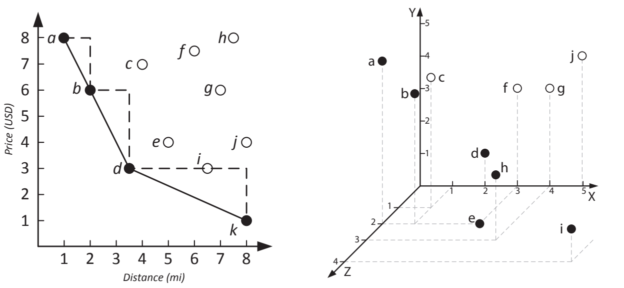
Efficient Evaluation of Skyline Queries in Wireless Data Broadcast Environments

Chih-Jye Wang and Austin L. Hudson
Department of Computer Science and Software Engineering, Auburn University
Auburn, AL, USA
{wangchj, alh0010}@auburn.edu

***Abstract*— Skyline is a query operation that retrieves data items that are not dominated by another item with respect to multiple attributes. Skyline has broad application and relevance due to multi-criteria benefit, but very little study have been done on skyline in broadcast data model and no study has been done on flexible skyline query evaluation in this model that can handle combination of minimum and maximum attributes. In this paper, we propose depth-first distributed index (DFDI) and skyline algorithms that incorporate DFDI to efficiently evaluate skyline in broadcast environment.**

I. Introduction

Skyline is a query operation that retrieves data items which are considered "interesting objects" with respect to multiple attributes of the data set. For example, someone who is planning for an ocean-view vacation would be interested to find a list of hotels that are close to the ocean and at the same time not too expensive. A hypothetical data set of hotels is shown in Table I. The two relevant attributes in this case are *minimum distance* to the ocean and *minimum price*. Figure 1(a) shows the skyline points in solid dots as the operation finds hotel records that are successively further from the ocean, but the prices are the best for such distances. In the figure, hotel a is a skyline point because it is closest to the ocean, although it is not the lowest price. Hotels b and d are part of the skyline because each have the closest distance at its price. Hotel k is a skyline point because it is the cheapest of all hotels.



(a) 2-D skyline of hotels of (X = min, Y = min)

(b) 3-D skyline of stocks with attributes (X = min, Y = min, Z = max)

Fig. 1. Sample Skylines

Skyline has broad applications and relevance due to multi-criteria benefit. Skyline computation in realtime streaming systems with frequent data updates has been studied in [14] and [17]. For distributed web services, skyline query solutions have been proposed in [1]. Skyline query has also been applied in sensor networks in [16]. Meanwhile, the data broadcast is a scalable way to disseminate data (e.g., FM broadcasting). Unlike the on-demand model, such as most of the services on the Internet, the broadcast model can scale almost indefinitely. A challenge and drawback of this model is the forward-only access characteristic. While many studies have been done on different query types to support efficient query evaluation in broadcast environments [7], [11], [20], to the best of our knowledge, the work in [5] is the only study on skyline query processing in broadcast environments. Although [5] considers broadcast efficiency (more details in Section II-B), the proposed solution is unable to support all possible skyline types (i.e., combinations of min and max attributes) and the work did not address skyline of higher data dimension ($>$ 2 dimensions) as illustrated in figure 1(b). To address these challenges, we design a flexible broadcast index and a skyline evaluation algorithm that utilizes the index to efficiently evaluate skyline queries. Specifically, our contributions of this research are as follows:

- We design a flexible on air index that supports skyline query evaluation in data broadcast environments for arbitrary number of data dimension.
- We propose an efficient data broadcast skyline query algorithm that handles both min and max attributes.
- We evaluate the performance of the proposed algorithms through extensive experiments.

The rest of this paper is organized as follows. Section 2

TABLE I
Sample Data

(a) 2-d Data of Fig 1(a)

Hotel	Distance	Price
a	1	8
b	2	6
c	4	7
d	3.5	3
e	5	4
f	6	7.5
g	7	6
h	7.5	8
i	6.5	3
j	8	4

(b) 3-d data of Fig 1(b)

Symbol	Price	P/E	Yield
a	0	5	2
b	1	4	2
c	1	4	1
d	2	2	0
e	3	0	2
f	3	3	0
g	4	3	0
h	4	2	3
i	7	1	4
j	5	4	0

provides background knowledge of the skyline operator and wireless data broadcast. We introduce our index structures to facilitate broadcast skyline query in Section 3. In Section 4 we propose our pruning region based technique for skyline query evaluation. The experimental validation of our design is presented in Section 5. Section 6 surveys related works. We conclude the paper with a discussion of future work in Section 7.

II. Preliminaries

A. Skyline Query

Skyline query is an operation that finds all data records which are not dominated by any other data objects in a given data set. Skyline query is closely related to the maximal vector problem [3], [12]. Here, we briefly introduce these concepts.

Definition 1 Maximal Vector Given the set of vectors P, for all vectors $v \epsilon P$, there does not exist $u \epsilon P$ such that $u \geq v$, then v is a maximal vector.

$$MaxVect(P) = \{v | \forall v, u \epsilon P \nexists u \geq v\} \qquad (1)$$

The comparison operator $\geq$ used in Equation 1 is defined on vectors such that for two vectors u, $v \epsilon V$, $u \geq v$ if $u_i \geq v_i$ for i = 1 to n. Vice versa for the $\leq$ operator. In other words, a vector v is "greater than" the other vector u if and only if all elements of v is "greater than or equal to" all elements of u. The comparison operators for each element of the vectors are the natural ordering operators defined on each domain.

The only difference between the maximal vector operation defined in definition 1 and the skyline operation is that the skyline operator defines a set of *preference specifiers*, σ, which are to be either *MIN* or *MAX* for each attribute.

Definition 2 Skyline Operator Give a set of tuples, P, and an ordered set of preference specifiers, σ, the skyline operator is defined as follows:

$$Skyline(P, \sigma) = MaxVect(P) \qquad (2)$$

In definition 2, the dominance relationship is defined as follows:

Definition 3 Dominance Relationship Give two points p_1 and p_2 in P, p_1 dominates p_2, if and only if all elements of p_1 dominates or is equal to all elements of p_2 and at least one element is dominant. Dominance for each element depends on the preference specifier, σ_i, for that attribute. Given x and y from D_i, the domanance for the elements is defined as follows:

1) If σ_i = *MIN*, x **dominates** y if $x < y$.
2) If σ_i = *MAX*, x **dominates** y if $x > y$.

Corollary 1 The dominance relationship is transitive, non-reflexive, and non-symmetric.

Proof: All three properties can be proven based on the fact that the values of each attribute domain are ordered: an attribute x is better than y then, y is worse than x. The transitive property of the dominance relationship follows from Definition 3, if record A dominates B then all attributes of A are equal or better and there is at least one attribute that is better; therefore, B dominates C implies that all attributes of A is equal or better than C therefore, the relationship is transitive. The relationship is non-reflexive since if all attribute of A is equal or better than attribute of B then the ordering of the values of each attribute domain imply that B does not dominate A. Similarly, D does not dominate itself since the definition states that at least one must be better for a record to dominate another. ■

Following the non-reflexive property of Corollary 1, given two record A and B and $A = B$, if A is a skyline record, then B is also a skyline record.

