Based on the Lagrange multiplier theorem, the access latency is minimized when

$$r_k = \frac{\sqrt{p_k}}{\sum_{m=i}^j \sqrt{p_m}} \cdot X$$

for each $i \leq k \leq j$. Note that these settings satisfy the differentiation constraint because Theorem 4 shows that $\frac{p_i}{p_1} < M^2$. The remaining fractions in the allocation are given by

$$r_1 = r_2 = \dots = r_{i-1} = \frac{1-X}{i-1 + M(n-j)},$$

and

$$r_{j+1} = r_{j+2} = \dots = r_n = \frac{M(1-X)}{i-1 + M(n-j)}.$$

Therefore,

$$T = \frac{1}{2} \left(\frac{(\sum_{m=i}^j \sqrt{p_m})^2}{X} + \frac{(M \cdot \sum_{m=1}^{i-1} p_m + \sum_{m=j+1}^n p_m) \cdot (i-1 + M(n-j))}{M(1-X)} \right).$$

TABLE 1
System Parameters

Parameter	Description	Default Value
n	number of data items	5000
M	replication bound	8
θ	Zipf parameter of access distribution	1.0
S_b	size of a bucket	1KB
S_d	size of a data item	128 bytes

Since $r_{i-1} < r_i$ and $r_j < r_{j+1}$, we have

$$\frac{1-X}{i-1 + M(n-j)} < \frac{\sqrt{p_i}}{\sum_{m=i}^j \sqrt{p_m}} \cdot X$$

and

$$\frac{\sqrt{p_j}}{\sum_{m=i}^j \sqrt{p_m}} \cdot X < \frac{M(1-X)}{i-1 + M(n-j)}.$$

It follows that

$$\frac{1}{\frac{((i-1)+M(n-j))\sqrt{p_i}}{\sum_{m=i}^j \sqrt{p_m}} + 1}} < X < \frac{M}{\frac{((i-1)+M(n-j))\sqrt{p_j}}{\sum_{m=i}^j \sqrt{p_m}} + M}}. \quad (4)$$

Computing the minimum value of T in the range of X specified by (4) is straightforward. If we denote the minimum value by $T_{\mathcal{B}}(i, j)$, the best allocation in form \mathcal{B} is then given by

$$\min_{\substack{p_i < M^2 \\ \frac{p_{j+1}}{p_{i-1}} \geq M^2}} T_{\mathcal{B}}(i, j).$$

Therefore, if $\frac{p_n}{p_1} > M^2$, the optimal MHash bandwidth allocation produces the minimum T value of

$$\min \left(\min_{1 < i \leq n} T_{\mathcal{A}}(i), \min_{\substack{p_i < M^2 \\ \frac{p_{j+1}}{p_{i-1}} \geq M^2}} T_{\mathcal{B}}(i, j) \right).$$

Assume that the least frequently accessed item is broadcast once per cycle. Having obtained the optimal allocation (r_1, r_2, \dots, r_n) , the number of times each item d_i should be broadcast in a cycle is given by

$$c_i = \begin{cases} 1 & i = 1, \\ \lceil \frac{r_i}{r_1} - \frac{1}{2} \rceil & 2 \leq i \leq n. \end{cases}$$

4 PERFORMANCE EVALUATION

4.1 Experimental Setup

We have conducted simulation experiments to compare MHash indexing with a wide range of existing schemes. Table 1 summarizes the system parameters and their settings in the experiments. We simulate a set of n data items whose access probabilities are assumed to follow a Zipf-like distribution. Specifically, the access probability p_i of the i th most popular item follows:

$$p_i \propto \frac{(1/i)^\theta}{\sum_{i=1}^n (1/i)^\theta},$$

where $\theta > 0$ is the Zipf parameter [29]. It is obvious that the higher the value of θ , the more skewed the access distribution. A θ value of 0 degenerates the Zipf-like distribution to a uniform distribution. The default Zipf parameter was set at 1.0. The default replication bound M was set at 8. The default bucket size and item size were set at 1 KB and 128 bytes, respectively.

We refer to the number of slots that can be accommodated in a bucket as the *bucket capacity* P . It is given by $P = \lfloor \frac{S_b - S_{bh}}{S_d + S_{sh}} \rfloor$, where S_b and S_d are the bucket size and item size, respectively, and S_{bh} and S_{sh} are the per-bucket and per-slot indexing overhead, respectively. For MHash indexing, as mentioned in Section 3.1, the following information is recorded in the header of each bucket: the hash function H , the cycle length N , and the replication bound M . We assume the hash function is characterized by two constants (e.g., see $\mathcal{H}_1(k, 1)$ below) and an offset (Theorem 1). The above information together with the bucket number and the bucket capacity⁵ results in an S_{bh} of 28 bytes. In addition to the broadcast item, the distance pointer is also broadcast in each slot. Therefore, S_{sh} was set at 4 bytes.

The keys of data items are assumed to be integers and were randomly generated in our experiments. The key values are uniformly distributed in the range of $[0, 2^{32} - 1]$. We used the following hash functions:

$$\mathcal{H}_1(k, 1) = (A \cdot ((A + B) \oplus k) + B) \bmod 2^{31},$$

where k is the key value, $A = 1103515245$, $B = 12345$, and \oplus is a bitwise exclusive-or operation [18], and a universal hash function

$$\mathcal{H}_2(k, 1) = (x_1 m_1 \oplus x_2 m_2 \oplus \dots \oplus x_{32} m_{32}),$$

where $x_1 x_2 \dots x_{32}$ is the binary representation of key value, and m_i s are 0-1 bit vectors of length 32 [6]. These hash functions were extended based on the techniques described in Sections 3.2 and 3.3 for MHash indexing. In addition, we also tested MHash with a synthetic collision-free hash table in which no pair of items are hashed to the same slot. On constructing the broadcast schedule, the expected access latency and tuning time of each data item were first calculated by taking an average over all possible locations of the initial probe. The overall access latency and tuning time were then computed by taking a weighted average over all data items based on their access probabilities. Both metrics are measured in the unit of bucket, which is the smallest accessible unit of broadcast. We have tested a wide range of parameter settings. For each setting, we randomly generated 100 sets of key values. Each set of keys were randomly ordered into a list, where the i th key on the list was assumed to be the i th most popular item. The average performance of these 100 simulation runs is plotted for performance comparison.

