Interpolating the Air for Optimizing Wireless Data Broadcast

Fotis Tsakiridis
Computer & Communication Engineering Department, University of Thessaly, Volos, Greece
fotsakir@uth.gr

Panayiotis Bozanis
Computer & Communication Engineering Department, University of Thessaly, Volos, Greece
pbozanis@inf.uth.gr

Dimitrios Katsaros*†
Computer & Communication Engineering Department, University of Thessaly, Volos, Greece
dkatsaro@csd.auth.gr

ABSTRACT

Energy conservation and access efficiency are two fundamental though competing goals in broadcast wireless networks. To tackle the energy penalty from sequential searching, the interleaving of *index* with data items has been proposed. The current broadcast indexes present significant shortcomings. This article proposes a novel parameterized air index, the *interpolation index*, which is a tunable structure able to optimize the latency with the tuning time kept at a given amount, and vice versa. Theoretical and experimental results attest that the novel indexing structure outperforms the state-of-the-art air indexing scheme.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; E.1. [**Data**]: Data Structures—*Distributed data structures*

General Terms

Design, Experimentation, Measurement, Performance.

Keywords

Indexing, Interpolation Search, Energy Conservation, Latency, Tuning Time, Mobile Computing, Wireless Networks.

1. INTRODUCTION

The rapid advent of wireless technology along with the growing popularity of smart mobile devices, led to the deployment of pervasive information services, which provide access to "ambient" information to large number of audiences (clients), anywhere, and anytime, through access points and a number of wireless channels. Additionally, the advances in miniaturization and the creation of low-power circuits, combined with small-sized batteries have made the development of wireless sensor networks a working reality, boosting the deployment of wireless networks even further and pushing the challenges concerning their development to the limits.

Consider the following scenario encountered in a cellular wireless network (e.g., a PCS), where resource-constraint mobile units within a wireless cell, retrieve information from a relational database, whose contents are repetitively broadcast by a base station serving the cell; in the general case, the information pieces consist of thousands of "projections" (i.e., columns) of relational table rows. The existence of wireless data broadcast service providers, such as Ambient[1] and Microsoft[2], confirms the industrial interest in such kind of services and exhibits their feasibility.

In such an application scenario, it is evident that:

- The information "consumers" need to retrieve the data as quick as possible (i.e., with small *access latency*, which is the time elapsed between when the need for a datum arises in a node and the moment the node gets that datum from the channel).
- The consumers are energy-starving nodes; therefore they should refrain from continuously monitoring the broadcast channel (i.e., pursuit a small *tuning time*, which is the amount of time a node spends while monitoring the channel).
- There could be several thousands of broadcast items, thus scalability in terms of broadcast items is a very important issue.

Access efficiency is a common target in many systems, (e.g., databases), but energy conservation is a vital goal in wireless networks for prolonging the longevity of the sensor network or for guaranteeing as much power-independence as possible for the mobile hosts. To achieve energy savings, mobile nodes support two generic modes of operation, the *active* mode, which is a fully operational state, and the *doze*

*Corresponding author.

†Research supported by a *ΓΓΕΤ* grant in the context of the project "Data Management in Mobile Ad Hoc Networks" funded by *ΠΥΘΑΓΟΡΑΣ II* national research program.

MobiWac'07, October 22, 2007, Chania, Crete Island, Greece.

[1]Ambient Information Network and Device Design (http://www.ambientdevices.com)
[2]DirectBand Network, Microsoft Smart Personal Objects Technology, SPOT) (http://www.microsoft.com/resources/spot)

mode[3], which is a power saving state. The ratio of energy consumption between the two modes is usually an order of magnitude [24]. Similarly, sensor nodes can be in one of three *active* states – transmit, receive, idle – or in *sleep* state; a sensor in the sleep state consumes 7–20 times less energy than when it is in the idle state [4].

As it is easily seen, access latency and tuning time are competing each other: to acquire the requested data as soon as possible, the client must actively listen to the broadcasting channel—retrieving mainly unwanted information — therefore consuming energy, and vice versa. Apparently, every solution for trading off access latency and tuning time must provide some auxiliary information which allows clients to alternate from doze mode to active mode when browsing the broadcast data. Thus, clients can remain in the doze mode most of the time and tune selectively into the broadcast channel only when significant data arrive. This set up calls out for a sort of directory or indexing information which specifies the arrival times of particular items over the broadcast channel. By accessing this index, known as *air index*, mobile clients are able to predict the arrival of desired data. Obviously, this scheme demands additional bandwidth for broadcasting the index. However, its costless scalability feature compensates for the incurred overhead and, so, it became the method of choice.

Two main indexing approaches have been were introduced in the literature: i) multiple replicated indexes are interleaved with data broadcast [11]; and ii) transmitted data are linked together through index information so that multiple entry points exist during each broadcast cycle [10].

In the following, we deal with the second approach since proven superior to the first one. The most prominent member of this category of air indexes is the *exponential index* [25]. This scheme can be loosely seen as the linearization of a directed acyclic graph (dag), laid over the sorted sequence of data items to be broadcasted, so that the client can simulate the binary search operation, irrespectively of the time point he tunes into. Alternatively, it can be seen as a distributed implementation of skip lists, i.e., a skip list with multiple entry points, which try to simulate the exponential searching technique [15] over the periodical broadcast. As a result, the exponential index shows logarithmic access complexity and resilient behavior to link errors due to bad transmitted packets. Even though this access time complexity is acceptable for a few hundreds of items, still is not satisfactory when dealing with a few thousands of items.