Extension SQL syntax for skyline was defined by Börzsönyi, et al. in [2] as illustrated in Figure 2. The syntax defines an additional SKYLINE clause in SQL that specifies how the skyline operation should be performed. In the SKYLINE clause, relevant attributes (such as H.price and H.dist in the figure) can be listed. For each attribute, the syntax defines three attribute specifiers, MIN, MAX, and DIFF, that tells the operator how each attribute is to be handled. MIN and MAX denote that the values of the attribute should be minimized or maximized. DIFF denotes that the values should be different. The MIN and MAX specifiers are considered in this paper and in designing our solution. DIFF is not discussed in this paper.

```
SELECT H.name, H.price, H.distance
FROM H
SKYLINE OF H.price MIN,
H.dist MIN ORDER BY H.name
```

Fig. 2. Skyline SQL Clause.

Skyline constraint regions is introduced in [5]. Constraint region limits the skyline queries to a subset of the data set, instead of the entire data set. For example a constraint could be a limit of hotel price within \$100 - \$150 and the distance within 0 to 1 mile from the beach. Constraint region is a separate issue in skyline queries that can be easily satisfied with a filtering step to remove all tuples not in the region preceding the main skyline algorithm. In this paper, we do not consider constraint region, and assume the entire space of the data set as the search space.

B. Wireless Broadcast

A wireless broadcast environment consists of a broadcast channel, a broadcast station (or a server), and a number of mobile clients who are interested in the broadcast program from the station. The server is the originator of the *broadcast program* which contains relevant data records and pushes the data onto the channel. The mobile clients receive desired data by listening to the channel. Examples of similar systems are AM/FM radio and satellite television. The model is illustrated in Figure 3.

TABLE II
SUMMARY OF ANALYTICAL NOTATIONS

Notation	Description
n	Number of dimensions of skyline
m	Number of records in data set
D	A set of attribute domains $D = \{D_1, ..., D_n\}$
P	A set of records (a set of tuples)
p	A record (a n-tuple)
S	A set of skyline points
σ	A set of skyline preference specifier
R	A pruning region
B	A minimal bounding box (MBR)
E	An R-Tree index entry (MBR, time)
b	Branching factor of an index tree
h	Height of an index tree
L	Tree level (1 to $h-1$). $L = h + 1$
L_r	Levels of index tree replication
ρ	Index Percentage
α	Initial Index Prob
β	Tuning Time
γ	Access Time
ι	Space of index (bytes)
θ	Size of the data set (bytes)
α	Space of the entire program cycle (bytes) $\iota + \theta$

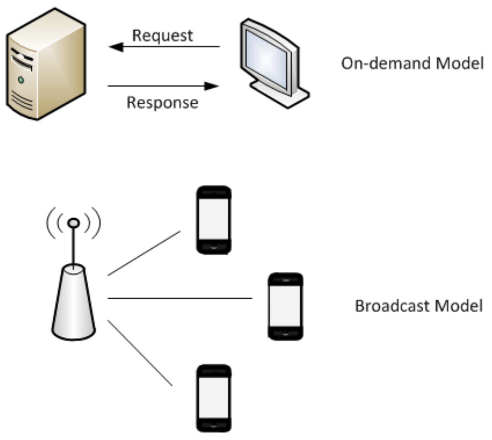


Fig. 3. On demand and broadcast models.

A complete dissemination of all the records in the data set from the server is a broadcast *cycle*. A cycle follows another cycle (see Figure 4). In this paper, we sometime interchange cycle and program to denote the content of the broadcast channel.

An inherit challenge of the broadcast model is the forward only data model and that there is no random access of the data set. When a client misses a piece of data from the current cycle, the client must wait until the next cycle. The reverberations of these properties of broadcast environment are: (1) we cannot adopt disk-based skyline computation algorithms designed for traditional database systems that require random access, and (2) we must design a self-explanatory broadcast program using data index.

Many previous studies have done on different broadcast environment. Multi-channels broadcast for data dissemination (and techniques of data allocation) has been previously studied [8] [19] [6]. Adaptive broadcast systems that allow limited uplink (client to server) communication has been studied in [18]. In this paper, we assume the following properties for our broadcast environment:

1) Channel is forward-only.
2) Only one channel is utilized.
3) No uplink bandwidth from clients to server.

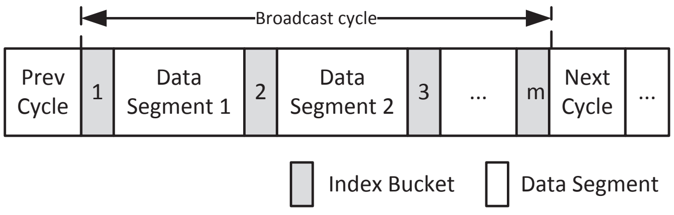


Fig. 4. Broadcast program cycle

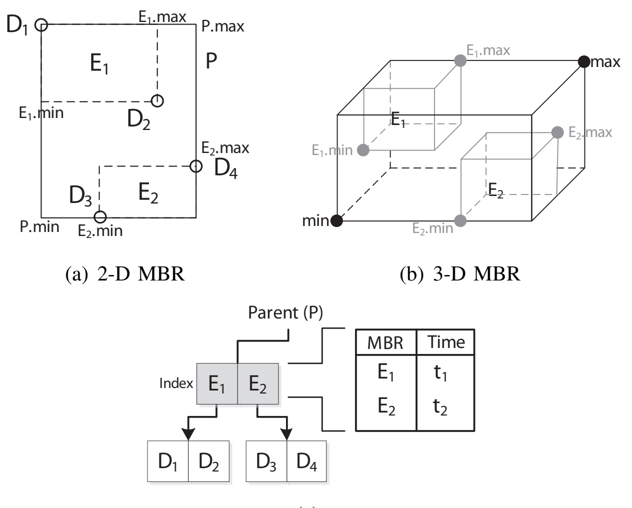
Power consumption is a major concern since in this environment, clients are small mobile devices, such as cellular phones, powered by a small battery; therefore, power is an scarce and valuable resource for these devices for most cases. Even for non-mobile clients energy conservation is a desirable attribute for a system. Actively listening to the broadcast channel is expensive in terms of power usage. Most mobile devices is able to turn off the radio receiver when it is not actively sending or receiving data to conserve power.

To reduce the power consumption, an index is used to make the broadcast cycle self-descriptive. As depicted in Figure 5, An index provides additional information in a broadcast program to tell the clients the approximate time when a data records will be broadcasted. A broadcast index is analogous to an index in traditional database management systems in which the indexes provide the location of data records and facilitate fast lookup of records. With the index information, the clients can turn off the radio receiver and transmitter to conserve power and only tune into the channel when the desire data is being broadcasted.