In addition to MHash, the following existing schemes described in Section 2 were included in the experiments for comparison purposes: latency-optimal broadcast (abbreviated as LatOpt) [19], one-argument hash index

5. The bucket number and bucket capacity are included for the purpose of deriving slot numbers.

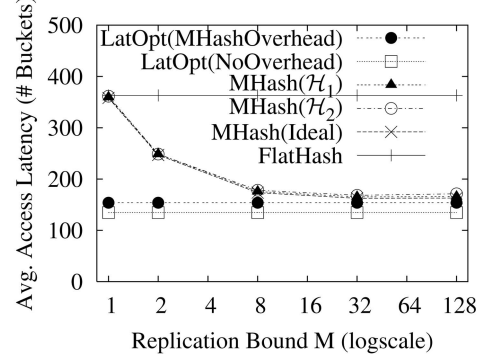


Fig. 4. Average access latency versus replication bound.

(FlatHash) [9], distributed tree index (DistTree) [10], exponential index (Exponential) [22], [23], unbalanced tree index (UnbalanceTree) [7], [16], and hybrid index (Hybrid) [8]. Interested readers are referred to Section 2 and the references for details of these schemes. The latency-optimal broadcast differentiates the broadcast frequencies of data items based on the square-root rule of bandwidth allocation (i.e., $r_i \propto \sqrt{p_i}$) [19]. The broadcast instances of each item are assumed to be equally spaced in the broadcast schedule. Thus, LatOpt is used as a yardstick (lower bound) on access latency. However, LatOpt does not include any index in the broadcast. This leads to a tuning time equal to the access latency. The remaining five schemes all build index on broadcast data. All these schemes, except the Hybrid index, construct a flat schedule where each item is broadcast exactly once in a cycle regardless of the access distribution. The replicated levels of index nodes in DistTree and the index base and chunk size in Exponential are tuned to optimize access latency. The Hybrid scheme uses three disks with relative broadcast frequencies of 3, 2, and 1 in broadcast disk scheduling. In all tree-based indexes, each key value and offset in the index tables is assumed to take up 4 bytes.

4.2 Comparison with LatOpt and FlatHash

In this section, we investigate the performance of MHash under different replication bounds and hash functions. We tested MHash with three different hash functions: \mathcal{H}_1 , \mathcal{H}_2 , and a synthetic collision-free hash table. Their results are labeled MHash(\mathcal{H}_1), MHash(\mathcal{H}_2), and MHash(Ideal), respectively. We also compare MHash against LatOpt and FlatHash.

Fig. 4 shows the access latency as a function of replication bound M . Note that FlatHash and LatOpt do not rely on M , so their performance is independent of M . When $M = 1$, MHash broadcasts all items exactly once in each cycle. So, its access latency is similar to that of FlatHash which also employs flat broadcast. When M increases, MHash becomes more flexible in bandwidth allocation. Therefore, the access latency of MHash decreases rapidly with increasing M . As shown in Fig. 4, an M value of 2 improves the access latency by 30 percent compared to that of $M = 1$. Comparing with FlatHash, MHash's access latency is 50 percent lower when $M = 8$.

We conducted two sets of experiments for LatOpt. In the first set, the entire bucket was fully used to accommodate

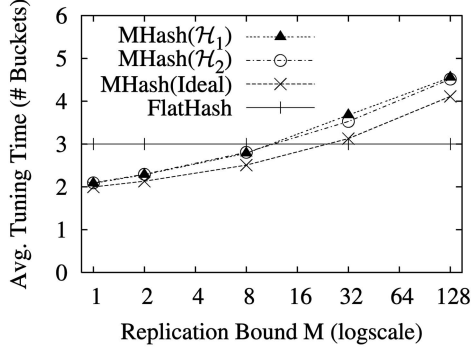


Fig. 5. Average tuning time versus replication bound.

data items, i.e., bucket capacity $P = \lfloor \frac{S_b}{S_d} \rfloor$. This represents the lower bound access latency without any indexing overhead. The second set of experiments assumed a bucket capacity equal to that of MHash. This represents the lower bound access latency in the presence of MHash’s indexing overhead. The results of these two sets of experiments are labeled LatOpt(NoOverhead) and LatOpt(MHashOverhead), respectively. As shown in Fig. 4, MHash approaches LatOpt(MHashOverhead) in access latency when M grows beyond 8. This implies our spacing and bandwidth allocation techniques presented in Sections 3.3 and 3.4 succeed in optimizing the access latency by means of nonflat broadcast. We also note that the difference between LatOpt(MHashOverhead) and LatOpt(NoOverhead) is small in access latency. This shows MHash has little indexing overhead.

Fig. 5 shows the tuning time as a function of M . Comparing MHash with FlatHash, we note that when $M = 1$, MHash has a lower tuning time than FlatHash. This is because they use different methods to resolve collisions. By using chaining, MHash guarantees that for each slot, one of the items hashed to it is broadcast in the slot. On the other hand, FlatHash uses linear probing. If a collision pushes an item forward to a slot, the slot would not be used to accommodate any item hashed to it. This has the effect of increasing the tuning time to all the items hashed to the slot by 1.

