Power Conservative Multi-Attribute Queries on Data Broadcast

Qinglong Hu*
Aleph Computer System, Inc.
1700 Shattuck Ave., Suite 1
Berkeley, CA 94709, USA
peter.hu@alephcomputer.com

Wang-Chien Lee
GTE Laboratories Inc.
40 Sylvan Road
Waltham, MA 02451, USA
wlee@gte.com

Dik Lun Lee
Dept of Computer Science
Univ. of Science and Technology
Clear Water Bay, Hong Kong
dlee@cs.ust.hk

Abstract

In this paper, we study power conservation techniques for multi-attribute queries on wireless data broadcast channels. Indexing data on broadcast channels can improve client filtering capability, while clustering and scheduling can reduce both access time and tune-in time. Thus, indexing techniques should be coupled with clustering and scheduling methods to reduce the battery power consumption of mobile computers. In this study, three indexing schemes for multi-attribute queries, namely, index tree*,* signature*, and* hybrid index*, are discussed. We develop cost models for these three indexing schemes and evaluate their performance based on multi-attribute queries on wireless data broadcast channels.*

1 Introduction

Wireless communication and mobile computing have gained much attention from the computer and communication research community. Among the various issues under study, efficient utilization of limited wireless bandwidth and battery power is critical for disseminating information to a large user population.

Wireless broadcast is an attractive approach for data dissemination, because it allows simultaneous information access by an arbitrary number of users, and thus facilitates efficient bandwidth usage. Meanwhile, mobile computers accessing data by monitoring broadcast channels consume less battery power than those accessing data by sending requests through traditional point-to-point communications. A lot of studies on data broadcast have appeared in the literature [1, 2, 9, 10, 3].

Power conservation techniques for broadcast data management include indexing methods and data organization methods (i.e., clustering and scheduling). Among the proposed indexing techniques, signature[8] and index-tree [7] methods represent two major classes[1] of indexing techniques for data broadcast. Evaluation has been conducted on each of the indexing technique. In recent papers [4, 5], we examined these two indexing techniques based on clustering and scheduling, and proposed a hybrid indexing method which combined the strength of the signature and index-tree methods.

All of the above studies focused on data management techniques for single-attribute based queries. In the real world applications, data items usually contain multiple attributes. Thus, multi-attribute based queries may provide more precise information to the users. In this paper, we investigate the issues of multiple-attribute based queries on wireless data broadcast channels. We study three different power conservative indexing techniques, namely *index tree*, *signature*, and *hybrid*, and their supporting data organizations for broadcast of multi-attribute data items. For each method, query processing algorithms for multi-attribute queries are presented. We develop cost models and evaluate the access time[2] and tune-in time[3] for the multiple attribute queries. Finally, empirical comparisons of the query performance are conducted.

The rest of the paper is organized as follows. Section 2 gives an informal introduction of the broadcast channels. In Section 3, indexing techniques and access methods for multiple attributes are introduced. Section 4 evaluates the indexing techniques and query efficiency for multiple attributes. Finally, Section 5 concludes the paper.

2 Wireless Data Broadcast

In a wireless data broadcast environment, a base station maintains a set of records for formatted data and multime-

*This work was done when the author was pursuing his doctorate at Hong Kong University of Science and Technology.

[1] In [6], hashing and flexible indexing methods were explored.

[2] The average time elapsed from the moment a query is issued to the moment when all the requested data frames are received.

[3] The period of time spent by a mobile computer staying active in order to obtain the requested data.

dia such as text, image, audio and video. To facilitate our discussion, we assume that each record consists of a number of common attributes. To support broadcast data delivery, the base station periodically delivers these records to its clients as a series of *data frames* through one *shared* broadcast channel. A data frame, the logical unit of information broadcasted on the air, consists of *packets*[4] which are the physical units of broadcast. A frame contains a header for synchronization and meta-information that indicates the type and length of the frame. Since data frames are periodically broadcast, a complete broadcast of data frames is called a *broadcast cycle*. From the user's viewpoint, broadcast data are perceived as a stream of frames flowing along the time axis. Logically, there is no specific start and end frame for a broadcast cycle. A broadcast cycle starts with any frame and ends when the frame appears again. Updates to the frames are reflected between successive broadcast cycles. To receive the frames from the air, mobile applications or users issue queries which specify values for one or multiple common attributes to their mobile computers. As a result, the mobile computers monitor the broadcast channels and return the frames that satisfy these queries.

Indexing techniques is based on the idea that by interleaving auxiliary information with data frames on broadcast channels, mobile computers are able to predict the arrival of requested data frames. Hence, by staying in doze mode most of the time and only waking up to receive the data when they arrive (also called *selected tune-in*), power consumption of the mobile clients can be conserved. Since only the qualified data are retrieved into the mobile clients, CPU time and cache space are also saved.

In addition to indexing, broadcast data organization and data access methods are two other major factors determining the access time and the tune-in time of mobile computers. In the following, we first discuss the assumptions about scheduling and clustering we make in this paper. The intention is to fix the related parameters about data organization on broadcast channel, so that we can focus on the indexing techniques and access methods in the next section.

Data scheduling determines the contents of the broadcast cycle and the broadcast frequency of the data frames [1, 9, 10] (i.e., the hot data set for broadcast and the relative broadcast frequency of each frame). A simple broadcast schedule, called *flat broadcast*, is to broadcast each data frame once in every cycle [1]. Since the client access pattern for an attribute is usually skewed (i.e., some data are accessed more frequently than others), another scheduling method, called *broadcast disks*, was proposed in [1]. Broadcast disks broadcast important data more often than the other data to reduce the average access time for queries based on that attribute value. However, broadcast disks increase the length of a broadcast cycle and the client has to spend more time to retrieve the less commonly requested data. For multi-attribute data records, a cycle can be organized in broadcast disks based on one of attributes. Due to data frame duplication, queries other than that attribute may take longer to answer. Therefore, broadcast disks are not efficient for data records with multiple attributes. In this paper we assume that flat broadcast scheduling is used.