Adding index to a broadcast cycle also adds space overhead to the cycle and ultimately consume more broadcast bandwidth. Quantities that measure the efficiency of a wireless broadcast program are defined below, of which *tuning time* and *access latency* are well-known metrics and has been studied in other works preceeding this paper [20] [9] [11] [13]. We also define *Index Percentage* as a novel meansurement of the space overhead of the index structure.

Definition 4 Index Percentage The ratio of the space allocated to index to the space of the entire cycle measure in bytes. This quantity is defined as $\rho = \frac{\iota}{\omega}$.

Definition 5 Initial Index Probe The amount of time for a client to get to the first index segment. There are many methods of initial index probe, such as probing every δ amount of time. Another method is to tune into the channel the entire time until the first index is retrieved; nonetheless this quantity is not added to the tuning time. This quantity is denoted by α.



(a) 2-D MBR (b) 3-D MBR

(c)

Fig. 5. Tree Index.

Definition 6 Tuning time The total amount of time the client actively listening to the channel in order to retrieve desire data. This is measured in terms of the space of the index and data segments. This quantity is denoted by β.

Definition 7 Access latency The amount of time from when the client makes the request to when the client receives all the desired data. Similar to tuning time, this is also measured in terms of the space of the index and data segments. This quantity is denoted by γ.

III. Depth-First Distributed Index

Before we present our skyline computation algorithm in the next section, in this section how the index and data are allocated on the broadcast channel to form the broadcast program. We introduce depth-first distributed-index (DFDI), our index allocation technique, that is based on the distributed index allocation proposed in [9]. The benefits of a distributed index over other allocation methods has been discussed in section VI-B.

We utilize an R-Tree [4] to index our multi-dimensional data records. We assume that each node of an R-Tree consists of b number of entries, $\{E_1, E_2, ...E_b\}$, where b is the branching factor, or the number of children at each internal node, of the index tree, as illustrated by Figure 5. Each entry contains a pointer to a child index node, if the node is not a leaf node, or a pointer to a set of data records, if the node is a leaf node. The pointer is the time when the child item will appear on the broadcast channel.

In addition, each entry contains an orthotope, or a n-dimensional minimal bounding rectangle (MBR), that denote the extend of the child object pointed by the pointer of the entry (see Figure 5). Each rectangle MBR is defined by two opposite corners, $MBR.min$ and $MBR.max$, also known as the “lower-left” and “upper-right” corners of the rectangle. $min = (min_1, min_2, ..., min_n)$ and $max = (max_1, max_2, ..., max_n)$ are n-dimensional points containing minimal and maximal value of each attribute in the rectangle, respectively. In other words, min_i is the lower bound, and max_i is the upper bound of ith attribute in MBR.

An advantage of using R-Tree is that it is a flexible index that can support other spatial queries, such as range queries and kNN queries. Using an R-Tree also satisfies our goal of creating a flexible index that supports combination of skyline queries.

A. Broadcast Structure

A broadcast program cycle is a linear representation of the index tree and data and consists of *index segments* and *data segments*. Index segments contains temporal pointers to either another index segment or a data segment. Data segments contain actual data records. Index and data segments intermingle to for a broadcast program. Each index segment is further divided into smaller units called buckets. Buckets are logical independent units that represent a portion of the tree index. The purpose for buckets is that a client does not have to download an entire index segment if it only needs a bucket.

In addition to a list of temporal pointers, each index bucket also contains a pointer to the next index segment and a pointer to the beginning of next broadcast cycle. The purpose of these pointers is to direct the client to the next index segment in the case that the client tunes in at the index bucket but is not interested in the data pointed by the index.

To save bandwidth, data segments do not contain any index information other than data records. The broadcast program structure is illustrated in Figure 6.

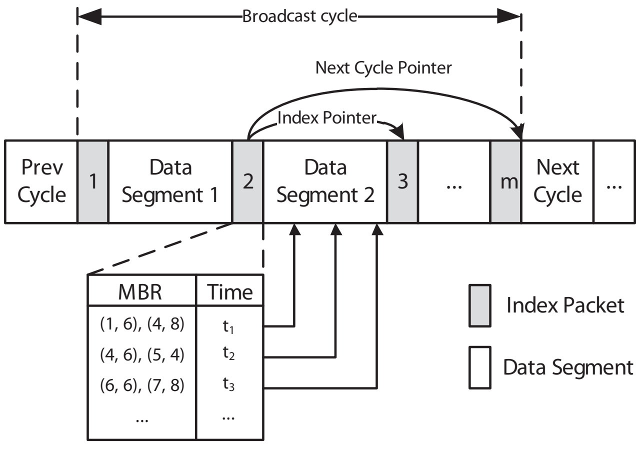


Fig. 6. Broadcast program cycle format with index segments and data segments.

B. DFDI Allocation

We have data indexed by a R-Tree. Our task now is to publish both data and index on to the linear broadcast channel. DFDI allocates space for index and data by performing a depth-first traversal of the index tree as illustrated in Figure 7. In this process, the root index node A is first included in the cycle because it is first traversed. Index node B_1 is then included in the program followed by C_1, then the data items D_1 and D_2. One might argue that why use depth-first traversal of the index tree, instead of breadth-first traversal (BFT). The reason is that BFT would cluster all index into one enormous index segment at the beginning of a cycle and basically create a 1-time index.

DFDI replicate L_r levels of the index tree b times, where the root node is considered level 1. Replication helps the clients get a broader picture of upcoming broadcast items. When an index node is replicated, only the MBRs that have no been broadcasted is published; therefore the replication is *not a complete replication* of the node. Given N to be the current node to be published, if the level of N is $L_r + 1$ or less, then the parent of N is replicated. Otherwise the parent is not replicated.

An example is shown in Figure 7, the root and the B level index nodes are replicated. The replication helps the client gets a broader picture of upcoming broadcast item. Of course the best view would be to replicate the entire index, but this would be the complete replication as discussed before and we try to avoid to save broadcast bandwidth.

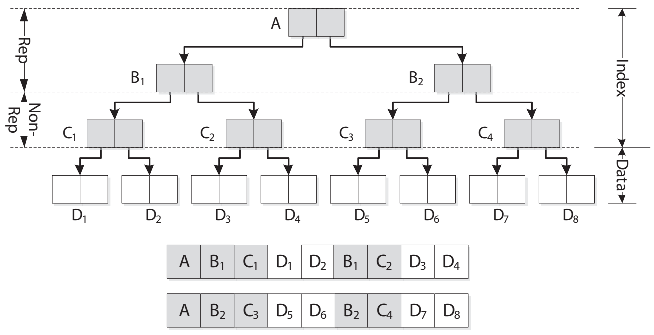


Fig. 7. Index Structure.

