Air Indexes for Spatial Databases

Baihua Zheng
School of Information Systems
Singapore Management University
80 Stanford Road, Singapore 178902
bhzheng@smu.edu.sg

SYNONYMS

None

DEFINITION

Air indexes refer to indexes employed in wireless broadcast environments to address scalability issue and to facilitate power saving on mobile devices [3]. To retrieve a data object in wireless broadcast systems, a mobile client has to continuously monitor the broadcast channel until the data arrives. This will consume a lot of energy since the client has to remain active during its waiting time. The basic idea of air indexes is that by including index information about the arrival times of data items on the broadcast channel, mobile clients are able to predict the arrivals of their desired data. Thus, they can stay in power saving mode during waiting time and switch to active mode only when the data of their interests arrives.

HISTORICAL BACKGROUND

In spatial databases, clients are assumed to be interested in data objects having spatial features (e.g., hotels, ATM, gas stations). "Find me the nearest restaurant" and "locate all the ATM that are within 100 meters to my current location" are two examples. A central server is allocated to keep all the data, based on which the queries issued by the clients are answered. There are basically two approaches to disseminating spatial data to clients: 1)**on-demand access**: a mobile client submits a request, which consists of a query and the query's issuing location, to the server. The server returns the result to the mobile client via a dedicated point-to-point channel. 2) **periodic broadcast**: data are periodically broadcast on a wireless channel open to the public. After a mobile client receives a query from its user, it tunes into the broadcast channel to receive the data of interest based on the query and its current location.

On-demand access is particularly suitable for light-loaded systems when contention for wireless channels and server processing is not severe. However, as the number of users increases, the system performance deteriorates rapidly. Compared with on-demand access, broadcast is a more scalable approach since it allows simultaneous access by an arbitrary number of mobile clients. Meanwhile, clients can access spatial data without reporting to the server their current location and hence the private location information is not disclosed.

In the literature, two performance metrics, namely *access latency* and *tuning time*, are used to measure access efficiency and energy conservation, respectively [1]. The former means the time elapsed between the moment when a query is issued and the moment when it is satisfied, and the latter represents the time a mobile client stays active to receive the requested data. As energy conservation is very critical due to the limited battery capacity on mobile clients, a mobile device typically supports two operation modes: *active mode* and *doze mode*. The device normally operates in active mode; it can switch to doze mode to save energy when the system becomes idle.

With data broadcast, clients listen to a broadcast channel to retrieve data based on their queries and hence are responsible for query processing. Without any index information, a client has to download all data objects to process spatial search, which will consume a lot of energy since the client needs to remain active during a whole broadcast cycle[1]. A solution to this problem is *air indexes* [3]. The basic idea is to broadcast an index before data objects (see Figure 1 for an example). Thus, query processing can be performed over the index instead of actual data objects. As the index is much smaller than the data objects and is selectively accessed to perform a query,

[1] A broadcast cycle means the minimal duration within which all the data objects are broadcast at least once.

the client is expected to download less data (hence incurring less tuning time and energy consumption) to find the answers. The disadvantage of air indexing, however, is that the broadcast cycle is lengthened (to broadcast additional index information). As a result, the access latency would be worsen. It is obvious that the larger the index size, the higher the overhead in access latency.

An important issue in air indexes is how to multiplex data and index on the sequential-access broadcast channel. Figure 1 shows the well-known $(1, m)$ scheme [3], where the index is broadcast in front of every $\frac{1}{m}$ fraction of the dataset. To facilitate the access of index, each data page includes an offset to the beginning of the next index. The general access protocol for processing spatial search involves the following three steps: 1) initial probe: the client tunes into the broadcast channel and determines when the next index is broadcast; 2) index search: The client tunes into the broadcast channel again when the index is broadcast. It selectively accesses a number of index pages to find out the spatial data object and when to download it; and 3) data retrieval: when the packet containing the qualified object arrives, the client downloads it and retrieves the object.