Data clustering refers to arranging the data items with the same value for a specific attribute to appear consecutively in a broadcast cycle [5, 6, 7]. By monitoring the arrival of the first data item with the desired attribute value, the client can retrieve all of the successive data items with the same attribute value. However, similar to data scheduling, a broadcast cycle can only be clustered based on one attribute. We call this attribute the *clustered attribute* and the other attributes the *non-clustered attributes*. Although the other attributes are non-clustered in the cycle, a second attribute can be chosen to cluster the data items within a data cluster of the clustered attribute. In turns, a third attribute can be chosen to cluster the data items within a data cluster of the second attribute.

For each non-clustered attribute, the broadcast cycle can be partitioned into a number of segments called *meta segments* [7], each of which holding a sequence of frames with non-decreasing (or non-increasing) values of that attribute. Thus, when we look at each individual meta segment, the data frames are clustered on that attribute and indexing techniques designed for clustered data broadcast can still be applied within the meta segment. To facilitate our study, we define the number of meta segments in the broadcast cycle for an attribute as the *scattering factor*[5] of the attribute. For multiple attributes, the broadcast cycle is partitioned into meta segments for each attribute in the order of decreasing access frequency. The scattering factors of an attribute increases as the importance of the attribute decreases. Organizing data broadcast with the clustering structure discussed above can improve retrieval efficiency. Thus, in this paper, we assume that the data frames within a broadcast cycle are partitioned into meta segments based on the first r attributes.

3 Multiple Attribute Indexing Techniques

In this section, we investigate the application of the index tree, the signature and the hybrid techniques to broadcast data with multiple attributes. The cost models of access time and tune-in time for the investigated indexing techniques are derived. The comparisons among these three techniques based on access time and tune-in time are presented in the next section. To save space, we do not repeat

[4]In our analysis, we assume that the size of a data frame is a multiple of the packet size.

[5]To simplify our discussion, we neglect the variance of the meta segment size.

the basic index tree, signature and hybrid techniques in the paper. Reader can refer to [5, 7, 8] for detail.

Before we discuss the multi-attribute indexing techniques and their associated access methods, we first describe the system parameters used in the study. We assume that there are m common attributes in each data frame and the attributes are sorted based on their access frequency. Let the ordered attributes be a_1,a_2,...,a_m and their probabilities contained in a query be p_1,p_2,...,p_m, where $p_i \geq p_{i+1}$ $(1 \leq i < m)$. a_1, called *major attribute*, is the most frequently accessed attribute and all other attributes are called *minor attributes*. The minor attribute a_m is the least accessed attribute. Indexes are built on multiple attributes (i.e. the frequently accessed attributes, a_1,a_2,...,a_r where $1 \leq r \leq m$). The rest of the system parameters are defined in Table 1.

Table 1. System Parameter

F	Information frame number in a broadcast cycle
P	Average packet number in an information frame
m	Attribute number in a frame
r	Indexed attribute number in a frame $(0 \leq r \leq m)$
q	Attribute number in a query $(1 \leq q \leq m)$
S_i	percentage of frames with required attribute value
p_i	Probability of queries based on a_i

A multi-attribute query usually contains multiple attributes and consists of many combinations of Boolean operators, such as conjunction ($\wedge$) and disjunction($\vee$). Since it is difficult to consider all combinations in a study, we choose the queries with either all conjunction or all disjunction operators as the representative multiple attribute queries for evaluation.

Let $Q\{a'_1, \ldots, a'_q\}$ denote a query with q attributes, where the attributes a'_i $(1 \leq i \leq q \leq m)$ are sorted according to the order they are indexed. Without loss of generality, we assume that q attributes are the first q frequently accessed attributes, a_1, ..., a_q. $Q\{a_1 \wedge \ldots \wedge a_q\}$ and $Q\{a_1 \vee \ldots \vee a_q\}$ are the only two query expressions. This is the minimal configuration to cover the common queries involved.

To provide the comparison baseline, we first introduce the non index approach (denoted *non-index*). Since every arriving frame must be retrieved into the client cache to check against the attribute values specified in the query, the tune-in time is very long and is equal to the access time. The estimated access time and tune-in time for $Q\{a_1 \wedge \ldots \wedge a_q\}$ is as follows.

$$\begin{aligned} A(a_1 \wedge \ldots \wedge a_q) &= T(a_1 \wedge \ldots \wedge a_q) \\ &= P \cdot F \cdot (1/2 + S_1) \end{aligned}$$

where a_1 is clustered. A and T, hereafter, are used to denote the access time and tune-in time of the queries. For $Q\{a_1 \vee \ldots \vee a_q\}$, the client must scan the entire broadcast cycle to retrieve the desired frames:

$$A(a_1 \vee \ldots \vee a_q) = T(a_1 \vee \ldots \vee a_q) \quad = \quad P \cdot F$$

3.1 The Index Tree Method

For a multiple-attribute data set, we can build index trees for each attribute separately. Several factors influence the order of attribute indexes in a cycle. To reduce index tree overhead for multiple attributes, the index tree should be built in the order that the attribute with the highest selectivity is indexed first. In this way, the clustered data frames can be received successively. On the other hand, to achieve the best average system performance, index should be built in the order of access frequency such that the higher the access frequency the higher the priority in choosing an attribute for indexing. From the index point of view, an index built on an attribute with a lower selectivity is better, because low selectivity attribute has high partitioning power and consequently high filtering capability. Obviously, the factors of selectivity and access probability have contradicting effects. We have to balance the access probability and the attribute selectivity to determine the order of attributes to be indexed.

Based on research results in [5, 7], data access is more efficient for clustered attribute than for non-clustered attribute. For example, both access time and tune-in time are improved by clustering. In addition, index built on clustered attribute requires less overhead than that built on non-clustered attributes. Since data frames can be clustered on at most one attribute, we choose to cluster data frames in a broadcast cycle based on major attribute (i.e., a_1). Thus, the minor attributes are non-clustered. Let $I = \{a_1, \ldots, a_r\}$ be the set of indexed attributes. The broadcast cycle is partitioned separately for each attribute in I ordered from a_1 to a_r. Hence, the scattering factor for a_i, denoted as M_i, increases with i. The index tree overhead of a_i depends on M_i. Hence, the smaller the subscript of the attribute in I, the lower the index overhead.