In general, the tuning time of MHash increases with M . This is because a larger replication bound increases the number of cheating slots. Due to pruning in data accesses, MHash’s tuning time increases almost logarithmically⁶ with M (i.e., very slowly). As shown in Fig. 5, the tuning time of MHash is lower than that of FlatHash up to an M value of 8 and is only 1.5 buckets higher than that of FlatHash when M rises to 128. On the other hand, since LatOpt(MHashOverhead) and LatOpt(NoOverhead) do not use air indexing, their tuning times are identical to the access latencies which are not shown in Fig. 5 due to high values.

The total energy cost of data accesses is the sum of that consumed in the active and doze modes. While tuning time measures the energy consumed in the active mode, access latency largely reflects the energy consumed in the doze mode. In the absence of concurrent accesses to multiple

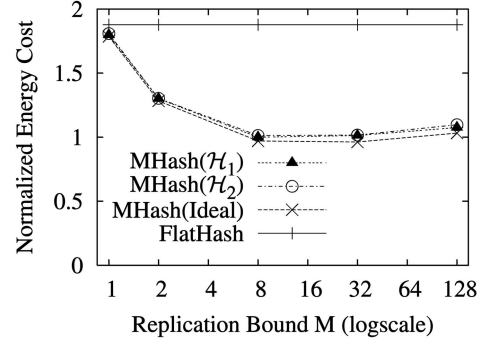


Fig. 6. Normalized energy cost versus replication bound.

items by a client, the total energy cost of a data access can be approximated by

$$E = (\text{access_latency} - \text{tuning_time}) \cdot r_{\text{doze}} + \text{tuning_time} \cdot r_{\text{active}},$$

where r_{doze} and r_{active} are the energy consumption rates of the doze and active modes, respectively. Fig. 6 plots the energy cost normalized by that of MHash(\mathcal{H}_1) at $M = 8$ when r_{doze} and r_{active} are, respectively, set at 60 mW and 950 mW, representing a typical wireless PC card ORiNOCO [20]. The energy costs of LatOpt(MHashOverhead) and LatOpt(NoOverhead) are too high to be shown in the figure. It is clearly seen that MHash significantly improves energy efficiency. When $M = 8$, MHash outperforms FlatHash by more than 45 percent and LatOpt(MHashOverhead) by more than 90 percent (not shown in Fig. 6) in energy cost.

Comparing different hash functions, we note that the performance of MHash improves with decreasing collision rate. As shown in Fig. 5, MHash(Ideal) outperforms MHash(\mathcal{H}_1) and MHash(\mathcal{H}_2) in tuning time. However, the improvement is not significant. This implies the hash functions \mathcal{H}_1 and \mathcal{H}_2 have low collision rates. On the other hand, it is seen from Fig. 4 that the three hash functions result in similar access latencies. Thus, we shall report only the results of MHash(\mathcal{H}_1) in the rest of this paper. Moreover, the access latency and energy cost of MHash indexing remains similar when M grows beyond 8. Therefore, the default value of M was set at 8 in the remaining experiments.

4.3 Comparison with DistTree, Exponential, UnbalanceTree, and Hybrid

In this section, we study the performance of MHash under different access distributions and compare it against DistTree, Exponential, UnbalanceTree and Hybrid indexes. Figs. 7 and 8 show the performance results for various Zipf parameters.

DistTree and Exponential both construct flat schedules regardless of the access distribution. Each item is broadcast exactly once in a cycle and in the order of key value. So, their access latencies remain similar over a wide range of access distribution (see Fig. 7). With multiple index trees that share links, Exponential allows searching to start at anywhere in the broadcast. Thus, its access latency is lower than that of DistTree. In contrast, MHash uses nonflat broadcast. Its access latency is similar to that of Exponential under uniform access distribution and reduces substantially when θ increases. When $\theta = 1.5$, MHash’s access latency is

6. Note that the axis of M is in log scale.

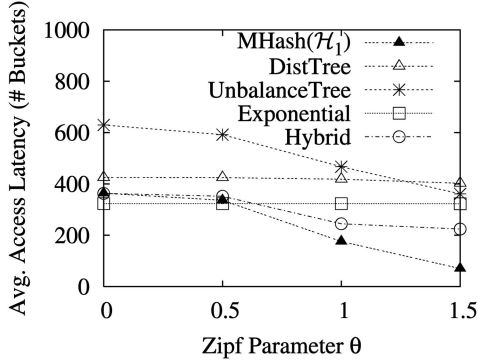


Fig. 7. Average access latency versus Zipf parameter.

more than 80 percent and 75 percent lower than those of DistTree and Exponential, respectively. Similar performance trends are observed for tuning time. Fig. 8 shows that the tuning time of MHash decreases with increasing access skewness. This demonstrates MHash’s nature that popular items have less tuning time than unpopular items (see Section 3.1). When the skewness in access distribution increases, popular items take up a larger portion of all requests, thereby decreasing the average tuning time. MHash’s tuning time is much lower than those of DistTree and Exponential at θ values higher than 1.

In UnbalanceTree, more frequently accessed items are placed closer to the root of the index tree. Therefore, its tuning time decreases with increasing access skewness. However, like other tree-based indexes, the searching must start from the root index node in UnbalanceTree. Normally, the client needs to tune to the broadcast channel at least twice (an initial probe and the root index node) before it accesses the data item. As a result, UnbalanceTree’s tuning time cannot be reduced beyond 3. In contrast, hash-based indexes allow searching to start from anywhere. As shown in Fig. 8, MHash reduces the tuning time by more than 18 percent and 30 percent compared to UnbalanceTree at θ values of 1.0 and 1.5, respectively. When constructing the broadcast schedule, UnbalanceTree places popular items closer than unpopular items to the root index node to reduce access latency. Thus, UnbalanceTree’s access latency decreases with increasing access skewness. However, since UnbalanceTree uses flat broadcast, Fig. 7 shows that its access latency is substantially

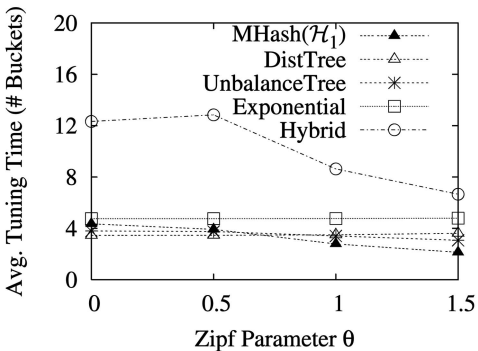


Fig. 8. Average tuning time versus Zipf parameter.