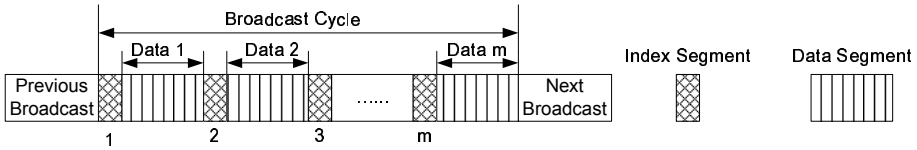
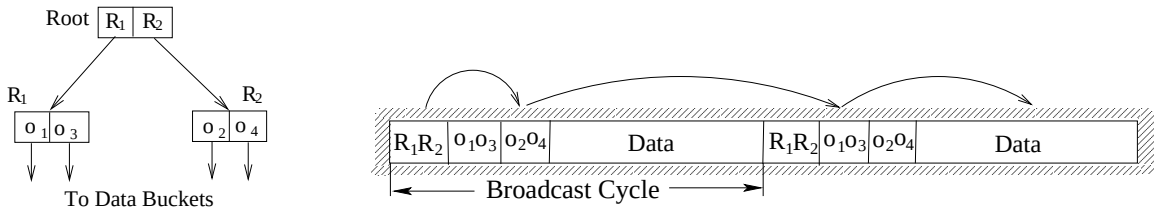


Figure 1: Air Indexes in Wireless Broadcast Environments

To disseminate spatial data on wireless channels, well known spatial indexes (e.g., R-trees) are candidates for air indexes. However, unique characteristics of wireless data broadcast make the adoption of existing spatial indexes inefficient (if not impossible). Specifically, traditional spatial indexes are designed to cluster data objects with spatial locality. They usually assume a resident storage (such as disk and memory) and adopt search strategies that minimize I/O cost. This is achieved by *backtracking* index nodes during search. However, the broadcast order (and thus the access order) of index nodes is extremely important in wireless broadcast systems because data and index are only available to the client when they are broadcast on air. Clients cannot randomly access a specific data object or index node but have to wait until the next time it is broadcast. As a result, each backtracking operation extends the access latency by one more cycle and hence becomes a constraint in wireless broadcast scenarios.



(a) R-tree Index

(b) Branch-and-Bound Search

Figure 2: Linear Access on Wireless Broadcast Channel

Figure 2 depicts an example of spatial query. Assume that an algorithm based on R-tree first visits root node, then the node R_2, and finally R_1, while the server broadcasts nodes in the order of root, R_1, and R_2. If a client wants to backtrack to node R_1 after it retrieves R_2, it will have to wait until the next cycle because R_1 has already been broadcast. This significantly extends the access latency and it occurs every time a navigation order is different from the broadcast order. As a result, new air indexes which consider both the constraints of the broadcast systems and features of spatial queries are desired.

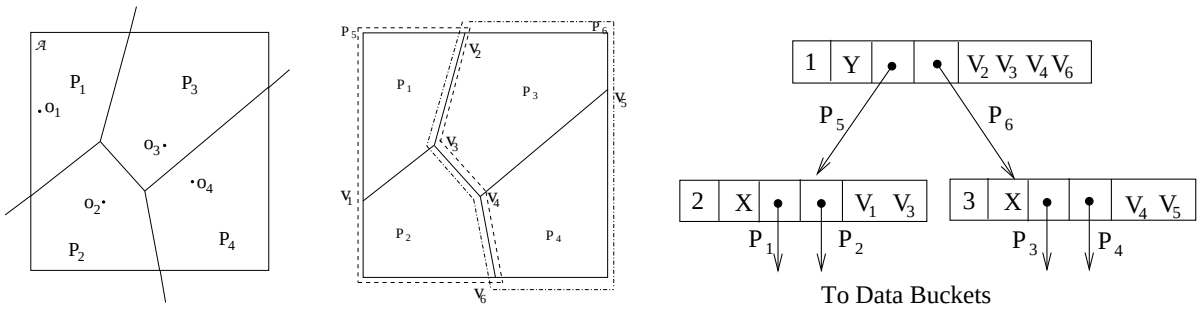
SCIENTIFIC FUNDAMENTALS

Several air indexes have been recently proposed to support broadcast of spatial data. These studies can be classified into two categories, according to the natural of the queries supported. The first category focuses on retrieving data associated with some specified geographical range, such as "Starbucks Coffee in New York City's Times Square" and "Gas stations along Highway 515". A representative is the index structure designed for *DAYS*