Since the broadcast channel is a linear medium and the index information increases the length of the cycle, index for an attribute influences the performance of queries based on not only that attribute but of those based on other attributes as well. When M_i (where $i > r$) is too large, constructing a distributed indexing tree on a_i does not help improve tune-in time much but results in a high index overhead. For example, the broadcast cycle can become very long and the access time for queries based on any attribute increases considerably. In this case, non-index is a better approach.

The value of M_j depends on S_i and the inter-relation among a_i, where $1 \leq i \leq j$. However, the dependency among the attributes is difficult to determine because it is most likely semantic-based and it may vary from application to application. To simplify the cost model, we assume that the attributes are *random and independent*. Based on this assumption, a simple estimate on M_i is $M_i = 1/\prod_{j=1}^{(i-1)} S_j$.

Table 2. Parameter for Index Tree

h_i	Height of the whole index tree built for a_i
t_i	Height of the replicated tree part for a_i
N	Number of packets in an index tree node
n	Search-key plus pointer number a node holds

Table 2 describes the parameter settings for index tree cost models. In addition, we let $X[h]$ and $X[t]$ be the total number of nodes of the full index tree and the replicated part of the index tree, respectively. Also, the number of nodes in the i-th level of the index tree is denoted as $L[i]$. For multi-attribute indexing, extra index information increases the cycle length to $P \cdot F + N \cdot \sum_{j=1}^{r} E_j$ packets, where E_j is the total number of index nodes allocated to a_j. For a_j ($j \neq 1$), there is an index tree constructed for each meta segment. Therefore, there are M_j index trees corresponding to a_j. Since each index tree has $X[h_j] + L[t_j + 1] - 1$ index nodes allocated for a_j within a meta segment, the total overhead incurred in building index trees for a_j in a broadcast cycle is $E_j = M_j \cdot (X[h_j] + L[t_j + 1] - 1)$ nodes.

To simplify the cost models, we average the index tree overhead to each data frame so that the size of a frame is considered to consist of a data part and an index overhead part. Of course, the actual index tree overhead for each data frame is different, but from the statistical point of view we can assume that all data frames have the same average index tree overhead which is $TREE = N \cdot \sum_{j=1}^{r} E_j / F$. The replicated index tree part for a_i is broadcast every $1/L[t_i + 1]$ of each meta segment. The data frames are divided into $M_i \cdot L[t_i + 1]$ data segments with replicated index nodes at the beginning of each data segment. The length of each data segment is $(P \cdot F + N \cdot \sum_{j=1}^{r} E_j)/(M_i \cdot L[t_i + 1])$. Hence, the *initial probe time* for the index tree built for a_i can be estimated as follows [7],

$$PROBE^{idx} = \frac{P \cdot F + N \cdot \sum_{j=1}^{r} E_j}{2 \cdot M_i \cdot L[t_i + 1]}$$

In the following paragraph, for conjunction operators ($\wedge$) and disjunction operators ($\vee$), their access methods are described and then the cost formulae are derived. The general access method for $Q\{a_1 \wedge \cdots \wedge a_q\}$ is:

Search: FOR each query attribute a_i DO

- Initial probe for the index tree built based on a_i within the data segment qualified for a_{i-1}.
- Search the index tree built based on a_i: the client follows a list of pointers to find out the arrival time of the desired data frame.

Retrieval: Scan the current meta segment for the desired data frames. At the end of the meta segment a jump is made to the other meta segments where the client should examine whether that meta segment is qualified for the attributes $a_1, \ldots, a_q$ contained in the query.

The access time for $Q\{a_1 \wedge \ldots \wedge a_q\}$ is upper bounded by:

$$\begin{aligned}
&A^{idx}(a_1 \wedge \ldots \wedge a_q) \\
&= \text{initial probe time for major attribute index} + \\
&\quad \text{waiting time for first desired data segment} + \\
&\quad \text{access time for desired data within the segment} \\
&= PROBE^{idx} + \frac{P \cdot F + N \cdot \sum_{j=1}^{r} E_j}{2} + P \cdot S_1 \cdot F
\end{aligned}$$

The tune-in time for $Q\{a_1 \wedge \ldots \wedge a_q\}$ depends on the number of levels in the index trees for a_i, $1 \leq i \leq r$ and the selectivity of the conjunction query. The initial probe of the client is to find the occurrence of the control index for a_1. The control index directs the client to the required higher level index tree nodes for a_1. A search in the index tree is conducted and the last pointer in the index tree points to the occurrence of the desired data segment. Within that data segment, a probe for index tree for a_2 is conducted. Once the index tree for a_2 is obtained, a search in the index tree is carried out to find the data segment which matches a_2. The filtering process continues with all other attributes involved with the query. Finally, the data frames of interests are received. The tune-in time is obtained as follows:

$$\begin{aligned}
&T^{idx}(a_1 \wedge \ldots \wedge a_q) \\
&= (1 + \sum_{i=1}^{l} h_i) \cdot N + (1 + F \cdot \prod_{i=1}^{l} S_i) \cdot P
\end{aligned}$$

where $l = min(r, q)$ and $\prod_{i=1}^{l} S_i$ is the number of matched items for $Q\{a_1 \wedge \ldots \wedge a_l\}$.

For $Q\{a_1 \vee \ldots \vee a_q\}$, the client monitors the channel for every index tree built on the attributes in the query as if the client queries each of the q attributes simultaneously, but within one scan. The following is the access method:

Initial probe: For any index trees built on $a_1, \ldots, a_q$, the client tunes into the broadcast channel and determines when the next index tree nodes for the data frames are broadcast.

Search: For each index tree built on $a_1, \ldots, a_q$, the client follows a list of pointers to find out the arrival time of the desired data frame.

Retrieval: The client tunes into the broadcast channel to download all the qualified frames.