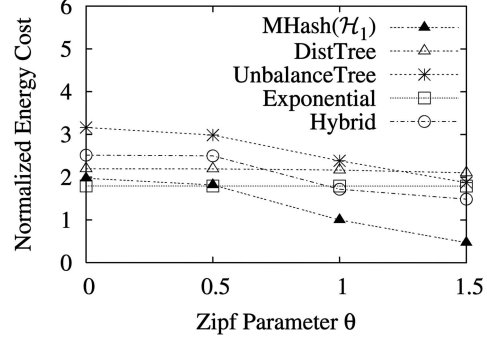


Fig. 9. Normalized energy cost versus Zipf parameter.

higher than that of MHash. Moreover, unlike DistTree, UnbalanceTree does not use replication to reduce the latency to access the root index node. Therefore, UnbalanceTree’s access latency is even higher than that of DistTree under uniform access distribution.

The Hybrid scheme incorporates broadcast disk scheduling [4], tree-based index [10], and signature-based index [12]. Broadcast disk scheduling differentiates the broadcast frequencies of data items based on their popularity. However, the differentiation is not as fine grained as that in MHash. Therefore, as seen from Fig. 7, the access latency of Hybrid reduces with increasing access skewness but is still much higher than MHash under skewed access distribution. Due to nonflat broadcast, Hybrid constructs an index only for short segments of broadcast, where each segment holds a sequence of items with increasing key values. So, the effectiveness of indexing diminishes. Moreover, since a signature does not provide the arrival times of data items, when a match is found, the buckets indexed by the signature have to be searched sequentially. As a result, Hybrid has considerably higher tuning time than the other schemes (see Fig. 8).

Fig. 9 summarizes the total energy cost for different schemes. It shows that the energy cost of MHash decreases with increasing access skewness and is much lower than that of other schemes under skewed access distribution. MHash saves more than 40 percent and 60 percent of the energy compared to the other schemes at θ values of 1.0 and 1.5, respectively.

5 CONCLUSIONS

We have presented an MHash indexing scheme that optimizes access latency and tuning time in an integrated fashion for wireless data broadcast. MHash reduces tuning time by mapping data items to the slots in the broadcast schedule via a hash function. We have shown that a hole-free hash function for the purpose of broadcast scheduling can be constructed by injecting an offset into an arbitrary hash function. Meanwhile, the two-argument nature of the hash function allows each data item to be broadcast for an adjustable number of times in a cycle. Popular items are broadcast more frequently than unpopular ones, thereby enabling nonflat data broadcast to reduce access latency.

We have derived an optimal bandwidth allocation for MHash indexing. Experimental results show that under skewed access distribution, MHash outperforms state-of-the-art air indexing schemes and achieves access latency close to optimal broadcast scheduling.

APPENDIX A

Proof of Theorem 1. We first show that $h_{N-1}(H) = h_{N-2}(H)$.

This is because if $f_{N-1}(H) > 0$, it follows from (2) in Section 3.2 that $h_{N-1}(H) = h_{N-2}(H)$. Otherwise, if $f_{N-1}(H) = 0$, since $\sum_{j=0}^{N-1} f_j(H) = N$, we have

$$\sum_{j=0}^{N-2} f_j(H) = \sum_{j=0}^{N-1} f_j(H) = N > N-1 \geq N-1 - h_{N-2}(H).$$

Therefore, based on (2), $h_{N-1}(H) = h_{N-2}(H)$. This implies the last slot in the cycle cannot be a hole, i.e., b exists and $0 \leq b \leq N-1$.

If all slots from 0 to $N-1$ are hole-free under $H(k, l)$, then $b=0$ and the conclusion is trivial. Otherwise, if $1 \leq b \leq N-1$, two properties follow from the definition of b : 1) slot $(b-1)$ is a hole, i.e., $h_{b-1}(H) = h_{b-2}(H) + 1$ and, 2) for each $b \leq i \leq N-1$, $h_i(H) = h_{i-1}(H)$.

Next, we show that the hash function

$$H'(k, l) = (H(k, l) - b) \bmod N$$

is hole-free, i.e., for each $0 \leq i \leq N-1$, $h_i(H') = 0$. Note that the definition of $H'(k, l)$ indicates

$$f_i(H') = f_{(b+i) \bmod N}(H).$$

Since $h_{b-1}(H) = h_{b-2}(H) + 1$, based on (2), we have $f_{b-1}(H) = 0$ and $\sum_{j=0}^{b-2} f_j(H) = b-1 - h_{b-2}(H)$. Thus,

$$\sum_{j=0}^{b-1} f_j(H) = \sum_{j=0}^{b-2} f_j(H) = b-1 - h_{b-2}(H) = b - h_{b-1}(H). \quad (5)$$

According to Property 2, $h_b(H) = h_{b-1}(H)$. Combining (2) and (5), we must have $f_b(H) > 0$. Therefore, $f_0(H') = f_b(H) > 0$. It follows from (2) that $h_0(H') = 0$.

To prove that, for each $1 \leq i \leq N-1$, $h_i(H') = 0$, we consider three cases separately.

Case 1. $1 \leq i \leq N-b-1$.

Prove by induction. Suppose $h_{i-1}(H') = h_{i-2}(H') = \dots = h_0(H') = 0$ for some $1 \leq i \leq N-b-1$, we show that $h_i(H') = 0$.