Algorithm 1 DFDIPublish($Node$, $Level$, L_r)

```
1.  PushToChannel($Node$) {Assume temporal ptrs are known}
2.  if $Node$ is Leaf then
3.     for all DataSegment in $Node$ do
4.        PushToChannel(DataSegment)
5.     end for
6.  else
7.     for all ChildNode in $Node$ do
8.        DFDIPublish(ChildNode, $Level$++)
9.        if $L_r \le L$ AND NOT Last ChildNode then
10.          PushToChannel($Node$) {Replication}
11.       end if
12.    end for
13. end if
```

C. Analysis

This section presents the efficiency analysis of our index design. The efficiency metrics are defined in section 2.2.

1) Index Percentage: Index percentage measures the space overhead of the index structure. It is defined to be the ratio between the space allocated to index on the broadcast cycle to the length of the entire broadcast cycle.

Let L_r be the levels of replication (for example, L_r for Figure 7 is 2), η be the space required for an index node, and ε be the space required for an index node entry. The number of nodes replicated in the cycle is $\sum_{i=0}^{L_r-1} b^i$. For each replicated node, the replication does not replicate the entire index node, only the MBRs that has not been pushed to the channel; therefore, additional space for each replicated node is $\eta \sum_{i=1}^{b-1} \frac{i}{b}$ or $\varepsilon \sum_{i=1}^{b-1} i$. The total space taken by index is the space for each node, plus the additional space for each replicated node and it is given by:

$$\iota = \eta(\sum_{i=0}^{L_r-1} b^i \sum_{i=1}^{b-1} \frac{i}{b} + \sum_{i=0}^{h} b^h) \tag{3}$$

Let θ be the size of the entire data set. The length of the broadcast cycle is the space of the index plus the space of the data:

$$\omega = \eta(\sum_{i=0}^{L_r-1} b^i \sum_{i=1}^{b-1} \frac{i}{b} + \sum_{i=0}^{h} b^h) + \theta \tag{4}$$

2) Initial Index Prob: The distributed index divides the entire data set into smaller data segments and reduces the initial prob of first index segment. The length of each data segment, denoted by ς, is determined by the number of index segments distributed among the broadcast cycle. As one see from Figure 7, there are four leaf-nodes and the data is divided into four segments; therefore, the number of data segment is b^h and size per data segment is

$$\varsigma = \frac{\theta}{b^h} \tag{5}$$

If we consider that the prob time is 0 when a client tunes in at an index segment, then the expected initial index prob is the average of length of each data segment:

$$E(\alpha) = \frac{2\theta}{b^h} = \frac{2\theta}{m} \tag{6}$$

This is a big time reduction compare with one-time index prob.

IV. Pruning Region Broadcast Skyline

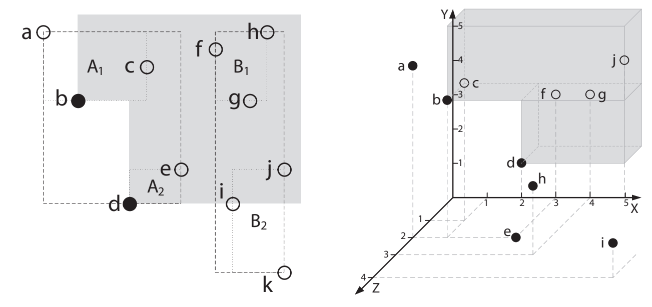
In this section, we present our point-based-pruning and index-based-pruning skyline computation algorithms. Point-based-pruning strategy is our first attempt to formulate a pruning-based algorithm and guarantees correct result with or without a broadcast index. Index-based-pruning generally provides better performance by performing early pruning. We present two index-pruning strategies here. The algorithms utilize the R-Tree index and allocation method described in previous section to build a pruning region to eliminate unwanted data records. Point-based-pruning skyline algorithm is explained and followed by index-based-pruning algorithm.

Definition 8 Pruning Region Given the data space defined by $D_1 \times D_2 \times ... \times D_n$, where D_i is the data domain of attribute i, a pruning region is a pair of a pivot point and an ordered list of preference specifiers (p, σ) that specifies a region of the data space that has been dominated by a subset of the data-set. As defined in Section II, σ_i is one of the values in $\{min, max\}$.

A pivot point, denoted by p here, is a n-dimensional point in the n-dimensional data space that defines the bounds of a pruning region. The bounds is defined by the list of preference specifiers. For each attribute i, if $\sigma_i = min$, then p_i is the lower bound of the pruning region for ith dimension and the region extends to the maximal value for ith dimension. Similarly, if $\sigma_i = min$, then p_i is the lower bound and extends to minimal value of data dimension. Pivot point and pruning region is illustrated in Figure 8(a) and 8(b). Pruning regions are illustrated as the gray regions and bounded by pivot points b and d. As illustrated, a pruning region is a region of n-dimensional space (or a n-dimensional box) that has been dominated by p and can be ignored in the upcoming broadcast data stream.

Corollary 2 The pivot point of a rectangle dominates all points of the rectangle.

Proof: Given a pivot point, p, of a rectangle, r, for all attribute i, if $\sigma_i = min$ then $p_i = r.min_i$, if $\sigma_i = max$ then $p_i = r.max_i$. This implies that the pivot point has the best value for all attributes in the rectangle, therefore no points can dominate it and it dominates all points of the rectangle except for itself. ■



(a) 2-D with preference specifiers (min, min)
(b) 3-D with preference specifiers (min, min, max)

Fig. 8. Pruning region with pivot points b and d

A. Point-Based Skyline Pruning

To compute the skyline, a pruning region is progressively augmented as more data records are examined. The algorithm keeps track of a list of candidate skyline points, S, and a pruning region that is the union of pruning region of S. As the pruning region grows, any p in S that is inside the pruning region is removed and any new MBR received from the broadcast cycle that is inside the pruning region is ignored. At the end of the algorithm, points in S that are not removed are returned as skyline.

Skyline computation is illustrated in Algorithm 2. At the beginning of the algorithm, the client does not have any candidate skyline points; therefore, the pruning region, R, is null ($R = \emptyset$). In this state, the client cannot prune any R-Tree MBR and therefore must stay tuned into the channel for the first index bucket broadcasted on the channel. The client follows a series of index buckets to the first data segment. When a data segment is reached, the client does:

1) Download all data points from the data segment.
2) Compute skyline points, S_0, using any existing skyline algorithm (NL, BNL, NN, BBS).
3) For each point in S_0, compute a pruning region, R', and and remove all points in S covered by R'.
4) For each R', $R = R \cup R'$
5) Add each point in S_0 to S.

This repeats until the entire space is inside the pruning region.

Note that the points found in each data segment are only candidate skyline points since there could data broadcasted later that dominate the earlier candidate points. If a candidate point p_2 dominates earlier point p_1, then p_1 is removed from S and the pruning region is enlarged by the later point p_2.

ComputePruneRegion() in Algorithm 2 is described next.