The access time for $Q\{a_1 \vee \ldots \vee a_q\}$ is,

$$\begin{aligned}
&A^{idx}(a_1 \vee \ldots \vee a_q) \\
&= \text{initial probe time} + \\
&\quad \text{waiting time for retrieving all desired frames} \\
&= PROBE_i + P \cdot F + N \cdot \sum_{j=1}^{r} E_j
\end{aligned}$$

and the tune-in time for $Q\{a_1 \vee \ldots \vee a_q\}$ is as follows:

if $r < q$

$$T^{idx}(a_1 \vee \ldots \vee a_q) \quad = \quad r \cdot N + P \cdot F$$

else (i.e., $q \leq r$)

$$T^{idx}(a_1 \vee \ldots \vee a_q) \leq (1 + \sum_{i=1}^{q} h_i) \cdot N + P \cdot \sum_{i=1}^{q} M_i + F \cdot P \cdot \sum_{i=1}^{q} (-1)^{i+1} S^i \left(\begin{array}{c} q \\ i \end{array} \right)$$

where the number of true matches is no greater than $P \cdot F \cdot \sum_{i=1}^{q} (-1)^{i+1} S^i \left(\begin{array}{c} q \\ i \end{array} \right)$ and $S = max\{S_i : 1 \leq i \leq q\}$.

3.2 Signature for Multiple Attributes

Table 3. Parameters for Signature Scheme

P_i	a frame false drop probability for a_i
P_f^S	a simple signature false drop probability
P_f^I	an integrated sig. false drop probability
k	number of frames indexed by an integrated sig.
l_i	average number of true matches in a frame group
p	number of bits in a packet
R_S	number of packets a simple signature takes
R_I	number of packets of an integrated sig. takes
s	number of bit strings superimposed into a sig.

For the index tree method, there is an index tree for each indexed attribute. Therefore, the index overhead is directly proportional to the number of indexed attributes. For the signature method, one signature can provide index information to all attributes. The index overhead is much less influenced by the number of attributes and the access method is simpler than the index tree method.

Table 3 defines the parameters for signature cost models. For multi-attribute indexing, the multi-level signature method is the best approach [8, 5] and is used in this paper for data indexing. We assume that every attribute is indexed in the signature (i.e., $r = m$). The the signature false drop probability for either simple signature or integrated signature can be estimated as follows [8]:

LEMMA 1 (Optimal false drop probability) *Given the size of a signature, R, hereafter denotes either R_S or R_I, and the number of bit strings superimposed into the signature, s, the false drop probability for the signature is:* $P_f^S = P_f^I = 2^{-R \cdot (p \cdot ln2)/s}$

We define P_i as the false drop probability of a data frame for queries based on a_i. P_i has two components P_f^I and P_f^S, which, respectively, reflects the false drops introduced by the integrated signature and the simple signature. Based on [8], we derive the estimate of P_i as follow:

$$P_i \quad = \quad ((1/k - \lceil S_i/l_i \rceil) \cdot P_f^I \cdot (k \cdot R_S/P + k \cdot P_f^S) + \lceil S_i/l_i \rceil \cdot (k - l_i) \cdot P_f^S)/(1 - S_i) \qquad (1)$$

To simplify our discussion, we assume that frames with the same attribute value for a_i ($1 \leq i \leq m$) are evenly distributed in each meta segment. Consequently, the average number of frames with the same value for a_i in each meta segment is $\lceil F \cdot S_i/M_i \rceil$. Let k be the number of data frames grouped in an integrated signature. The number of distinct attribute values in k data frames can be estimated as $\lceil k/\lceil F \cdot S_i/M_i \rceil \rceil$. For frames in a meta segment partitioned for a_i, the average number of qualified frames corresponding to a matched integrated signature, called *locality* of true matches l_i ($1 \leq l_i \leq k$ for $1 \leq i \leq r$), can be estimated as, $l_i = k/\lceil k/\lceil F \cdot S_i/M_i \rceil \rceil$.

For randomly distributed frames, the locality of true matches corresponding to a_i ($r < i \leq m$), l_i, equals to 1. Since each data frame contains m attribute values[6] corresponding to $a_1, ..., a_m$, the expected distinct superimposed bit strings, s, for an integrated signature can be estimated as, $s = \sum_{i=1}^{m} \lceil k/l_i \rceil$, where $\lceil k/l_i \rceil$ is the expected number of distinct bit strings superimposed for a_i.

To simplify the estimates of the tune-in time, the initial probe for the integrated signature is a true match. The average waiting time for retrieving one data frame is $SIG_S + SIG_I + P$, where SIG_S and SIG_I, the average simple and integrated signature overheads for a data frame, can be calculated as $SIG_S = R_S$ and $SIG_I = R_I/k$.

Generally speaking the length of a signature (R_S or R_I) is very small compared to a data frame (i.e., $P >> R$). Otherwise, both the access time and the tune-in time would be large. Therefore, the time for the client to filter out the partial signature and the partial data frame, called initial probe time, can be be approximated as half of the data frame size, $PROBE = P/2$. The tune-in time in the initial probe period is the expected sum of the time when the client is active for filtering a partial simple signature, a partial integrated signature, and a partial data frame. For similar reasons, the tune-in time for filtering a partial signature is negligible and the initial tune-in time can be approximated as $P/2$.

One major difference between the index tree and the signature method is that an index tree node is good for only one indexed attribute, while a signature contains the information for all indexed attributes. Therefore, the client first retrieves the signature into the cache, by scanning that signature, the client knows whether the associated data frame matches more than one attribute. Only when the signature

[6]We assume that the hashed attribute values for different attributes are not the same (i.e., independent event). Therefore, the number of distinct bit strings superimposed into a simple signature for m attributes is m.

matches all query attributes, is the corresponding data frame fetched into the cache.

We develop the access method for queries $Q\{a_1 \wedge \ldots \wedge a_q\}$ as follows. Signature match for a_1 is conducted first. Matched signatures (either true match or false drops) are subjected to further signature matches based on the other query attribute values.

Initial probe: The client tunes into the broadcast channel for the first received signature

Filtering: For each signature received in a broadcast cycle do the following:

for each query attribute a_i,

IF the signature does not match the query signature based on attribute a_i

THEN repeat the filtering step

Retrieval and checking: The client retrieves frames corresponding to the matched signature for further checking to eliminate false drops. Repeat filtering step.