Since $1 \leq i \leq N-b-1$, we have $f_i(H') = f_{b+i}(H)$. According to Property 2, $h_{b+i}(H) = h_{b+i-1}(H)$. It follows from (2) that either $f_{b+i}(H) > 0$ or

$$\sum_{j=0}^{b+i-1} f_j(H) > b+i - h_{b+i-1}(H).$$

If $f_{b+i}(H) > 0$, we have $f_i(H') = f_{b+i}(H) > 0$ and, hence, based on (2), $h_i(H') = h_{i-1}(H')$. Otherwise, if

$$\sum_{j=0}^{b+i-1} f_j(H) > b+i - h_{b+i-1}(H),$$

based on (5),

$$\begin{aligned} \sum_{j=0}^{i-1} f_j(H') &= \sum_{j=0}^{i-1} f_{b+j}(H) \\ &= \sum_{j=b}^{b+i-1} f_j(H) \\ &= \sum_{j=0}^{b+i-1} f_j(H) - \sum_{j=0}^{b-1} f_j(H) \\ &= \sum_{j=0}^{b+i-1} f_j(H) - (b - h_{b-1}(H)) \\ &> b+i - h_{b+i-1}(H) - (b - h_{b-1}(H)) \\ &= i - h_{b+i-1}(H) + h_{b-1}(H). \end{aligned}$$

Note that Property 2 implies

$$h_{b+i-1}(H) = h_{b+i-2}(H) = \dots = h_{b-1}(H).$$

Therefore,

$$\sum_{j=0}^{i-1} f_j(H') > i = i - h_{i-1}(H').$$

It follows from (2) that $h_i(H') = h_{i-1}(H') = 0$.

Case 2. $i = N-b$.

Based on (5) and $\sum_{j=0}^{N-1} f_j(H) = N$,

$$\begin{aligned} \sum_{j=0}^{N-b-1} f_j(H') &= \sum_{j=0}^{N-b-1} f_{b+j}(H) \\ &= \sum_{j=b}^{N-1} f_j(H) \\ &= \sum_{j=0}^{N-1} f_j(H) - \sum_{j=0}^{b-1} f_j(H) \\ &= N - (b - h_{b-1}(H)). \end{aligned}$$

According to Property 1,

$$h_{b-1}(H) = h_{b-2}(H) + 1 > 0.$$

Also, it has been proven in Case 1 that $h_{N-b-1}(H') = 0$.

Therefore,

$$\begin{aligned} \sum_{j=0}^{N-b-1} f_j(H') &= N - b + h_{b-1}(H) \\ &> N - b \\ &= N - b - h_{N-b-1}(H'). \end{aligned}$$

It follows from (2) that

$$h_{N-b}(H') = h_{N-b-1}(H') = 0.$$

Case 3. $N-b+1 \leq i \leq N-1$.

Again, prove by induction. Note that it has been proven in Case 2 that $h_{N-b}(H') = 0$. Suppose

$$h_{i-1}(H') = h_{i-2}(H') = \dots = h_{N-b}(H') = 0$$

for some $N-b+1 \leq i \leq N-1$, we show that $h_i(H') = 0$.

For each $N - b + 1 \leq j \leq N - 1$, $f_j(H') = f_{j+b-N}(H)$. Based on (5) and $\sum_{j=0}^{N-1} f_j(H) = N$, we have

$$\begin{aligned} \sum_{j=0}^{i-1} f_j(H') &= \sum_{j=0}^{N-b-1} f_j(H') + \sum_{j=N-b}^{i-1} f_j(H') \\ &= \sum_{j=b}^{N-1} f_j(H) + \sum_{j=0}^{i+b-N-1} f_j(H) \\ &= \sum_{j=0}^{N-1} f_j(H) - \sum_{j=0}^{b-1} f_j(H) + \sum_{j=0}^{i+b-N-1} f_j(H) \\ &= N - (b - h_{b-1}(H)) + \sum_{j=0}^{i+b-N-1} f_j(H). \end{aligned}$$

Recall from Section 3.2 that for each $1 \leq m \leq N - 1$,

$$\sum_{j=0}^{m-1} f_j(H) \geq m - h_{m-1}(H).$$

Thus,

$$\sum_{j=0}^{i+b-N-1} f_j(H) \geq (i + b - N) - h_{i+b-N-1}(H).$$

On the other hand, it follows from Property 1 that $h_{b-1}(H) > h_m(H)$ for any $0 \leq m \leq b - 2$. Since

$$0 \leq i + b - N - 1 \leq b - 2,$$

we have

$$h_{b-1}(H) > h_{i+b-N-1}(H).$$

Therefore,

$$\begin{aligned} \sum_{j=0}^{i-1} f_j(H') &\geq N - b + h_{b-1}(H) + (i + b - N) - h_{i+b-N-1}(H) \\ &= i + h_{b-1}(H) - h_{i+b-N-1}(H) \\ &> i \\ &= i - h_{i-1}(H'). \end{aligned}$$

Based on (2), $h_i(H') = h_{i-1}(H') = 0$.

Therefore, for each $0 \leq i \leq N - 1$, $h_i(H') = 0$. Hence, the theorem is proven. \square

APPENDIX B

Proof of Theorem 2. Assume on the contrary that there exist $1 \leq i < j \leq n$ such that $r_i > r_j$ in $R = (r_1, r_2, \dots, r_n)$. We consider two cases, $p_i < p_j$ and $p_i = p_j$, separately.