Algorithm 2 Point-Based Skyline(σ)

```
1.  $S \leftarrow \emptyset$
2.  $R \leftarrow \emptyset$
3.  $index \leftarrow$ search first index bucket
4.  $queue.enqueue(index.pairs(mbr, time))$
5.  while $queue$ is not empty do
6.    $bucket \leftarrow GetBucket(queue.dequeue)$
7.    if $bucket$ is index bucket then
8.      for all $pair(mbr, time)$ in $bucket$ do
9.        if $pair$ is not in $pruneRegion$ then
10.         $queue.enqueue(pair)$
11.       end if
12.     end for
13.   else
14.     $S_0 \leftarrow ComputeSkyline(bucket, \sigma)$
15.     $R' \leftarrow ComputePruneRegion(S_0, \sigma)$
16.     $Prune(S, R')$
17.     $Prune(queue, R')$
18.     $R \leftarrow R \cup R'$
19.     $S \leftarrow S \cup S_0$
20.   end if
21. end while
22. return $S$
```

1) Pruning and Pruning Region: Here we describe pruning region computation and pruning strategy listed in Algorithm 2.

To determine if a point q is covered (should be pruned) by a pruning region, all attributes of the point has to be checked against the pivot p of the pruning region. The condition for a point to be covered is that all attributes of point q is inside the pruning region. For a point to be covered, each attribute i satisfy one of the following:

1) If σ_i = MIN, then $q_i \geq p_i$.
2) If σ_i = MAX, then $q_i \leq p_i$.

To determine if an index bucket is covered by a pruning region and should be ignored when computing skyline as in Point-Based Skyline discussed previously, we need to determine if the MBR of the index bucket is covered. To do so, we need to determine the pivot point of the MBR according to the preference specifiers. Based on Corollary 2, if the pivot point of a pruning region dominates the pivot point of the MBR, then the index bucket can be pruned and ignored, since the pivot point of the MBR dominates the entire bucket. Following the definition of a rectangle defined in Section II-B, a MBR is covered by a pruning region if its pivot is covered by the pruning region as illustrated by Figure 9. A pruning region is illustrated in gray and defined by pivot point p with the skyline query specifiers of $(X = min, Y = max)$. MBR A can be pruned since its pivot p_1 falls inside the pruning region, while although MBR B is partially covered by the pruning region, it can not be pruned since its pivot, p_2, does not fall inside the pruning region, in other words p dominates p_1, but does not dominate p_2.

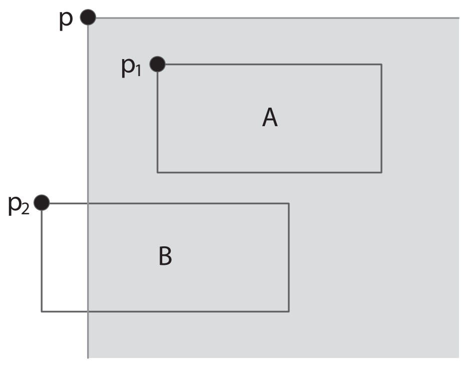


Fig. 9. Pruning region defined by p and preference specifier $(X = min, Y = max)$ and MBRs A and B.

B. Index-Based Skyline Pruning

The drawback of Algorithm 2 is that the pruning regions are formed from candidate skyline points that have been retrieved from the broadcast channel. The pruning regions are formed late in the program since data segments are broadcasted after corresponding index segments. In this section we present index-based pruning algorithm that perform early index-based pruning and two pruning strategies that produce the same tuning time from a client view. The expectation is that with early pruning, more of the index tree would be pruning and improve efficiency over point-based pruning strategy.

The two index-pruning strategies are one-region and n-regions. The one-region pruning strategy, forms only one pruning region per MBR of the index tree as illustrated by Figure 10(a). This strategy is based on the claim that we can prune everything from the pivot point of the MBR of the index. The following present a short proof of the correctness of this claim for a 2-dimensional case:

Proof: Given a 2-dimensional MBR, r, and skyline query of (min, min). There must exist at least one point, p, in r such that $p_2 = r.min_2$, p_1 in range $[r.min_1, r.max_1]$. Similarly, there must exist another point, q such that $q_1 = r.min_1$, q_2 in range $[r.min_2, r.max_2]$.

p dominates anything to the right of p_1 and above $r.min_2$ and q dominates all records above q_2 and to the right of $r.min_2$. We extend the p_2 value of the prune region of p and the q_1 value of the pruning region of q and these two are the same as the values for the pruning region of point $(r.min_1, r.min_2)$. Since MBRs do not overlap, pruning regions for point $(r.min_1, b.min_2)$ is the pruning region of the MBR r. ■

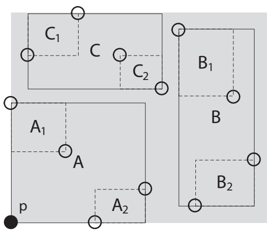
Although this approach is simple, the client must keep track of the current MBR which is inside the pruning region, but should not be pruned. For example, MBR A is inside pruning region, but it is the MBR of the index currently being investigated. The region of A should not be pruned.

n-region pruning strategy is illustrated in Figure 10(b). This pruning strategy is more natural way of query evaluation. The remaining of this section discuss the point-based skyline evaluation algorithm using n-region strategy.

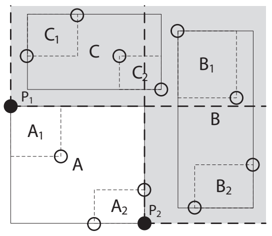
Index-Based skyline algorithm start the same way as the point-based skyline algorithm in that initially, the pruning region is null and the client must stay tuned to the broadcast channel until it finds the first index bucket. When the client receives an index bucket that is not covered by the pruning region (in this case it is the first index bucket), it performs the following:

1) For each entry, create a set of n pruning regions, R', where n is the dimensionality of the dataset. This is explained below.
2) Remove points in candidate skyline, S, that are covered by R'.
3) Add the new pruning regions, R', to the total pruning region. For each R', $R = R \cup R'$.

The rationale for step 1 above to create n pruning regions for each entry is that we want prune the data dominated by the entry, but we do not want to prune the data that is bounded by the MBR of the entry before we download data. Figure 10 demonstrates this idea. Assuming the current index bucket we get from the broadcast channel is MBR A and it is not covered by the pruning region R, since MBR A is not covered, we want to follow this index to its children A_1 and A_2 and ultimately download the data points in A. If we form the pruning region of A using the strategy of Figure 10(a), then A_1 and A_2 will be pruned before we have the chance to download the data.



(a) Pruning with one Pruning Region



(b) Prune with n Pruning Regions

Fig. 10. Index-Based Pruning Strategies (min, min)

Following, the strategy of Figure 10(b), we have a chance to download the data.

To create n pruning regions, we need a pivot point for each pruning region. Let r be an orthotope (a MBR) and $i, j = \{1, 2, ...n\}$, such that p_i is the ith pivot point of r, and $p_{i,j}$ is the jth attribute (or coordinate) of ith pivot point. p_i is computed in the way that $p_{i,i}$ is the opposite of σ_i and $p_{i,j}$ is the same as σ_i for all $j \neq i$.