In this paper, we are mainly motivated by the scalability problem of the indexing structures, and improve upon the state-of-the-art indexing method, namely the exponential index. Deviating from the philosophy of this index, we recursively apply a squared-distance partitioning of the linear broadcast order, which, effectively, overlays index search paths of log-logarithmic length in the average case. Additionally, we implement this technique in a distributed fashion, so that it allows index access from any tuning point, permitting the recovery from link errors quickly and easily. This novel air indexing scheme is called the *interpolation air index*. Theoretical and experimental results attest that the novel indexing structure outperforms state-of-the-art air indexing schemes and demonstrate its great flexibility in trading access latency with tuning time.

[3]Following the terminology of [11].

The rest of the paper is organized as follows. Section 2 reviews basic notions and related work. Section 3 introduces the interpolation air index. In Section 4 we provide both theoretical and experimental evidence on the superiority of our scheme over exponential index. Finally, Section 5 concludes our work.

2. PRELIMINARIES

2.1 Basic notions

We consider a generic data broadcasting system, where a server cyclically broadcasts a collection of totally ordered data items onto a down-link wireless channel. The mobile clients must tune into the broadcast channel and actively must find their way to the required information. In order to aid the search process, the server interleaves auxiliary index items with the actual data items, forming a broadcast cycle, i.e., *bcast*. Every bcast is organized as a sequence of equal sized *buckets*, which constitute the smaller unit of information a mobile client has access to. Each bucket is categorized as being either *data bucket* when it contains a number of (pure) data items or *index bucket*, in case it holds index information. Sometimes, an index bucket may accommodate and some data items, and, then, it is termed as *hybrid* one. In this context, a pointer to a specific bucket is defined as an offset from the bucket containing the pointer to the bucket to which the pointer points to, and designates the number of basic bucket transmission time units one has to wait to start retrieving the pointed bucket. It is widely accepted that every data bucket contains a pointer to the immediately upcoming index bucket. As it is easily seen, the employment of index buckets reduces the tuning time, however it increases the access time.

2.2 Related work

Air-indexing has received much attention during the last years after its introduction in the seminal papers [10, 11]. In [11] $(1, m)$-indexing was introduced as an index allocation method, according to which the index information is broadcasted m times during each bcast. The main problem with the $(1, m)$-indexing scheme is the replication of the entirety of the index m times, since it prolongs the broadcast cycle and thus the average access time. In the same paper, a tree-based indexing method, called distributed indexing, was also suggested: the data file is associated with a B^+-tree, and since the wireless channel is a sequential medium, the formed tree is linearized with a pre-order traversal. Additionally, the first k levels of the index are partially replicated in the broadcast, while the remaining levels are not. The nodes at the replicated levels are repeated at the beginning of the first broadcast of each of their children. Compared to the $(1, m)$-index, the tree-based scheme has lower access time due to its shorter broadcast cycle while its tuning time is analogous to that of the $(1, m)$-index.

In [10] it was exhibited how hash functions can be used for allocating data items to the slots in the broadcast schedule. Since collisions — that is, multiple items are mapped to the same slot — are inevitable, the authors adopted the linear probing method for collision resolution. Namely, overflow items are relocated into succeeding slots, pushing forward every item originally hashed to them, and, thus, penalize them with an extra tuning time of one slot. The hash-based scheme incurs minimal overhead, compared to the over in-

dexing techniques, since only the hash function is broadcast together with data. However, one has to examine the entirety of the implicit partitions, corresponding to areas of overflowed items, to find the desired item. This may incur high tuning time, especially for large partitions. This approach was extended in [26] to the case of skewed bcasts, by introducing the *MHash air-index*.

In [10], the flexible indexing scheme was also proposed, according to which, the sorted sequence of data items is partitioned into several equal-sized segments. At the beginning of each segment, a global index and a local index are accommodated. The global index at a segment contains a logarithmic number of (key, pointer) pairs to guide the search towards succeeding segments. On the other hand, the local index holds m (key, pointer) pairs that split further the hosting segment into $m + 1$ subsegments, and, thus, comprises an inside directory. This approach was generalized in [25], by introducing the exponential index. Specifically, the sizes of the indexed segments increase exponentially by a base of $r \geq 1$, r being a system parameter. This approach was further studied in error-prone environments [25].

Seifert and Hung [21] suggested the flexible distributed indexing scheme which employs partitioning of the broadcast program into a number of equal-sized data segments, such that a given limit on the tuning time will not be exceeded, in conjunction with multiplexing of a dense B+tree-like index on the data items. Similar to $(1, m)$-indexing, index information is broadcast multiple times during a bcast. Both [22, 2], considered unbalanced tree structures to optimize broadcast schedule for non-uniform data access, while Tan and Yu [23] studied scheduling policies for skewed bcasts.

Signatures have been successfully employed to facilitate retrieval in several types of databases and was adapted to broadcast environments in [13]. A hybrid among the signature method and the distributed index tree was proposed in [7] and applied it also to the multi-attribute indexing case [8, 9]. Remotely related to the present article, are the air indexing schemes that have been proposed for the case of multiple broadcasting channels [1, 6, 12, 19].

For the case of wireless sensor networks, since the majority of research has focused for the moment on topics like routing, clustering, sleep scheduling, localization, medium access control, the issue of indexing has received much less attention and the literature has solely developed distributed indexes that reside on the sensor nodes and are not broadcast. These indexes comprise (in one form or another) adaptations of the traditional disk-based indexes, with special care to achieve only local (to the extend possible) communication during their creation or maintenance, and small storage overhead. The GHT [20] is based on a (geographic) hashing scheme, DIM [14], DIFS [5] and DIST [16] are based on the quadtree structure, and TSAR [3] is based on Skip Graphs (a generalization of Skip Lists for distributed environments). None of these indexes is broadcast over wireless channels and they all assume global ordering for the data they index.