If $p_i < p_j$, we construct a new allocation R' by swapping the bandwidth fractions allocated to d_i and d_j in R , i.e.,

$$R' = (r_1, \dots, r_{i-1}, r_j, r_{i+1}, \dots, r_{j-1}, r_i, r_{j+1}, \dots, r_n).$$

The access latency of R' is given by

$$T' = \frac{1}{2} \left(\frac{p_i}{r_j} + \frac{p_j}{r_i} + \sum_{k=1, k \neq i, k \neq j}^{n-1} \frac{p_k}{r_k} \right).$$

Note that the access latency of $R = (r_1, \dots, r_n)$ is

$$T = \frac{1}{2} \sum_{k=1}^n \frac{p_k}{r_k}.$$

Since

$$\begin{aligned} T - T' &= \frac{1}{2} \left(\left(\frac{p_i}{r_i} + \frac{p_j}{r_j} \right) - \left(\frac{p_i}{r_j} + \frac{p_j}{r_i} \right) \right) \\ &= \frac{(r_j - r_i)(p_i - p_j)}{2r_i r_j} \\ &> 0, \end{aligned}$$

it follows that $T > T'$, which contradicts with the optimality of R .

If $p_i = p_j$, we construct a new allocation R'' by equalizing the bandwidth fractions allocated to d_i and d_j in R , i.e.,

$$R'' = (r_1, \dots, r_{i-1}, \frac{1}{2}(r_i + r_j), r_{i+1}, \dots, r_{j-1}, \frac{1}{2}(r_i + r_j), r_{j+1}, \dots, r_n).$$

The access latency of R'' is given by

$$T'' = \frac{1}{2} \left(\frac{p_i}{\frac{1}{2}(r_i + r_j)} + \frac{p_j}{\frac{1}{2}(r_i + r_j)} + \sum_{k=1, k \neq i, k \neq j}^n \frac{p_k}{r_k} \right).$$

Since

$$\begin{aligned} T - T'' &= \frac{1}{2} \left(\left(\frac{p_i}{r_i} + \frac{p_j}{r_j} \right) - \left(\frac{p_i}{\frac{1}{2}(r_i + r_j)} + \frac{p_j}{\frac{1}{2}(r_i + r_j)} \right) \right) \\ &= \frac{(r_j - r_i)(p_i r_j - p_j r_i)}{2r_i r_j (r_i + r_j)} \\ &= \frac{p_i (r_j - r_i)^2}{2r_i r_j (r_i + r_j)} \\ &> 0, \end{aligned}$$

it follows that $T > T''$, which also contradicts with the optimality of R .

Hence, the theorem is proven. \square

APPENDIX C

Proof of Theorem 3. Assume on the contrary that $\frac{r_n}{r_1} < M$.

We construct a new allocation R' by reallocating the bandwidth between d_1 and d_n such that their bandwidth fractions differ by a factor of M , i.e.,

$$R' = (r'_1, r_2, r_3, \dots, r_{n-1}, r'_n),$$

where $r'_1 = \frac{r_1 + r_n}{M+1}$, and $r'_n = \frac{M(r_1 + r_n)}{M+1}$. The access latency of R' is given by

$$T' = \frac{1}{2} \left(\frac{p_1}{r'_1} + \frac{p_n}{r'_n} + \sum_{k=2}^{n-1} \frac{p_k}{r_k} \right).$$

The access latency of $R = (r_1, r_2, \dots, r_n)$ is

$$T = \frac{1}{2} \sum_{k=1}^n \frac{p_k}{r_k}.$$

Thus,

$$\begin{aligned} T - T' &= \frac{1}{2} \left(\left(\frac{p_1}{r_1} + \frac{p_n}{r_n} \right) - \left(\frac{p_1}{r'_1} + \frac{p_n}{r'_n} \right) \right) \\ &= \frac{1}{2} \left(\left(\frac{p_1}{r_1} + \frac{p_n}{r_n} \right) - \left(\frac{p_1}{\frac{r_1+r_n}{M+1}} + \frac{p_n}{\frac{M(r_1+r_n)}{M+1}} \right) \right) \\ &= \frac{(r_n - Mr_1)(Mp_1r_n - p_nr_1)}{2Mr_1r_n(r_1 + r_n)}. \end{aligned}$$

Since $\frac{p_n}{r_1} > M^2$ and $\frac{r_n}{r_1} < M$, we have $r_n - Mr_1 < 0$ and $Mp_1r_n - p_nr_1 < 0$. Therefore, $T - T' > 0$, which contradicts with the optimality of R .

Hence, the theorem is proven. \square

APPENDIX D

Proof of Theorem 4. We first prove Claim 1 by contradiction. Assume, on the contrary, that $\frac{p_i}{r_i} \geq M^2$. We construct a new allocation R' by moving some bandwidth fraction from d_i to d_j under the constraint that the relative order of bandwidth fractions does not change, i.e.,

$$R' = (r_1, \dots, r_{i-1}, r'_i, r_{i+1}, \dots, r_{j-1}, r'_j, r_{j+1}, \dots, r_n),$$

whereas $r'_i = r_i - \Delta$, $r'_j = r_j + \Delta$, and

$$\Delta = \min(r_{j+1} - r_j, r_i - r_{i-1}) > 0.$$

It is easy to see that R' satisfies the differentiation constraint of the MHash bandwidth allocation problem. The access latency of R' is given by

$$T' = \frac{1}{2} \left(\frac{p_i}{r_i - \Delta} + \frac{p_j}{r_j + \Delta} + \sum_{k=1, k \neq i, k \neq j}^n \frac{p_k}{r_k} \right).$$

Note that the access latency of $R = (r_1, r_2, \dots, r_n)$ is

$$T = \frac{1}{2} \sum_{k=1}^n \frac{p_k}{r_k}.$$

Thus,

$$\begin{aligned} T - T' &= \frac{1}{2} \left(\left(\frac{p_i}{r_i} + \frac{p_j}{r_j} \right) - \left(\frac{p_i}{r_i - \Delta} + \frac{p_j}{r_j + \Delta} \right) \right) \\ &= \frac{1}{2} \left(\frac{p_j \Delta}{(r_j + \Delta)r_j} - \frac{p_i \Delta}{(r_i - \Delta)r_i} \right) \\ &= \frac{\Delta(p_j r_i (r_i - \Delta) - p_i r_j (r_j + \Delta))}{2r_i r_j (r_i - \Delta)(r_j + \Delta)}. \end{aligned}$$