For example, for a skyline query of all MIN attributes, $\sigma = (MIN, MIN, MIN)$ and therefore $\sigma_i = MIN$. In this case, $p_{i,i} = r.max_i$ and $p_{i,j} = r.min_j$ for $j \neq i$.

Algorithm 3 Index-Based Skyline(σ)

```
1.  $S \leftarrow \emptyset$
2.  $R \leftarrow \emptyset$
3.  $index \leftarrow$ search first index segment
4.  $queue.enqueue(index.pairs(mbr, time))$
5.  while $queue$ is not empty do
6.    $bucket \leftarrow GetBucket(queue.dequeue)$
7.    if $bucket$ is Index Packet then
8.      for all $entry \leftarrow pair(mbr, time)$ in $bucket$ do
9.        if $R$ is not in $entry$ then
10.         $queue.enqueue(pair)$
11.         $R' \leftarrow ComputePruneRegion(e.mbr, \sigma)$
12.         $Prune(S, R')$
13.         $Prune(queue, R')$
14.         $R \leftarrow R \cup R'$
15.       end if
16.     end for
17.   else
18.     $S \leftarrow S \cup ComputeSkyline(bucket)$
19.   end if
20. end while
21. return $S$
```

C. Analysis

This section we consider tuning time and access latency of the two pruning region skyline algorithm we presented in this section. We assume that the client makes skyline query at very beginning of a cycle. Since the amount of time of time listening to the channel is the amount of time for a client to get all desire data, the tuning time is equal to access latency; therefore we use the same analysis for both quantities.

We first consider the tuning time for the algorithm that uses the point-based pruning strategy. The best case is that the client gets all desired data from the first data segment and the rest of the cycle is pruned. In this case, $\beta = h \times \eta + \varsigma$. The expected case is when the client has to listen to half of the program and $E(\beta) = \frac{1}{2}(\iota + \theta)$

We now consider the same quantity for the index-based pruning skyline algorithm. The assumption is the same as before, but a client does not have to traverse the index to the leaf level before start pruning. On average, a client would have to traverse half of the index tree but still have to download half of the data; therefore $E(\beta) = \frac{1}{4}\iota + \frac{1}{2}\theta$

V. Experimental Evaluation

In this section, we report our simulation results and evaluate the efficiency of DFDI, our program allocation algorithm, and point-based and index-based skyline computation algorithms discussed in previous sections of this paper. Our simulations are implemented in C# with .NET Framework 4 and backward compatible with .NET Framework 2.

We generated our synthetic data-sets for our simulation. Each data record is contains n attributes or an n-dimensional point. Our data generator is able to generate three kinds of data-sets:

- Uniformed: data uniformly distributed in the data space of each attribute.
- Rising: the attributes of the records are correlated to the first attribute of the data-set.
- Falling: the attributes of the records are inversely correlated to the first attribute of the data-set.

Each data-set is characterized by the following two additional parameters:

- Record count: the number of records in a data-set.
- Dimension: the number of attributes of the data-set.

The records count for our data-sets ranges from 20,000 to 100,000 and the dimension ranges from 2 to 10 dimension. Each data-set is a file on disk and are loaded into memory during simulation.

Our simulations are memory-based. Since this paper is the first on the topic of using R-Tree for broadcast skyline evaluation, memory-based simulations give us a quick tool to verify our approach since they are faster than disk-based implementations. The simulations load the data-sets from disk and runs all simulations from memory. We implemented our own in-memory R-Tree index to facilitate our simulations. Table III lists the size matrix that is used in the implementation of the simulation of tuning time and index percentage.

Our simulations are conducted on a machine with 3.4 GHz Intel Pentium 4 processor and 3 GB of RAM running Windows 7. Since our results are measured in the number of bytes and that .NET Framework has been implemented on many difference systems, the execution environment has little affect on the experimental result.

TABLE III
SIMULATION SIZE MATRIX

Item	**Size in Bytes**
Pointer of index (ptr)	4
Field of record (f)	8
Record/point (p)	$f \times n$
Minimal bounding rectangle (MBR)	$2 \times p$
Index entry (E)	MBR + ptr

A. *Dominance Tests*

Dominance tests measures the number record comparisons the client has to perform to evaluate a skyline query. A comparison determines if a record dominates another record. Intuitively, as the number of records in the a data-set increases, the number of dominance tests also increases.

In our proposed skyline computation algorithms, dominance tests are performed when the client reaches a data segment. The client downloads the data records in the data segment and computes the candidate skyline points in the segment. Our simulation uses Block-Nested-Loop (BNL) [2] algorithm to compute skyline points inside a data segment.

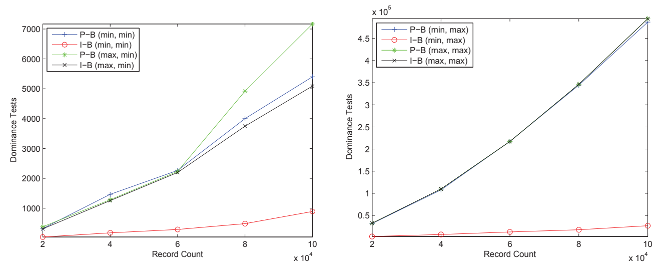
Figure 11 shows the results of simulating a client finding all skyline record from the broadcast program and the number of dominance tests incurred with increasing record count. For example, the client performed 321 dominance tests to get all skyline points from the broadcast cycle when the number of data records is 20,000 (lower bound) for I-P (min, min). The simulation is run with dimension (d) equals 3 and the branching factor (b) of the tree index equals 10.

Figure 11 covers all cases of combinations of min and max attributes in 2-dimensional data for Point-Based (P-B) and Index-Based (I-P) skyline algorithms. Figure 11(a) shows I-P algorithms performs better than I-B and I-P (min, min) performs the best. The performance is R-Tree implementation depend and due to our implementation order index entries based on the distance to the origin, (min, min) skyline queries naturally perform better than other queries. Similarly, in figure 11(b), the algorithm can only prune very little due to the ordering of the MBRs; therefore, P-B and I-P based algorithm performs almost the same for (max, max) skyline queries.

Figure 12 compares the number of dominance tests with increasing data dimensions. The experiments are conducted with the record count (rc) of 10,000, and branching factor of 10. Figure 12(a) compares the result when skyline query attribute specifiers are all min and all max. Figure 12(b) shows the result of the number of dominance tests with different data types. The two figures show that as the number of data dimensions increases, the volume (or space) of the data also increases. This leads to more space for the records to "hide" and not fall into the pruning region and therefore the number of dominance tests increases.