For $Q\{a_1 \wedge \ldots \wedge a_q\}$, the access time is:

$$
\begin{aligned}
&A^{sig}(a_1 \wedge \ldots \wedge a_q)\\
&= \text{initial probe time} +\\
&\quad \text{filtering time for the first desired frame} +\\
&\quad \text{retrieving time for all the desired frames}\\
&= PROBE + CYCLE/2 + S_1 \cdot F\\
&= P/2 + F \cdot (SIG_I + SIG_S + P)/2 + P \cdot S_1 \cdot F
\end{aligned}
$$

The tune-in time for $Q\{a_1 \wedge \ldots \wedge a_q\}$ is:

$$
\begin{aligned}
&T^{sig}(a_1 \wedge \ldots \wedge a_q)\\
&= \text{the tune-in time in the initial probe period} +\\
&\quad \text{true match data frames in the broadcast} +\\
&\quad \text{every integrated signature in half the cycle} +\\
&\quad \text{simple sig associated with qualified integrated sig} +\\
&\quad \text{false drop data frames in half the cycle}\\
&= \frac{P}{2} + P \cdot F \cdot \prod_{i=1}^{q} S_i + \frac{F}{2} \cdot (R_I/k + \lceil S_1/l_1 \rceil k \cdot R_S)\\
&\quad + \text{false drop data frames in half the broadcast}\\
&\leq \frac{P}{2} + P \cdot F \cdot \prod_{i=1}^{q} S_i + \frac{F}{2} \cdot (R_I/k + \lceil S_1/l_1 \rceil k \cdot R_S)\\
&\quad + P \cdot F \cdot \sum_{i=1}^{q} (q \cdot S^{i-1} \cdot D^{q+1-i})/2
\end{aligned}
$$

where the number of true matches is estimated as $F \cdot \prod_{i=1}^{q} S_i$. $S = max\{S_i : 1 \leq i \leq q\}$. P_i is given in Equation (1). $P_i \cdot (1 - S_i)$ is the false drop probability for a_i. $D = max\{P_i \cdot (1 - S_i) : 1 \leq i \leq q\}$.

For a query $Q\{a_1 \vee \ldots \vee a_q\}$, the access method is:

Initial probe: The client tunes into the broadcast channel for the frame signature in a broadcast cycle.

Filtering: The client tunes into retrieve the signatures arriving and matches the frame signatures with the signatures for any attribute values in $a_1, \cdots, a_q$.

Retrieval and checking: For the matched signatures, the client tunes into get the corresponding data frames from the channel for further checking to eliminate false drops.

The access time for $Q\{a_1 \vee \ldots \vee a_q\}$ is:

$$
\begin{aligned}
&A^{sig}(a_1 \vee \ldots \vee a_q)\\
&= \text{initial probe time} + \text{a broadcast cycle}\\
&= P/2 + F \cdot (SIG_I + SIG_S + P)
\end{aligned}
$$

and the tune-in time for $Q\{a_1 \vee \ldots \vee a_q\}$ is upper bounded by:

$$
\begin{aligned}
&T^{sig}(a_1 \vee \ldots \vee a_q)\\
&= \text{tune-in time in the initial probe period} +\\
&\quad \text{true match data frames in the cycle} +\\
&\quad \text{every integrated signature in the cycle} +\\
&\quad \text{simple sig associated with qualified integrated sig} +\\
&\quad \text{false drop data frames in the cycle}\\
&\leq P/2 + F \cdot P \cdot \sum_{i=1}^{q} (-1)^{i+1} S^i \begin{pmatrix} q \\ i \end{pmatrix} +\\
&\quad F \cdot (R_I/k + \lceil S_1/l_1 \rceil k \cdot R_S) +\\
&\quad \text{false drop data frames in the whole broadcast}\\
&\leq P/2 + F \cdot P \cdot \sum_{i=1}^{q} (-1)^{i+1} S^i \begin{pmatrix} q \\ i \end{pmatrix} +\\
&\quad F \cdot (R_I/k + \lceil S_1/l_1 \rceil k \cdot R_S) +\\
&\quad P \cdot F \cdot \sum_{i=1}^{q} ((1 - S_i) \cdot P_i)
\end{aligned}
$$

where the number of true matches is no greater than $P \cdot F \cdot \sum_{i=1}^{q} (-1)^{i+1} S^i \begin{pmatrix} q \\ i \end{pmatrix}$ and $S = max\{S_i : 1 \leq i \leq q\}$.

3.3 The Hybrid Methods

The hybrid index consists of two parts: sparse index tree and signature. The sparse tree is built on the major attribute and used for global filtering. The multi-level signature is used to carry out the local filtering based on attribute values specified in the queries. In this way, the index tree part helps improve tune-in time of the client. The average waiting time for retrieving one data frame from the broadcast cycle can be expressed as:

$$TREE{+}SIG{+}P = N \cdot (X[t_1{+}1]{-}1)/F + R \cdot (1{+}1/k) + P$$

where $TREE$ and SIG respectively are the index overheads of the index tree part and the signature part of a frame. The average number of data frames in one data block, $F[B]$, can be calculated in a similar way as in the index tree method, which is $F/L[t_1+1]$. Thus, the overheads of the index tree and the signatures in a data block are $F[B] \cdot TREE$ and $F[B] \cdot SIG$, respectively. Hence, the average initial probe time for the signature and the length of a cycle is given by:

$$\begin{aligned} PROBE^{hyb} &= (TREE + SIG + P)/2 \\ CYCLE^{hyb} &= (TREE + SIG + P) \cdot F \end{aligned}$$

The access method for a query $Q\{a_1 \wedge \ldots \wedge a_q\}$ is as follows:

If a_1 is the major attribute

- based on the single index tree access protocol to retrieve the frame block which contains all the frames satisfying the value for a_1
- successive signature matches based on $\{a_2, \ldots, a_q\}$ are conducted to filter out the desired frames

Else based on the signature access method to retrieve the qualified frames based on $\{a_1, \ldots, a_q\}$. The sparse index tree is skipped.