3. INTERPOLATION AIR INDEXING

The method proposed in the present article exploits the idea of recursively applying a squared-distance partitioning of the linear broadcast order, which, effectively, overlays index search paths of log-logarithmic length, in the average case. This idea is further improved by implementing it in a distributed fashion to allow for multiple "entry" points from the broadcast. Firstly, we exemplify the idea behind this log-logarithmic scheme by applying it in a ordinary array of items (the word 'ordinary' means that the items are not broadcast, but reside in a computer's main memory).

3.1 The idea behind the log-log scheme

Suppose that there is an array of totally ordered numeric values (keys) $x_1 < x_2 < \cdots < x_n$, drawn independently from a uniform distribution over the range (x_0, x_{n+1}). Searching by interpolation for the item y in the array proceeds as follows: Let $p = \frac{y - x_0}{x_{n+1} - x_0}$, that is the percentage of the keys expected to be less than y. Then, we compare y to $x_{\lceil p*n \rceil}$, and, in case of inequality, we search recursively either subarray $x_1, x_2, \ldots, x_{\lceil p*n \rceil - 1}$ $(y < x_{\lceil p*n \rceil})$ or subarray $x_{\lceil p*n \rceil + 1}, x_{\lceil p*n \rceil + 2}, \ldots, x_n$ $(y > x_{\lceil p*n \rceil})$.

The access time is $\log \log n$ in the average case, and linear in the worst case [17]. This approach, with the same time bounds, can be employed in every totally ordered data set whose cumulative distribution function is known.

A further improvement to this scheme can be achieved [18], namely the binary interpolation search, which modifies the actions after comparing y to $x_{\lceil p*n \rceil}$, as follows: If $y > x_{\lceil p*n \rceil}$, then y is successively compared with $x_{\lceil p*n + i\sqrt{n} \rceil}, i = 1, 2, \ldots$, to locate the smallest i such that $y < x_{\lceil p*n + i\sqrt{n} \rceil}$. If $y < x_{\lceil p*n \rceil}$, then y is successively compared with $x_{\lceil p*n - i\sqrt{n} \rceil}, i = 1, 2, \ldots$ In either case, the located pertinent subarray of size $\sqrt{n}$ is recursively searched.

It can be shown that the average access time is bounded by $2.03 \log \log n$, but its worst case time complexity is $\sqrt{n} + O(\sqrt[4]{n})$. It can be relatively easily showed that the worst case complexity of this approach can be further improved to $2 \log n$, if one employs exponential search to determine i with $\log i$ comparisons [15], without affecting the average case performance.

3.2 Description of the proposed Interpolation index

The bcast in our proposal consists of n hybrid buckets, $b_1, b_2, \ldots, b_n$. The embedded index entries will guide the search, simulating the interpolation search in the wireless set up. In the following, i) M_i, m_i will denote the maximum and the minimum data entry, respectively, of i-th bucket — therefore, every data element belongs to the range $[m_1, M_n]$; iii) m_1 and M_n will be also available to every bucket; and iv) $\mathcal{F}_{\mathcal{R}}(y) = \mathbf{Prob}[Y \leq y | Y \in \mathcal{R}]$.

In the first version, apart from m_1 and m_n, the index part of k-th bucket, $k = 1, 2, \ldots, n$, will consist of pairs $(pntr, maxKey)$, where $pntr$ is an offset and $maxkey$ is the maximum data entry of the bucket $pntr$ slots ahead. The 0-th entry refers to the immediately upcoming bucket, while the i-th entry holds $m_{k+n^{2^{-i}}}$, namely, the maximum data entry $n^{\frac{1}{2^i}}$ slots ahead; every such entry is characterized as i-th level. It follows immediately that:

LEMMA 1. *The space overhead of every accommodated index structure is* $1 + \log \log n$ *entries.*

Please note that the offsets are immediately inferred given the sequence id of the hosting bucket and, therefore, are not stored.

Based on stored information, the access protocol is as follows (please cf. Fig. 1): Let k be the bucket the client currently tuned into, y the search item, l the level of recursion,

and j the probing position, calculated using $\mathcal{F}$. When y belongs to the key range of either k or $k+1$, or equals to one of the maximum values M_i of the indexed buckets, then we do not have much to do. So, in the sequel we assume that none of these cases occurred.

During the first time the client tunes into (i.e., $l = 1$), three possibilities, besides the general case (see below) may happen. When both j and y lie ahead of k (Fig. 1(a)), the rest of the bcast is useless. Thus, we check whether $\frac{j}{\sqrt{n}} \leq c_d$, c_d a tuning distance parameter. If so, the client dozes until bucket 1 arrives, and then he conducts linear search with $\sqrt{n}$ long jumps, using level 1 pointers, until he locates the left delimiting bucket of the $\sqrt{n}$-sized subfile containing y, to which we recur; since, on the average, y lies within approximately $\pm 2\sqrt{n}$ slots from j ([18]), and bucket 1 is close enough to j, we adopt this conservative policy, to avoid a possible second missed bcast. Otherwise, the client switches to doze mode until bucket $j - c_a\sqrt{n}$ arrives, c_a being an interpolation adjustment parameter. Again, we prefer to "step" back a little from j, trading-off an increase of tuning time with sparing some latency time, by decreasing the probability of a second consecutive unutilized bcast. In case of successful prediction, we start employing the above mentioned linear search to narrow down the area of interest in a sub-transmission of $\sqrt{n}$ extent. If, however, a second missed bcast does happen, we employ the linear search starting from bucket 1.