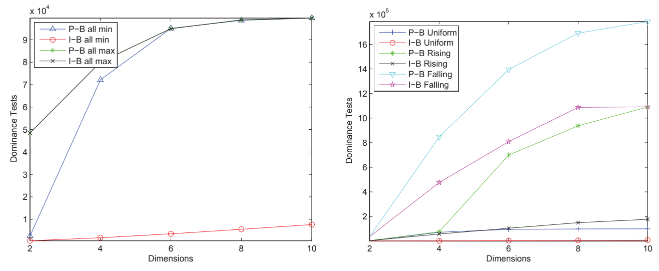
B. *Tuning Time*

As discussed in section II-B, tuning time is total amount of data the client has to download to fulfill a skyline query and it is measured in bytes. The experiment simulates the server



(a) (min, min) and (max, min) (b) (min, max) and (max, max)

Fig. 11. Dominance Tests vs. Record Count. d = 2, b = 10, Data = uniformed



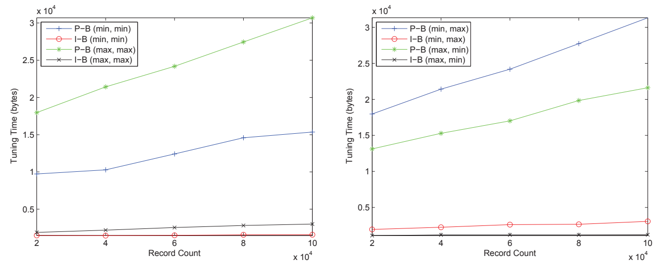
(a) All Min and All Max. Data = uniformed (b) Mixed Data Types

Fig. 12. Dominance Tests vs. Dimensions. rc = 10000, b = 10

creating the DFDI broadcast program. Tuning time is found by simulating a client evaluating a skyline query from the beginning of the cycle.

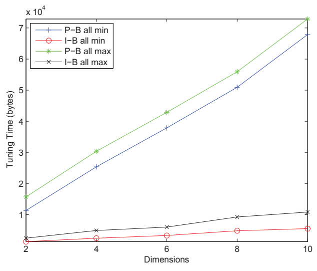
Figure 13 illustrates tuning time versus increasing record count for all combinations of min and max attribute for 2-dimensional data. In all cases, the Index-Based (I-P) pruning strategy performs several factors better than Point-Based (P-B) pruning strategies.

Figure 14 illustrates the simulation result for tuning time with increasing data dimension. The experiments are run with record count (rc) of 10000, and branching factor (b) of index tree of 10. Although the number of record stays the same, each additional dimension or data attribute of the data-set adds space complexity to the data-set. With increasing increasing dimensionality, the cycle length also increases, so tuning time.

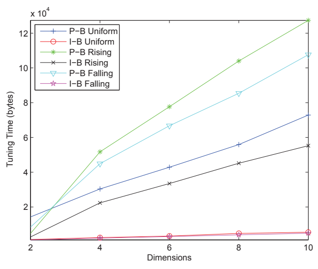


(a) (min, min) and (max, max) (b) (min, max) and (max, min)

Fig. 13. Tuning Time vs. Record Count. d = 2, b = 10

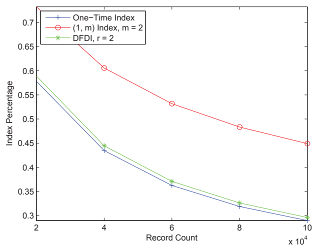


(a) All Min and All Max

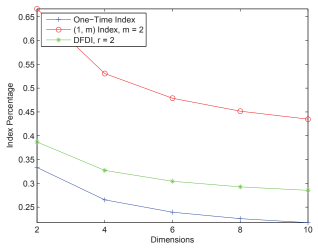


(b) Mixed Data Types

Fig. 14. Tuning Time vs. Dimensionality. rc = 10000, b = 10



(a) IP vs. Record Count



(b) IP vs. Dimensionality

Fig. 15. Index Percentage. b = 10

C. *Index Percentage*

Index percentage measures the efficiency of the DFDI broadcast program allocation technique under increasing record count and increasing data dimension. Index percentage is defined in section II-B.

Figure 15(a) shows index percentage versus increasing record count. For the DFDI, the simulation is run with replication level of 2, which means the root and the first level below the root are replicated. For the (1, m) index, m = 2, meaning the complete index is duplicated 2 times in the broadcast cycle. The figure shows that the overhead of the index is high when the number of record is low, but the overhead flattens as the number of records grow. The one-time index is the baseline and as expected has the lowest index percentage. Although DFDI replicated the index for 2 levels, its index percentage is only slightly (16%) higher than one-time index. This shows DFDI is efficient in terms of space overhead. Whereas the (1, m) index has far worse overhead for only 2 duplications.

Figure 15(b) shows index percentage over increasing data dimension. The experiment is conducted with 10000 records and branching factor of 10. As the data size grows with the number of dimensions, the index is only slightly affected by the growth. The size of index grows because of the index needs extra information to index the extra dimensions, but the tree height is largely unaffected; thus gives the falling of index percentage with growing dimension.

VI. Related Work

A. *Skyline Computation*

Techniques of computation of skyline records in traditional database systems have been studied in [2], [10], and [15]. The well known algorithms are Nest-Loop, Block-Nested-Loop, and Divide and Conquer. The Nest-Loop algorithm is an intuitive way to compute skyline points in the way that every record is compared with every other records in a table to determine if the record is dominated. In Block-Nested-Loop, a record is only compared with other records in the same block. The candidate skyline points in each block is compare to obtain the final skyline points. In Divide and Conquer (D&C), the data set is recursively divided until there are only two records. Skyline points are calculated for each segment produced in the division phase. The division phase is followed by a merge phase in which the skyline of all divisions are compare and merge to obtain the final skyline set.

[10] introduces an online skyline computation algorithm in which the skyline are computed progressively. The first skyline is return almost immediately and more skyline points are added to the result set.

Nearest Neighbor (NN) and Branch-and-Bound (BBS) skyline algorithms are two of the best performing algorithms for progressive skyline computation for traditional database systems presented in [15]. In NN skyline algorithm, the records of the data set are presented geometrically in a Euclidean space with the relevant attributes as the coordinate in each dimension. In the example of hotel close to the ocean in previously presented, a record would be placed on a plane with the distance of the hotel to the ocean and its price as the two axis that determine and the values in the two attributes as the coordinates of the records on the Euclidean plane as in Figure 16. The NN algorithm finds a nearest neighbor from the axes and that is the first skyline point. Then the algorithm marks the region that is dominated by the first skyline point so that anything in that region will not be searched. The search results in two more search regions and the search continues until no more records are left.

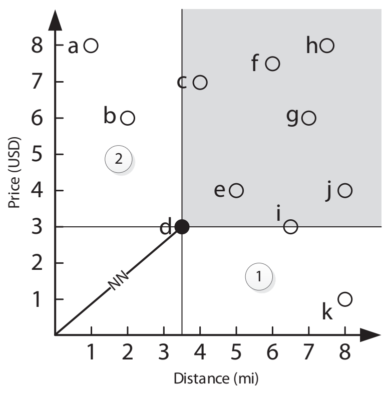


Fig. 16. NN d and its pruning region.