Therefore, the access time for $Q\{a_1 \wedge \ldots \wedge a_q\}$ is:

$$\begin{aligned} & A^{hyb}(a_1 \wedge \ldots \wedge a_q) \\ &= PROBE^{hyb} + CYCLE^{hyb}/2 + F \cdot S_1 \cdot P \\ &= (TREE + SIG + P) \cdot (1+F)/2 + F \cdot S_1 \cdot P \end{aligned}$$

The tune-in time primarily depends on the initial probe of the client to determine the next occurrence of the sparse index, the access time for the index tree part which equals to the number of levels t_1 of the sparse index tree, the tune-in time for the data block B, and the selectivity of the query $Q\{a_1 \wedge \ldots \wedge a_q\}$. Therefore, it is upper bounded by:

$$\begin{aligned} & T^{hyb}(a_1 \wedge \ldots \wedge a_q) \\ &= (1+t_1) \cdot N + T(B) + (1 + F \cdot \prod_{j=1}^{q} S_j) \cdot P \end{aligned}$$

where $T(B)$ is the tune-in time for filtering the data block B with the signature method, which can be estimated as:

$$\begin{aligned} T(B) = & \text{ every sig in the data block B} + \\ & \text{false drop data frames in data block B} \quad (2) \end{aligned}$$

For $Q\{a_1 \vee \ldots \vee a_q\}$, the sparse index tree part does not help the retrieval of multiple attributes and the access method is similar to the multi-level signature. Thus, the access time for $Q\{a_1 \vee \ldots \vee a_q\}$ is:

$$\begin{aligned} A^{hyb}(a_1 \vee \ldots \vee a_q) &= PROBE^{hyb} + CYCLE^{hyb} \\ &= (TREE + SIG + P) \cdot (1/2 + F) \end{aligned}$$

In order to skip each of the index tree, we assume that the client needs to retrieve one tree node to get the control information such as the size of the sparse index tree and the size of the data block. Therefore, the tune-in time for $Q\{a_1 \vee \ldots \vee a_q\}$ is:

$$T^{hyb}(a_1 \vee \ldots \vee a_q) = N + T^{sig}(a_1 \vee \ldots \vee a_q)$$

where, $T^{sig}(a_1 \vee \ldots \vee a_q)$ represents the tune-in time for the corresponding signature scheme used (i.e., multi-level signature).

4 Evaluation of Query Performance

In this section, we make analytical comparisons of the performance for the multi-attribute indexes discussed in the previous sections. The comparisons are based on the formulae developed in Section 3. The access time and the tune-in time for the index tree, signature, and hybrid methods, are evaluated. The study addresses two kinds of Boolean query expressions: conjunction and disjunction.

Table 4 lists the parameter settings used in the comparisons. We assume that the broadcast data file contains 10 attributes (i.e., $a_1, \cdots, a_{10}$). For multiple attribute queries, three most frequently accessed attributes (i.e., a_i, $1 \leq i \leq 3$) are included in the queries. Flat broadcast schedule is used for data broadcasting. A data frame consists of 1000 packets and a packet contains 64 bits. For the index tree, each tree node, which consists of 100 packets, contains 128 search-key plus pointers. For the signature method, the frame group size is 4. To simplify the discussion, the selectivity of all attributes is set to the same value (i.e., 0.01).

For the index tree method, the index overhead is proportional to the number of attributes. For the signature and the hybrid methods, the index overhead is much less influenced by the number of attributes. Therefore, among the 10 attributes, the first six frequently accessed attributes are indexed in the index tree method and all attributes are indexed for the signature and the hybrid methods. The index tree is balanced (all leaves are on the same level), and each node has the same number of children.

To show the experimental results in a clear way, access time overhead is computed with respect to non-index approach. ON the other hand, tune-in time for non-index approach is included in the figures as a baseline for tune-in time comparisons. We first investigate multi-attribute conjunction queries. Then, we evaluate the multi-attribute disjunction queries.

4.1 Conjunction Queries

In this set of experiments, we assume the conjunction queries are used and the queries contain three most frequently accessed attributes a_1, a_2, and a_3. Figures 1

Table 4. Parameters of the cost models

$F = 10^4 \sim 10^6$	$P = 10^3$	$S = 0.01$	$p = 64$	$m = 10$
$n = 128$	$N = 100$	$k = 4$	$r = 6$	$q = 3$

and 2 illustrate the access time and the tune-in time for $Q\{a_1 \wedge a_2 \wedge a_3\}$ when the cycle length is varied. Obviously, the access time of the index tree method is much worse than the other two index methods. This is because the index tree overhead for non-clustered attributes (i.e., a_i, $i > 1$) is very large. Based on the construction of the multiple-attribute index tree, the less the attribute accessed, the higher its index overhead. On the other hand, for the signature and the hybrid methods, the index overhead is almost independent of the number of attributes and is much less than that of the index tree. The sparse index tree in the hybrid method has a very low overhead. As a result, the access time for the signature and the hybrid methods is similar (Figure 1).