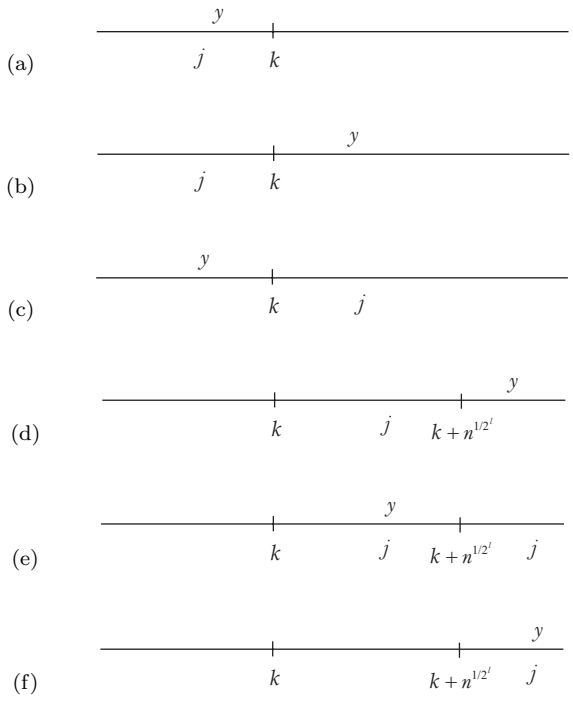


Figure 1: Relative positions of k, j and l: (a)-(c) can happen only when $l = 1$, (d)-(f) general recursive cases ($l \geq 1$).

In the second case, j precedes k and y succeeds both of them (Fig. 1(b)); we simply start switching between doze and active mode, with $\sqrt{n}$ intervening slots, until we locate the desired $\sqrt{n}$-sized interval, to which we recur. When y precedes k and j succeeds k (Fig. 1(c)), the client missed the current bcast. This situation is similar to the one of Figure 1(a), with k substituting j. Specifically, if $\frac{k}{\sqrt{n}} \leq c_d$, the client dozes until the arrival of bucket 1 to conduct the linear search. Else, the client goes back to active mode when bucket $k - c_a\sqrt{n}$ arrives. If he is fortunate in his prediction, linear searching commences; otherwise, the bcast is useless, he dozes and applies the linear search starting from bucket 1.

Turning now to the general case ($l \geq 1$), the client already knows that y is lying between buckets k and $\min\{k + \sqrt[2^{l-1}]{n}, n\}$, the interpolation obviously returns an index $j \in [k, \min\{k + \sqrt[2^{l-1}]{n}, n\}]$, and the length of jumping during linear searching equals to $\sqrt[2^l]{n}$. There are also tree possibilities. In the first one, y seems to belong after the position the level l pointer points to, i.e., bucket $k + \sqrt[2^l]{n}$, while j lies between k and $k + \sqrt[2^l]{n}$ (Fig. 1(d)). Then, we doze until bucket $k + \sqrt[2^l]{n}$ is transmitted and we start the linear search procedure. When y is located between k and $k + \sqrt[2^l]{n}$, (Fig. 1(e)), irrespectively of j's position, we have nothing to do; the $\sqrt[2^l]{n}$ subfile, to which we must recur, has been found. In the last case (Fig. 1(f)), both y and j lie after bucket $k + \sqrt[2^l]{n}$. If $j - c_a\sqrt[2^l]{n} < k + \sqrt[2^l]{n}$ holds, we linearly search the rest of the bcast from slot $k + \sqrt[2^l]{n}$. Else, we doze until bucket $j - c_a\sqrt{n}$ arrives. If the adjusted prediction is proven to be true, we execute the linear search; otherwise, we missed the bcast and linear searching is performed from "secure" bucket $k + \sqrt[2^l]{n}$.

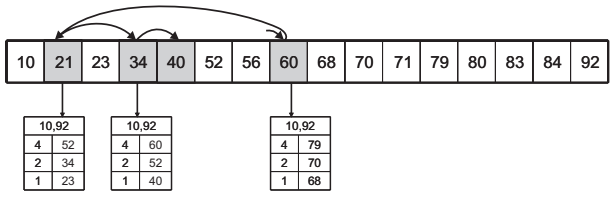


Figure 2: An instance of searching for item 40 with interpolation index.

The described protocol is presented in the appendix, while Figure 2 illustrates a searching instance, where we assume that $c_d = 1.2$, $c_a = c_x = 1$, and 40 is the key we search. We initially tune into bucket 8. Since $40 < 60$, we interpolate and get $j = \left\lceil \frac{40-10}{92-10} \cdot 16 \right\rceil = 6$. Since $\frac{6}{\sqrt{16}} = 1.5 > 1.2$, we decide to switch into doze mode until the bucket $6 - 1 \cdot \sqrt{16} = 2$ arrives, where we have $21 < 40$, and, thus, we must continue linear searching in upcoming buckets. Since level 1 pointer (52) bounds 40, we recur in level 2. We apply once again interpolation which give us $j = 2 + \left\lceil \frac{40-21}{52-21} \cdot 4 \right\rceil = 5$. Because 40 is located after $2 + \sqrt[4]{16} = 4$, we sleep until bucket 4 is transmitted, where we figure out that 40 belongs to the following bucket 5.