BBS is an algorithm that surpass the computation efficiency of the NN algorithm. BBS utilizes the best first search technique to traverse the index tree to prune the branches unnecessary branches. Unfortunately, both NN and BBS can not be easily adopted to the broadcast environment due to the linear nature of broadcast program. Both NN and BBS requires backtracking of the index tree to find the best path to prune, which is impermissible in broadcast environment or require waiting for the next broadcast cycle, which incurs long waiting time. The solutions presented in this paper will use the pruning regions strategy used in NN. Our algorithms systematically builds pruning regions, as presented in NN algorithm, as the client receives and discovers more data from the broadcast channel.

B. Wireless Broadcast Index

Many excellent studies have been done on improving efficiency of wireless broadcast system using indexing techniques. This section considers several popular index allocation techniques and discusses benefits and drawbacks.

The intuitive technique of no index and one-time index at beginning of a broadcast cycle has been considered in [9]. With no index, the length of a cycle is minimized, but the tuning time is the entire cycle since the program is not self-descriptive. With one-time index, the clients are able to filter unwanted data and reduce tuning time, but if a client misses the one-time index, then it will have to wait until the next cycle even if there is useful data in current cycle.

$(1, m)$ indexing, proposed by Imielinski, et al. in [9], is a mitigation to the problem of one-time index by replicating the entire index every $1/m$ length of the broadcast cycle. The benefit of this index technique is when a client misses an index segment, it can wait for the next index in the same cycle [11]. The drawback of this technique is the space consumption of replicating the full index several times in the cycle.

Distributed index was also proposed in [9]. This index also replicate the index in the broadcast cycle, but only a part of the index is replicated. Advantages of this method are (1) index can be obtained throughout cycle, (2) reduced bandwidth consumption comparing with $(1, m)$, and (3) not limited to particular index structure.

Data filtering based on data signature was proposed in [13]. During data retrieval, the signature of desire data is compared with the signature of a data segment prepared by the broadcast server. If the signatures match, then data is downloaded, else ignored.

Distributed index for spatial data in error-prone air broadcast was introduced in [20]. Instead of replication, his paper proposes a distributed index in the broadcast cycle with no duplicate of index. The paper index broadcast spatial data using Hilbert values. Each data record contains an index table that contains the data segments that will be pushed onto the channel in the near future. A drawback of this approach is the lose of spatial precision due to the use of space filling curve as index.

VII. Conclusion

In this paper, we presented a broadcast data stream allocation technique (DFDI) that utilize the R-Tree and performs a depth-first traversal of the index to create a distributed index. The goal of DFDI are to facilitate query processing from broadcast data, to reduce index overhead (IP) and, to improve the initial index prob. DFDI is able to achieve all these with reasonable efficiency. DFDI is a flexible R-Tree based index and well-supports skyline queries as well as other data query types. The allocation distributes b^h number of index segments among the broadcast program to reduce initial index at the same time keeps the index overhead low. The simulation results for index percentage show DFDI performs very well with 2 levels of replication and follow the efficiency of one-time index with only 16% increase in index overhead.

The simulation result show that the approach also performs well with data of higher dimensions. The index overhead decreases as the number of records remain constant and the number of data dimension increase. This is due to the growth of dimensionality does not make the index tree grow "taller" and does not incur the cost of new nodes when the index grows. The height of an index tree does increase as the number of records increase, but as seen in Figure 15(a), the growth of the index is not as fast as the growth of the amount of data; therefore, the index overhead decreases as records increase.

In addition, we introduced point-based and index-based pruning skyline algorithms. The experiments shows both algorithms are capable of evaluate skyline queries of combined min and max attributes with reasonable tuning time and dominance tests. The index-based skyline had always performed better than the point-based skyline, in some cases several factors. The performance of the algorithms is also affected by the data arrangement and R-Tree implementation. From our simulation, we find that R-Tree that index records with lower attributes first, performs better for min skyline queries, and vice versa.

References

[1] W.-T. Balke, U. Güntzer, and J. X. Zheng. Efficient Distributed Skylining for Web Information Systems. In *EDBT*, pages 256–273, 2004.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *ICDE*, pages 421–430, 2001.

[3] P. Godfrey, R. Shipley, and J. Gryz. Maximal Vector Computation in Large Data Sets. In *VLDB*, pages 229–240, 2005.

[4] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD Conference*, pages 47–57, 1984.

[5] J. Ha, Y. Kwon, J.-H. Choi, and S. Lee. Energy Efficient and Progressive Strategy for Processing Skyline Queries on Air. In *DEXA*, pages 486–500, 2009.

[6] S. Hameed and N. H. Vaidya. Log-Time Algorithms for Scheduling Single and Multiple Channel Data Broadcast. In *MOBICOM*, pages 90–99, 1997.

[7] T. Hara. Cooperative caching by mobile clients in push-based information systems. In *CIKM*, pages 186–193, 2002.

[8] C.-H. Hsu, G. Lee, and A. L. P. Chen. A Near Optimal Algorithm for Generating Broadcast Programs on Multiple Channels. In *CIKM*, pages 303–309, 2001.

[9] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on Air: Organization and Access. *IEEE Trans. Knowl. Data Eng.*, 9(3):353–372, 1997.

[10] D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *VLDB*, pages 275–286, 2002.

[11] W.-S. Ku, R. Zimmermann, and H. Wang. Location-Based Spatial Query Processing in Wireless Broadcast Environments. *IEEE Trans. Mob. Comput.*, 7(6):778–791, 2008.

[12] H. T. Kung, F. Luccio, and F. P. Preparata. On Finding the Maxima of a Set of Vectors. *J. ACM*, 22(4):469–476, 1975.

[13] W.-C. Lee and D. L. Lee. Using Signature and Caching Techniques for Information Filtering in Wireless and Mobile Environments. *ACM Journal of Wireless Networks*, 5:57–67, 1996.

[14] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the Sky: Efficient Skyline Computation over Sliding Windows. In *ICDE*, pages 502–513, 2005.

[15] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[16] D. O. Seong, J. H. Park, M. H. Yeo, and J. S. Yoo. An energy-efficient skyline query processing method using priority-based bottom-up filtering. In *Proceedings of the 2009 International Conference on Hybrid Information Technology*, ICHIT '09, pages 159–162, New York, NY, USA, 2009. ACM.

[17] Y. Tao and D. Papadias. Maintaining Sliding Window Skylines on Data Streams. *IEEE Trans. Knowl. Data Eng.*, 18(2):377–391, 2006.

[18] J. Wong. Broadcast delivery. *Proceedings of the IEEE*, 76(12):1566 –1577, dec 1988.

[19] W. G. Yee and S. B. Navathe. Efficient data access to multi-channel broadcast programs. In *CIKM*, pages 153–160, 2003.

[20] B. Zheng, W.-C. Lee, K. C. K. Lee, D. L. Lee, and M. Shao. A distributed spatial index for error-prone wireless data broadcast. *VLDB J.*, 18(4):959–986, 2009.