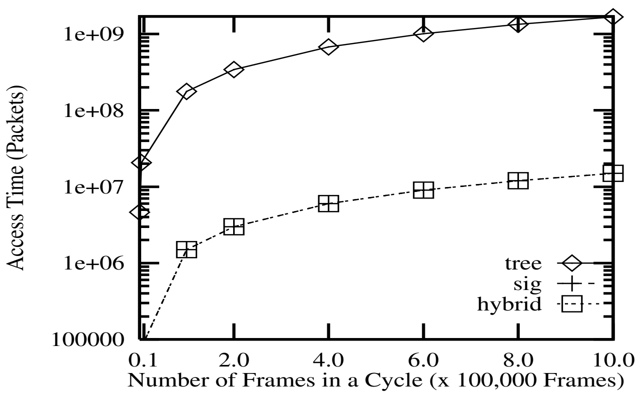

Figure 1. Access Time for Conjunction Query

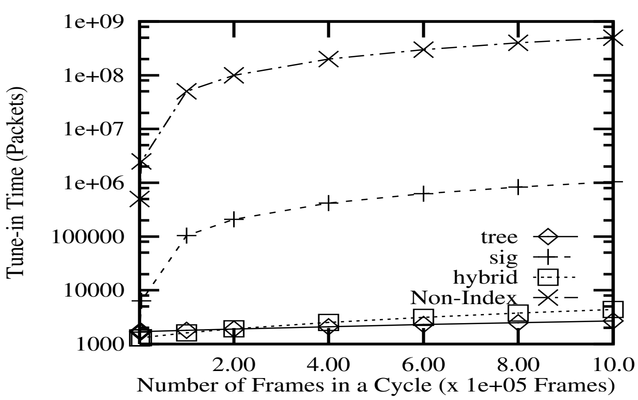

Figure 2. Tune-in Time for Conjunction Query

When the tune-in time is considered (refer to Figure 2), the signature method performs worse than the other two index methods and the index tree shows a marginally better performance than the hybrid method. This is because the tune-in time for signature false drop is greater than that for searching in the index tree. All three index methods give a considerably better performance than the non-index approach.

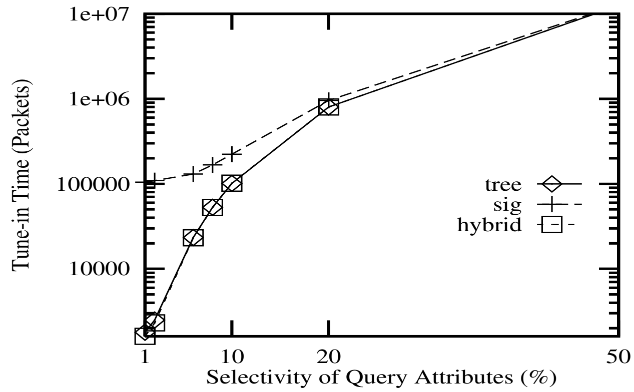

Figure 3. Tune-in Time vs Query Selectivity

Figure 3 shows the tune-in time as a function of the query selectivity for $Q\{a_1 \wedge a_2 \wedge a_3\}$ where the cycle length is 10^5. For any query selectivity, the index tree always has the similar performance as the hybrid method. For small query selectivity, the signature does worse than the other two. However, as the query selectivity is increased, all three methods give similar performance, because for large selectivity the tune-in time for the retrieval of the qualified data dominates the tune-in time of the queries.

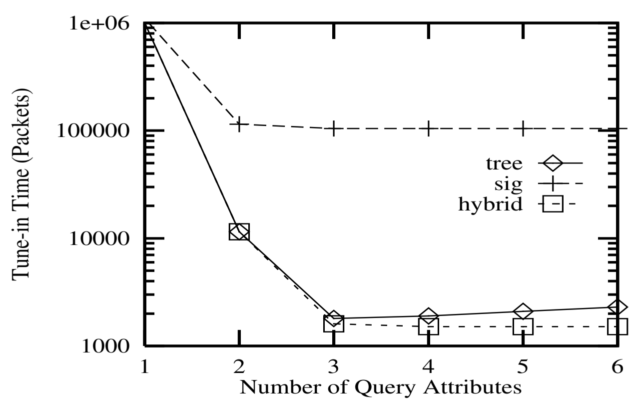

Figure 4. Tune-in Time vs Query Number

Intuitively, a conjunction query with more attributes in the predicates should result in a smaller set of qualified data items. This is demonstrated in Figure 4 where the number of attributes in a query is varied while the cycle length is fixed to 10^5. As the number of attributes in a query increases, the tune-in time for all index methods decreases as a result of the decrease in the number of qualified data. The tune-in time for the signature method is similar to that of the index tree and the hybrid methods only for single attribute query. For queries with more than one attribute, the signature has an obvious poor performance than the other two. That is because the tune-in time for eliminating false matches may become too significant even though the tune-in time for receving qualified frames decreases. For queries with few attributes (i.e., less than three), the index tree and the hybrid methods have similar performance. However as the number of attributes in a query is increased, the hybrid method outperforms the index tree method. Due to the tune-in time for each index trees, the curve representing the index tree method goes up a little bit after the number of attributes is greater than three.

4.2 Disjunction Queries

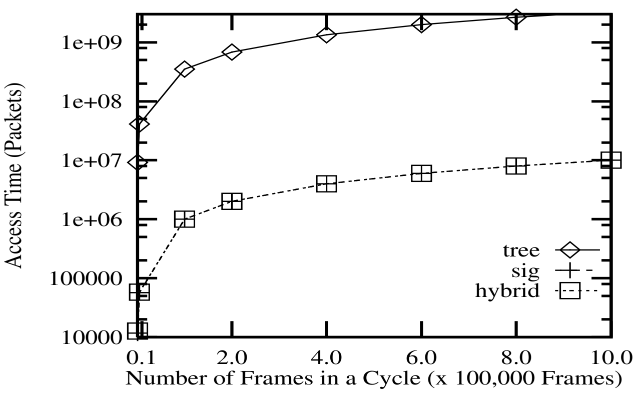

Figure 5. Access Time for Disjunction Query

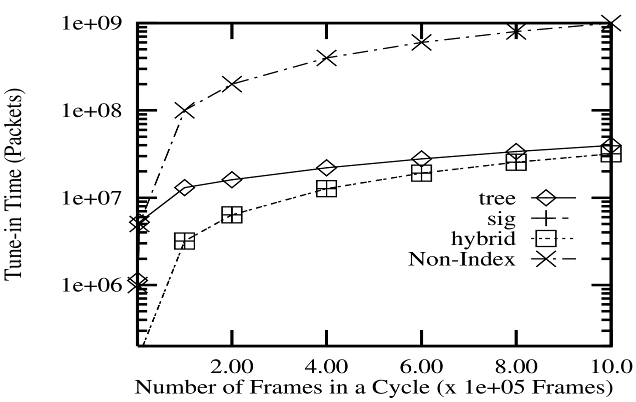

Figure 6. Tune-in Time for Disjunction Query

This section investigates the performance of disjunction queries. In Figures 5 and 6, the access time and the tune-in time for $Q\{a_1 \vee a_2 \vee a_3\}$ are shown. Similar to conjunction queries, the index tree method has the worst access time among the three index methods and the other two have similar access time. Figure 6 shows all three index methods have much better tune-in time performance than the non-index method. The hybrid and the signature have similar tune-in time performance which is better than that of the index tree.