As far as tuning time is concerned, the following Lemma holds, whose proof is omitted due to space limitations:

LEMMA 2. *The average tuning time is $O(\log \log n)$.*

4. PERFORMANCE EVALUATION

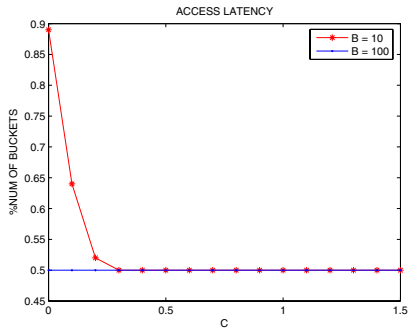
This section provides a detailed study of the performance of the proposed index against the state-of-the-art scheme of exponential index [25], using similar system parameters in order to conduct fair comparisons. The database size

n ranges from 1000 to 1000000 items. Additionally, the database is characterized as small one when N between 1000 and 50000; otherwise, is considered as a big one. In this article, similar to [25] we present the results for uniform access pattern; the conducted experiments for skewed (Zipfian) access patterns favor the interpolation index even more. We have also investigated different combinations of bucket capacity B, ranging thus the item size. In the sequel, we report the results for $B = 10$ and $B = 100$; intermediate values for B led to the same conclusions. Finally, the tuning time and access latency, are both measured in terms of number of buckets.

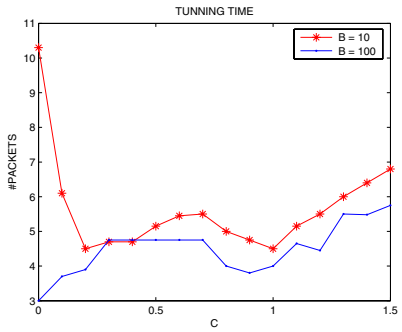
4.1 Tuning the Interpolation index

Our first experiment investigated the impact of parameters c_d and c_a on the performance of the interpolation index. Due to space limitations, we consider only the case $c_a = c_d = C$. Figure 3 depicts our findings when $N = 30\,000$, B is either 10 or 100, and C varies between 0 (i.e., the index 'blindly' trusts the interpolation estimation) and 1.5 (we are a bit conservative and adjust the estimation by going back 1.5 steps; we have arrived to analogous conclusions for various values of N, while being more conservative, that is, $C > 1.5$ proven to deteriorate the index performance.

Figure 3(a) shows that every value greater or equal to 0.3 is fine for bringing the average latency time to 50% of the bcast. On the other hand, $C = 0.9$ appears to be the best value for achieving the overall best tuning time performance. This value is used for the rest of the experimental section, and as we will see, certainly beats the exponential index, proving that our proposal is a very easily configurable indexing scheme.



(a)



(b)

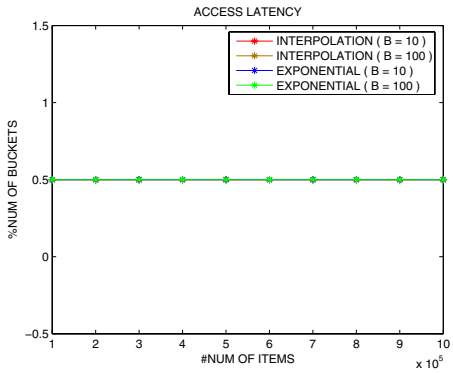
Figure 3: Investigation of the C parameter when $N = 30\,000$: (a) access latency; and (b) tuning time w.r.t. the C parameter.

4.2 Comparison with the Exponential index

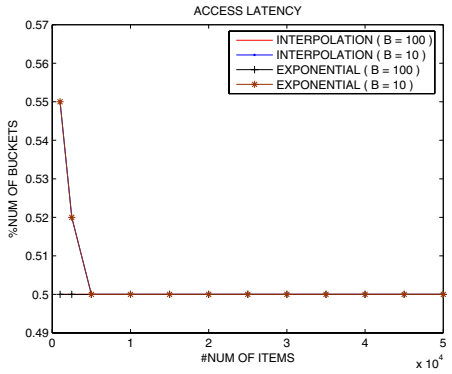
In this section, we compare our proposal to the exponential index. The investigation is twofold: we explore both the time as well as the space overhead of the two proposals.

4.2.1 Access latency and tuning time

Firstly, we explored the tuning time performance of both schemes. The comparison was performed on the basis of tuning the Interpolation air index to achieve the same access time with that achieved by the exponential index and then comparing their tuning time performance. The independent parameter was the number of broadcast items. The performance of the indexes was compared for both small and large databases, and for the case where the parameter r of exponential index was set to $r = 2$ and $r = 3$. The obtained results are depicted in Figures 4(a)-(d). Confirming the conclusions of [25], we observe that the tuning time of exponential index increases as the number of broadcast items gets larger. The increase is more steep for large number of items. On the contrary, the tuning time incurred by the Interpolation index is almost constant for small database sizes and increases moderately for large databases, as a result of its log log performance.



(a)



(b)

Figure 5: Average access latency: (a) big data base with $r = 2$; (b) small data base with $r = 2$.