Since $r_i - \Delta \geq r_{i-1}$ and $r_j + \Delta \leq r_{j+1}$, we have

$$\frac{r_j + \Delta}{r_i - \Delta} \leq \frac{r_{j+1}}{r_{i-1}} = M.$$

Note that $\frac{r_j}{r_i} < M$ and $\frac{p_i}{r_i} \geq M^2$. Therefore,

$$\frac{r_j}{r_i} \cdot \frac{r_j + \Delta}{r_i - \Delta} < M^2 \leq \frac{p_j}{p_i},$$

and

$$p_j r_i (r_i - \Delta) > p_i r_j (r_j + \Delta).$$

Thus, it follows that $T > T'$ which contradicts with the optimality of R .

Next, we prove Claim 2 by contradiction. Assume, on the contrary, that $\frac{p_{j+1}}{r_{j+1}} < M^2$. We construct a new allocation R'' by reallocating the bandwidth between d_{i-1} and d_{j+1} proportionally to the square-root of their access probabilities, i.e.,

$$R'' = (r_1, \dots, r_{i-2}, r''_{i-1}, r_i, \dots, r_j, r''_{j+1}, r_{j+2}, \dots, r_n),$$

where

$$r''_{j+1} = (r_{i-1} + r_{j+1}) \cdot \frac{\sqrt{p_{j+1}}}{\sqrt{p_{j+1}} + \sqrt{p_{i-1}}}$$

and

$$r''_{i-1} = (r_{i-1} + r_{j+1}) \cdot \frac{\sqrt{p_{i-1}}}{\sqrt{p_{j+1}} + \sqrt{p_{i-1}}}.$$

It is easy to see that R'' satisfies the differentiation constraint of the MHash bandwidth allocation problem. The access latency of R'' is given by

$$T'' = \frac{1}{2} \left(\frac{p_{i-1}}{r''_{i-1}} + \frac{p_{j+1}}{r''_{j+1}} + \sum_{k=1, k \neq i-1, k \neq j+1}^n \frac{p_k}{r_k} \right).$$

Since $r_{j+1} = Mr_{i-1}$,

$$\begin{aligned} T - T'' &= \frac{1}{2} \cdot \left(\left(\frac{p_{i-1}}{r_{i-1}} + \frac{p_{j+1}}{r_{j+1}} \right) - \left(\frac{p_{i-1}}{r''_{i-1}} + \frac{p_{j+1}}{r''_{j+1}} \right) \right) \\ &= \frac{1}{2} \left(\frac{p_{i-1}}{r_{i-1}} + \frac{p_{j+1}}{r_{j+1}} - \frac{p_{i-1}(\sqrt{p_{j+1}} + \sqrt{p_{i-1}})}{(r_{i-1} + r_{j+1})\sqrt{p_{i-1}}} \right. \\ &\quad \left. - \frac{p_{j+1}(\sqrt{p_{j+1}} + \sqrt{p_{i-1}})}{(r_{i-1} + r_{j+1})\sqrt{p_{j+1}}} \right) \\ &= \frac{(M\sqrt{p_{i-1}} - \sqrt{p_{j+1}})^2}{2r_{i-1}(M+1)M} \\ &> 0. \end{aligned}$$

Therefore, $T > T''$ which contradicts with the optimality of R .

Hence, the theorem is proven. \square

REFERENCES

- [1] Hughes Network Systems, <http://www.direcway.com>, 2006.
- [2] MSN Direct Service, <http://www.msndirect.com>, 2006.
- [3] StarBand, <http://www.starband.com>, 2006.
- [4] 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.
- [5] D. Barbara, "Mobile Computing and Databases—A Survey," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 1, pp. 108-117, Jan./Feb. 1999.
- [6] J.L. Carter and M.N. Wegman, "Universal Classes of Hash Functions," *J. Computer and System Sciences*, vol. 18, no. 2, pp. 143-154, Apr. 1979.
- [7] M.-S. Chen, K.-L. Wu, and P.S. Yu, "Optimizing Index Allocation for Sequential Data Broadcasting in Wireless Mobile Computing," *IEEE Trans. Knowledge and Data Eng.*, vol. 15, no. 1, pp. 161-173, Jan./Feb. 2003.