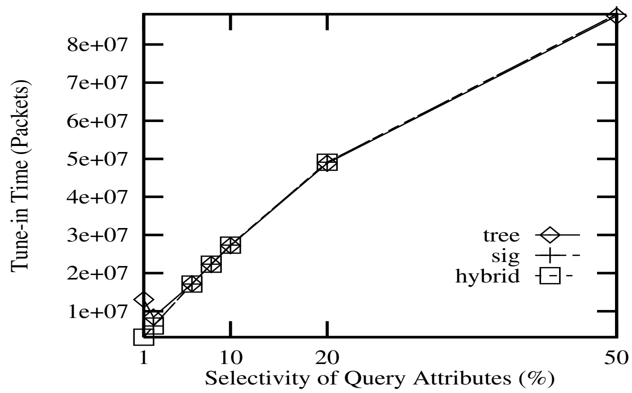

Figure 7. Tune-in Time vs Query Selectivity

The query selectivity is varied in Figure 7 while other parameters are fixed. The tune-in time for the signature and the hybrid methods is proportional to the selecrivity of queries, while their performance are about the same. For the index tree method, the tune-in time is similar to that of other index methods when query selectivity is greater 6%. It has an odd performance when query selectivity is less than 6%. For example, the tune-in time first decreases and then increases. This probably is due to two contradicting factors: 1) a larger selectivity may reduce the size of an index tree and thus introduce lower index overhead; 2) a larger selectivity may result in more qualified frames and thus higher tune-in time for retrieving these frames. After the query selectivity is greater than 6%, the retrieval factor dominates the tune-in time of the query.

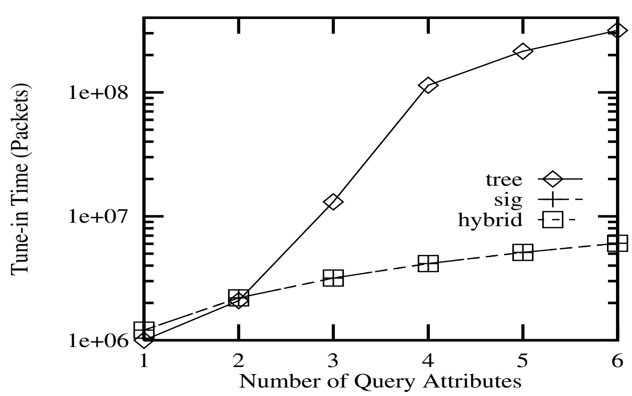

Figure 8. Tune-in Time vs Query Number

In contrast to conjunction queries, the more attributes involved in a disjunction query may result in the more qualified data retrieved. Thus, Figure 8 shows a reversed performance compared to Figure 4. For queries with many attributes, the index tree does worse than the other two. This is due to the retrieval of each meta segment in each index tree. The number of meta segments retrieved increases rapidly when the number of attributes involved in a query increases.

5 Conclusions and Future Work

In this paper, we examined the problem of multi-attribute queries for wireless data broadcast. We studied the index tree, the signature, and the hybrid index methods for multi-attribute queries. To facilitate our evaluation, we assumed a query involves all of the attributes either conjunctively or disjunctively.

We provided cost models of the three multi-attribute indexing methods for the estimates of access time and tune-in time and compared their performance. We found that the index tree method, while performing well for single-attribute queries results in large access time overhead. This is due to the creation and replication of index trees for the indexed attributes. Moreover, the index method has an update constraint, i.e., updates of a data frame are not reflected until the next broadcast cycle. The comparisons revealed that all index methods introduced certain access time overhead while improving power conservation by reducing their tune-in time. We conclude from our study that the hybrid is the best choice for multi-attribute queries due to its good access time and tune-in time. The signature method has the similar performance as the hybrid method except conjunction queries. The index tree method is poor in access time for any types of multi-attribute queries. It gives similar tune-in time as the hybrid method for the conjunction queries.

In this paper, we only studied two simple multi-attribute queries: conjunction and disjunction. In the future, we will investigate more complicated ones such as, mixed conjunction and disjunction queries, join queries, and range queries etc. In a mobile computing environment, index information is frequently accessed data. Caching this index information in the clients may reduce both access time and tune-in time considerably. One future research would incorporate the index schemes with data caching algorithms to achieve a better system performance.

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 199–210, San Jose, California, May 1995.

[2] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 43–54, Dallas, TX, October 1998.

[3] D. Aksoy and M. Franklin. Scheduling for large-scale on-demand data broadcasting. In *Proceedings of IEEE INFOCOM'98*, San Francisco, CA, March 1998.

[4] 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 Eighth ACM International Conference on Information and Knowledge Management*, pages 351–358, Kansas City, Missouri, November 1999.

[5] Q. L. Hu, W.-C. Lee, and D. L. Lee. A hybrid index technique for power efficient data broadcast. *Distributed and Parallel Databases Journal*, to appear.

[6] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Power efficienct filtering of data on air. In *Proceedings of the International Conference on Extending Database Technology*, pages 245–258, 1994.

[7] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on the air - organization and access. *IEEE Transactions of Data and Knowledge Engineering*, June 1997.

[8] W.-C. Lee and D. L. Lee. Using signature techniques for information filtering in wireless and mobile environments. *Distributed and Parallel Databases Journal: Special Issue on Database and Mobile Computing*, 4(3):205–227, July 1996.

[9] K. Stathatos, N. Roussopoulos, and J. S. Baras. Adaptive data broadcast in hybrid networks. In *Proceedings of the 23rd VLDB Conference*, pages 326–335, Athens, Greece, August 1997.

[10] C.J. Su and L. Tassiulas. Broadcast scheduling for information distribution. In *Proceedings of IEEE INFOCOM'97*, Kobe, Japan, April 1997.