In Figures 5(a)-(b) we performed the reverse experiment: we kept the tuning time the same for both schemes and

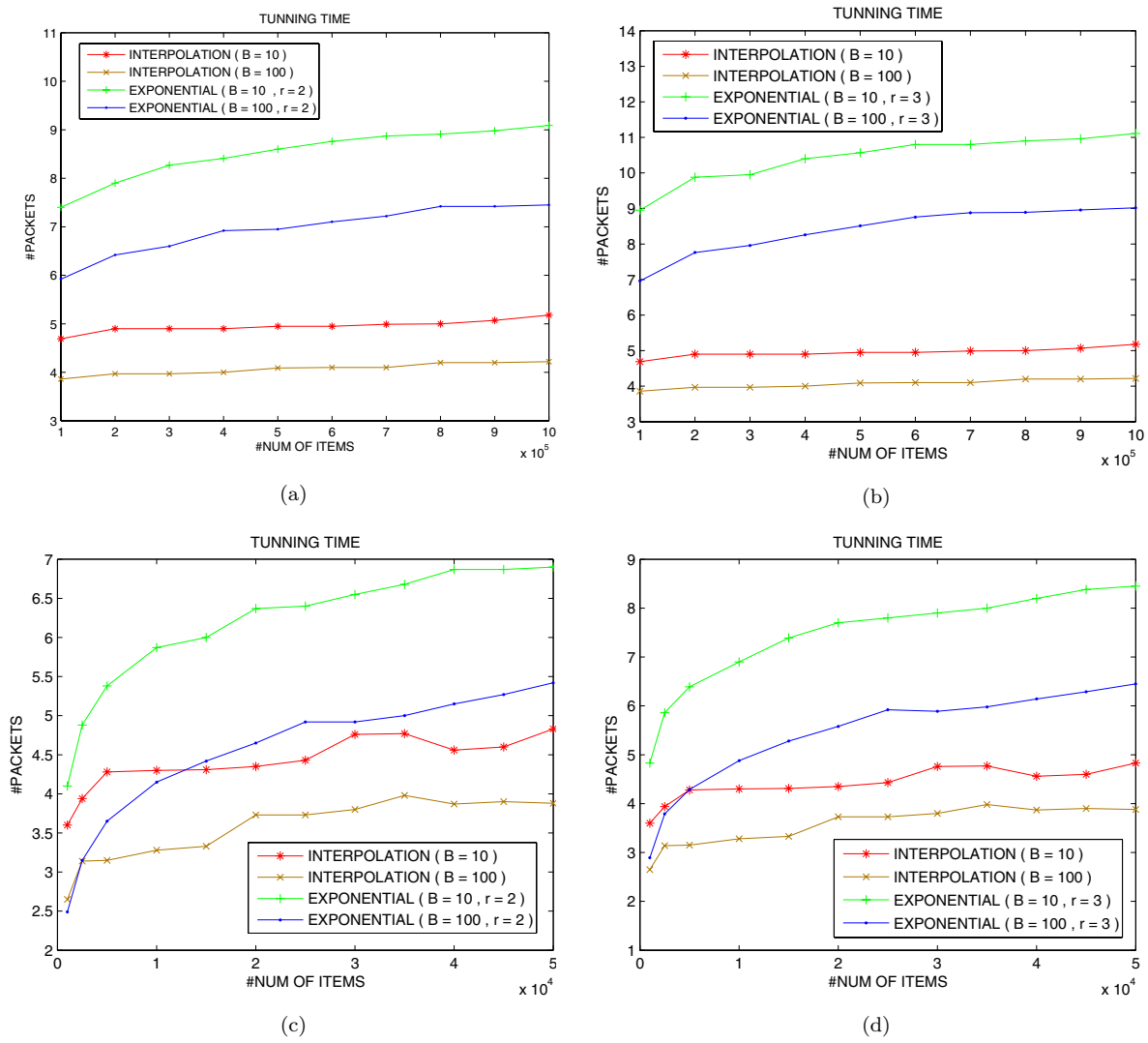


Figure 4: Average Tuning Time: (a) big data base with $r = 2$; (b) big data base with $r = 3$; (c) small data base with $r = 2$; and (d) small data base with $r = 3$.

examined the access latency. We can see that our scheme achieves the same average access latency performance, irrespectively of the data base size, while the exponential index is slightly worst for small data sets.

4.2.2 *Index overhead investigation*

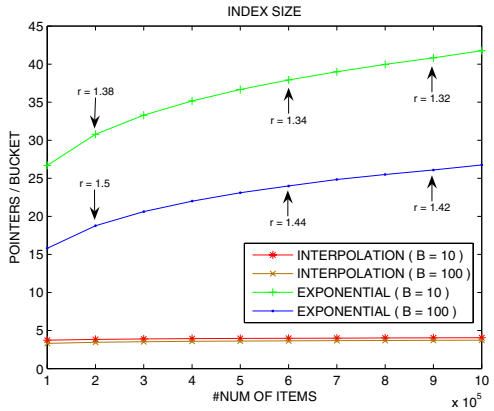
This experiment shows the results of the interrelation among the tuning time and the index overhead which is measured in "pointers" per transmitted bucket. Following the same policy as before, we performed the investigation by keeping constant for both indexes the first "quantity" and measuring the other, and vice versa. Firstly, we evaluated the average index overhead for the same tuning time; to achieve the same tuning time, we adjusted the index base r of the exponential index. The results are illustrated in Figure 6. It is obvious that on the average, the overhead of exponential index is four times larger than the respective of the interpolation index! This implies that the length of the broadcasting program generated by the exponential index is much larger than the program generated by the interpolation index.

Then, we forced both schemes to employ the same index space overhead and investigated the tuning time performance. Figure 7 confirms that interpolation air index undoubtedly outperforms the exponential index by almost an order of magnitude.