- [8] Q. Hu, W.-C. Lee, and D.L. Lee, "A Hybrid Index Technique for Power Efficient Data Broadcast," *Distributed and Parallel Databases* vol. 9, no. 2, pp. 151-177, Mar. 2001.
- [9] 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.
- [10] T. Imielinski, S. Viswanathan, and B.R. Badrinath, "Data on Air: Organization and Access," *IEEE Trans. Knowledge and Data Eng.*, vol. 9, no. 3, pp. 353-372, May/June 1997.
- [11] K.C.K. Lee, H.V. Leong, and A. Si, "Semantic Data Broadcast for a Mobile Environment Based on Dynamic and Adaptive Chunking," *IEEE Trans. Computers*, vol. 51, no. 10, pp. 1253-1268, Oct. 2002.
- [12] 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.
- [13] W.-C. Peng, J.-L. Huang, and M.-S. Chen, "Dynamic Leveling: Adaptive Data Broadcasting in a Mobile Computing Environment," *ACM/Kluwer Mobile Networks and Applications*, vol. 8, no. 4, pp. 355-364, Aug. 2003.
- [14] W.-C. Peng and M.-S. Chen, "Efficient Channel Allocation Tree Generation for Data Broadcasting in a Mobile Computing Environment," *ACM/Kluwer Wireless Networks*, vol. 9, no. 2, pp. 117-129, Mar. 2003.
- [15] T. Sakata and J.X. Yu, "Statistical Estimation of Access Frequencies: Problems, Solutions and Consistencies," *ACM/Kluwer Wireless Networks*, vol. 9, no. 6, pp. 647-657, Nov. 2003.
- [16] N. Shivakumar and S. Venkatasubramanian, "Efficient Indexing for Broadcast Based Wireless Systems," *ACM/Baltzer Mobile Networks and Applications*, vol. 1, no. 4, pp. 433-446, Dec. 1996.
- [17] C.J. Su, L. Tassioulas, and V.J. Tsotras, "Broadcast Scheduling for Information Distribution," *ACM/Baltzer Wireless Networks*, vol. 5, no. 2, pp. 137-147, Mar. 1999.
- [18] D.G. Thaler and C.V. Ravishankar, "Using Name-Based Mappings to Increase Hit Rates," *IEEE/ACM Trans. Networking*, vol. 6, no. 1, pp. 1-14, Feb. 1998.
- [19] N.H. Vaidya and S. Hameed, "Scheduling Data Broadcast in Asymmetric Communication Environments," *ACM/Baltzer Wireless Networks*, vol. 5, no. 3, pp. 171-182, May 1999.
- [20] M.A. Viredaz, L.S. Brakmo, and W.R. Hamburgren, "Energy Management on Handheld Devices," *ACM Queue*, vol. 1, no. 7, pp. 44-52, Oct. 2003.
- [21] J. Xu, Q. Hu, W.-C. Lee, and D.L. Lee, "Performance Evaluation of an Optimal Cache Replacement Policy for Wireless Data Dissemination," *IEEE Trans. Knowledge and Data Eng.*, vol. 16, no. 1, pp. 125-139, Jan. 2004.
- [22] 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.
- [23] 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.
- [24] J. Xu, X. Tang, and D.L. Lee, "Performance Analysis of Location-Dependent Cache Invalidation Schemes for Mobile Environments," *IEEE Trans. Knowledge and Data Eng.*, vol. 15, no. 2, pp. 474-488, Mar./Apr. 2003.
- [25] J. Xu, X. Tang, and W.-C. Lee, "Time-Critical On-Demand Data Broadcast: Algorithms, Analysis, and Performance Evaluation," *IEEE Trans. Parallel and Distributed Systems*, vol. 17, no. 1, pp. 3-14, Jan. 2006.
- [26] J. Xu, B. Zheng, W.-C. Lee, and D.L. Lee, "The D-Tree: An Index Structure for Planar Point Queries in Location-Based Wireless Services," *IEEE Trans. Knowledge and Data Eng.*, vol. 16, no. 12, pp. 1526-1542, Dec. 2004.
- [27] J.X. Yu and K.L. Tan, "An Analysis of Selective Tuning Schemes for Nonuniform Broadcast," *Data and Knowledge Eng.*, vol. 22, no. 3, pp. 319-344, May 1997.
- [28] B. Zheng, J. Xu, W.-C. Lee, and D.L. Lee, "Energy-Conserving Air Indexes for Nearest Neighbor Search," *Proc. Ninth Int'l Conf. Extending Database Technology*, pp. 48-66, Mar. 2004.
- [29] G.K. Zipf, *Human Behavior and the Principles of Least Effort*, Addison-Wesley, 1949.



Yuxia Yao received the BSc degree in computer science in 2003 from Huazhong University of Science and Technology, Wuhan, China. Currently, she is a PhD candidate in the School of Computer Engineering at Nanyang Technological University, Singapore. Her research interests include mobile computing, wireless sensor networks, and spatial databases.



Xueyan Tang received the BEng degree in computer science and engineering from Shanghai Jiao Tong University, Shanghai, China, in 1998 and the PhD degree in computer science from the Hong Kong University of Science and Technology in 2003. He is currently an assistant professor in the School of Computer Engineering at Nanyang Technological University, Singapore. He has served as a program committee member for IEEE Infocom '04 and WWW '05. He is also an editor of the book *Web Content Delivery* (Springer). His research interests include mobile and pervasive computing, wireless sensor networks, Web and Internet, and distributed systems, particularly the data management aspects of these areas. He has published more than 30 papers in prestigious journals and conferences. He is a member of the IEEE.



Ee-Peng Lim received the PhD degree from the University of Minnesota, Minneapolis in 1994 and the BSc degree in computer science from National University of Singapore. He is an associate professor with the School of Computer Engineering, Nanyang Technological University, Singapore. He is the head of the division of information systems at the School of Computer Engineering of the Nanyang Technological University. His research interests include information integration, data/text/web mining, digital libraries, and wireless intelligence. His papers have appeared in the *ACM Transactions on Information Systems*, the *IEEE Transactions on Knowledge and Data Engineering*, *Data and Knowledge Engineering*, and other major journals. He is currently an associate editor of the *ACM Transactions on Information Systems*, the *International Journal of Digital Libraries*, and the *International Journal of Data Warehousing and Mining*. He was a program cochair of the 2004 ACM/IEEE Joint Conference on Digital Libraries and a conference and program cochair of the 2004 International Conference on Asian Digital Libraries. He was a cochair of the third, fourth and fifth ACM Workshops on Web Information and Data Management. He is a senior member of the IEEE and a member of the ACM.



Aixun Sun received the PhD and BASc degrees with first-class honors from the Nanyang Technological University, Singapore, in 2004 and 2001, respectively, both in computer engineering. He is an assistant professor with the School of Computer Engineering, Nanyang Technological University, Singapore. His research interests include text/Web mining, information retrieval, and machine learning. His papers have appeared in the *IEEE Transactions on Knowledge and Data Engineering*, the *Journal of the American Society for Information Science and Technology*, *Knowledge and Information Systems*, and major data mining conferences including CIKM and ICDM. He is a member of the IEEE, the IEEE Computer Society, and the ACM.