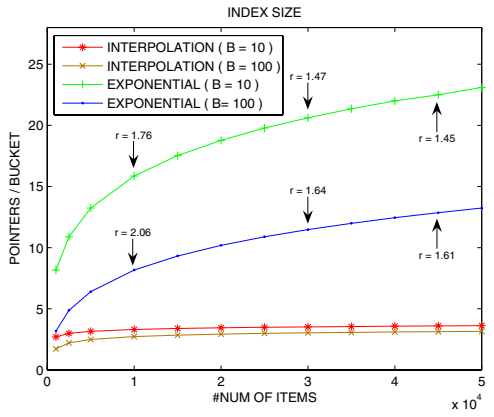
5. DISCUSSION AND CONCLUSION

Modern wireless broadcasting systems, like Microsoft's SPOT technology or emerging applications of wireless sensor networks, involve the broadcasting of several thousands of items over wireless channels. Given the fact that mobile nodes of cellular systems or sensor nodes are energy-starving devices, it is mandatory that the access to the transmitted information is as energy-conserving as possible. This requirement calls for the deployment of distributed air indexes able to scale up to larger than ever numbers of items, and tunable so as to be able to tradeoff energy-conservation for access latency when needed.

This article is motivated by the aforementioned requirement, and proposes the *interpolation air index*, a very easily



(a)



(b)

Figure 6: Average index overhead when both schemes exhibit the same tuning time performance: (a) big data base; and (b) small data base.

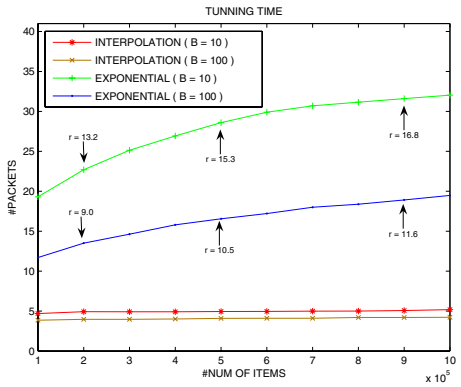


Figure 7: Average tuning time when both schemes exhibit the same index space overhead (big data base).

configurable and efficient air indexing scheme. This scheme exhibits a linear structure, suiting the broadcast environment very well. The index space overhead is log-logarithmic per transmitted bucket, while the tuning time is log-logarithmically proportional to the broadcast size. Additionally, the access latency and tuning time of the interpolation index can be simply adjusted by a single parameter. To evaluate the suitability and behavior of the proposed novel air index, we investigated its performance against the state-of-the-art exponential index [25]. The experimental results attest that our index outperforms the exponential index both in tuning time and space overhead, while achieving the same access latency.

There are a number of issues that are left for future work. Firstly, we plan to investigate how skewed data can be accommodated in our scheme. Secondly, it is very interesting to explore its performance in multi-channel data broadcast environments.

6. REFERENCES

[1] D. Amarmend, M. Aritsugi, and Y. Kanamori. An air index for data access over multiple wireless broadcast channels. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, page 135, 2006.

[2] M.-S. Chen, K.-L. Wu, and P. S. Yu. Optimizing index allocation for sequential data broadcasting in wireless mobile computing. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):161–173, 2003.

[3] P. Desnoyers, D. Ganesan, and P. Shenoy. TSAR: A two tier sensor storage architecture using interval skip graphs. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 39–50, 2005.

[4] L. M. Feeney. Energy efficient communication in ad hoc wireless networks. In S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, editors, *Mobile Ad Hoc Networking*, pages 301–327. IEEE/Wiley, 2004.

[5] B. Greenstein, S. Ratnasamy, S. Shenker, R. Govindan, and D. Estrin. DIFS: A distributed index for features in sensor networks. *Ad Hoc Networks*, 1(2–3):333–349, 2003.

[6] C.-H. Hsu, G. Lee, and A. L. P. Chen. Index and data allocation on multiple broadcast channels considering data access frequencies. In *Proceedings of the Conference on Mobile Data Management (MDM)*, pages 87–93, 2002.

[7] Q. L. Hu, W.-C. Lee, and D. L. Lee. Indexing techniques for wireless data broadcast under data clustering and scheduling. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 351–358, 1999.

[8] Q. L. Hu, W.-C. Lee, and D. L. Lee. Power conservative multi-attribute queries on data broadcast. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 157–166, 2000.

[9] Q. L. Hu, W.-C. Lee, and D. L. Lee. A hybrid index technique for power efficient data broadcast. *Distributed and Parallel Databases*, 9(2):151–177, 2001.

[10] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Power efficient filtering of data on air. In *Proceedings of the International Conference on Extending Data Base Technology (EDBT)*, pages 245–258, 1994.

[11] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air: Organization and access. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):353–372, 1997.

[12] S. Jung, B. Lee, and S. Pramanik. A tree-structured index allocation mathod with replication over multiple broadcast channels in wireless environments. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):311–325, 2005.

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

[14] X. Li, Y.-J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 63–75, 2003.

[15] K. Mehlhorn. *Data Structures & Algorithms, I: Sorting and Searching*. Springer-Verlag, Berlin, 1984.

[16] A. Meka and A. K. Singh. DIST: A distributed spatio-temporal index structure for sensor networks. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 139–146, 2005.

[17] Y. Perl, A. Itai, and H. Avni. Interpolation search — A $\log \log N$ search. *Communications of the ACM*, 21(7):550–553, 1978.

[18] Y. Perl and E. M. Reingold. Understanding the complexity of interpolation search. *Information Processing Letters*, 6(6):219–222, 1977.

[19] K. Prabhakara, K. Hua, and J. Oh. Multi-level multi-channel air cache designs for broadcasting in a mobile environment. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 167–176, 2000.

[20] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-centric storage in sensornets with GHT: A geographic hash table. *ACM Mobile Networks and Applications*, 8(4):427–442, 2003.

[21] A. Seifert and J.-J. Hung. FlexInd: A flexible and parameterizable air-indexing scheme for data broadcast systems. In *Proceedings of the International Conference on Extending Data Base Technology (EDBT)*, volume 3896 of *Lecture Notes on Computer Science*, pages 902–920. Springer, 2006.

[22] N. Shivakumar and S. Venkatasubramanian. Efficient indexing for broadcast based wireless systems. *ACM/Baltzer Mobile Networks and Applications*, 1(4):433–446, 1996.

[23] K. L. Tan and J. X. Yu. Energy efficient filtering of nonuniform broadcast. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 520–527, 1996.

[24] M. A. Viredaz, L. S. Brakmo, and W. R. Hamburgen. Energy management on handheld devices. *ACM Queue*, 1(7):44–52, 2003.

[25] 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 Transactions on Knowledge and Data Engineering*, 18(3):392–404, 2006.

[26] Y. Yao, X. Tang, E.-P. Lim, and A. Sun. An energy-efficient and access latency optimized indexing scheme for wireless data broadcast. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1111–1124, 2006.

APPENDIX

AUXILIARY FUNCTION — SearchI

```
1.  SearchI(y, x, z, l)
2.  //probing of interpolated value x;z is the 'retry' bucket
3.    if (x != current bucket)
4.      doze until bucket x arrives;
5.    if (y < m_x)
6.      switch to doze mode until bucket z arrives;
7.      if ((m_z ≤ y ≤ M_z) || (M_z ≤ y ≤ M_{z+1}))
8.      //y belongs to z or z + 1
9.        search y among the data elements of bucket z
10.       or z + 1 and exit;
11.     if ((y == M_p) && (p among indexed buckets))
12.       switch to doze mode until bucket p arrives;
13.       return the result of search and exit;
14.     SearchL(y, z, l);
15.   else
16.     if ((m_x ≤ y ≤ M_x) || (M_x ≤ y ≤ M_{x+1}))
17.     //y belongs to x or x + 1
18.       search y among the data elements of bucket x
19.       or x + 1 and exit;
20.     if ((y == M_p) && (p among indexed buckets))
21.       switch to doze mode until bucket p arrives;
22.       return the result of search and exit;
23.     SearchL(y, x, l);
24. end of SearchI
```

AUXILIARY FUNCTION — SearchL

```
1.  SearchL(y, x, l) // linear search with n^{1/2^l} jumps
2.    b = x + n^{1/2^l};
3.    while (M_b < y)
4.    //known from the index of the current bucket b − n^{1/2^l}
5.      switch to doze mode until bucket b arrives;
6.      b = b + n^{1/2^l};
7.    InterpolationSearch(y, b − n^{1/2^l}, l + 1);
8.  end of SearchL
```

CLIENT ACCESS PROTOCOL FOR INTERPOLATION INDEX

Algorithm InterpolationSearch(y, k, l)
Input: The key item y to be searched, k the current bucket the client tuned into, and l the level of recursion
Output: The data bucket r such that $y \in [m_r, M_r]$, and may contain y

```
1.  if ((m_k ≤ y ≤ M_k) || (M_k ≤ y ≤ M_{k+1})) //y ∈ {k, k + 1}
2.    search y among the data elements of bucket k or k + 1;
3.    exit;
4.  if ((y == M_p) && (p among indexed buckets))
5.    switch to doze mode until bucket p arrives;
6.    return the result of search and exit;
7.  j = (l > 1 ? ⌈n^{1/2^{l−1}} F_{[m_k, M_{k+n^{2−l+1}}]}(y)⌉ : ⌈nF_{[m_1, M_n]}(y)⌉);
8.  if (j <= k)
9.    if (y < m_k) //this may happen only if l = 1
10.     if (⌈j / 2^l√n⌉ ≤ c_d)
11.     //within heuristic distance parameter c_d
12.       SearchI(y, 1, 1, l);//we search from bucket 1
13.     else //we will try the estimation, in case of failure
14.     //we will resume on bucket 1
15.       SearchI(y, j − c_a 2^l√n, 1, l);
16.   else //y > M_k, linear search from k
17.     SearchI(y, k, k, l);
18. else //j > k
19.   if (y < m_k)
20.     if (⌈k / 2^l√n⌉ ≤ c_d)
21.     //within heuristic distance parameter c_d
22.       SearchI(y, 1, 1, l);
23.     else //we will try the estimation
24.       SearchI(y, max{k − c_a 2^l√n, 1}, 1, l);
25.   else // y > M_k
26.     if (M_k < y < M_{k+ 2^l√n}) //the right subfile is found
27.     //recursive searching
28.       InterpolationSearch(y, k, l + 1);
29.     else // y > M_{k+ 2^l√n}
30.       if (j ≤ k + 2^l√n) //linear search from k + 2^l√n
31.         SearchL(y, k + 2^l√n, l);
32.       else //(j > k + 2^l√n) ∧ (y > M_{k+ 2^l√n})
33.         if (j − c_d 2^l√n > k + 2^l√n)
34.         //we will try the estimation
35.           SearchI(y, j − c_d 2^l√n, k + 2^l√n, l);
36.         else //linear search from k + 2^l√n
37.           SearchL(y, k + 2^l√n, l);
end of InterpolationSearch
```