project [1]. It proposes a location hierarchy and associates data with locations. The index structure is designed to support query on various types of data with different location granularity. The authors intelligently exploit an important property of the locations, i.e., containment relationship among the objects, to determine the relative location of an object with respect to its parent that contains the object. The containment relationship limits the search range of available data and thus facilitates efficient processing the supported queries. In brief, a broadcast cycle consists of several sub-cycles, with each containing data belonging to the same type. A major index (one type of index buckets) is placed at the beginning of each sub-cycle. It provides information related to the types of data broadcasted, and enables clients to quickly jump into the right sub-cycle which contains her interested data. Inside a sub-cycle, minor indexes (another type of index buckets) are interleaved with data buckets. Each minor index contains multiple pointers pointing to the data buckets with different locations. Consequently, a search for a data object involves accessing a major index and several minor indexes.

The second category focuses on retrieving data according to specified distance metric, based on client's current location. An example is nearest neighbor (NN) search based on Euclidian distance. *D-tree* is a paged binary search tree to index a given solution space in support of planar point queries [5]. It assumes a data type has multiple data instances, and each instance has a certain *valid scope* within which this instance is the only correct answer. For example, restaurant is a data type, and each individual restaurant represents an instance. Take NN search as an example, Figure 3(a) illustrates four restaurants, namely o_1, o_2, o_3, and o_4, and their corresponding valid scopes p_1, p_2, p_3, and p_4. Given any query location q in, say, p_3, o_3 is the restaurant to which q is nearest. D-tree assumes the valid scopes of different data instances are known and it focuses only on planar point queries which locate the query point into a valid scope and returns the client the corresponding data instance.

The D-tree is a binary tree built based on the divisions between data regions (e.g., valid scopes). A space consisting of a set of data regions is recursively partitioned into two complementary subspaces containing about the same number of regions until each subspace has one region only. The partition between two subspaces is represented by one or more polylines. The overall orientation of the partition can be either x-dimensional or y-dimensional, which is obtained, respectively, by sorting the data regions based on their lowest/uppermost y-coordinates, or leftmost/rightmost x-coordinates. Figure 3(b) shows the partitions for the running example. The polyline pl(v_2, v_3, v_4, v_6) partitions the original space into p_5 and p_6, and polylines $pl(v_1, v_3)$ and $pl(v_4, v_5)$ further partition p_5 into p_1 and p_2, and p_6 into p_3 and p_4, respectively. The first polyline is y-dimensional and the remaining two are x-dimensional. Given a query point q, the search algorithm works as follows. It starts from the root and recursively follows either the left subtree or the right subtree that bounds the query point until a leaf node is reached. The associated data instance is then returned as the final answer.

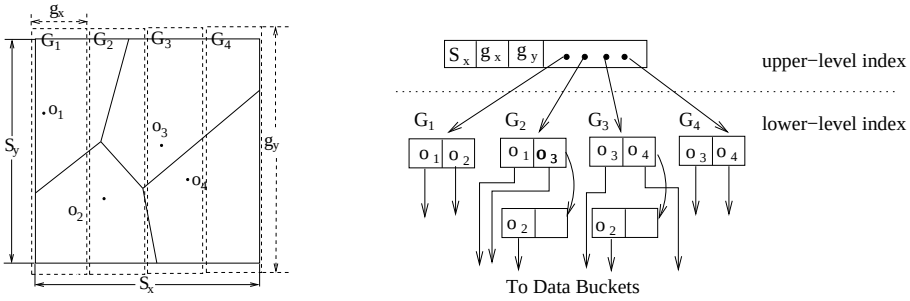


(a) Data Instances and Valid Scopes (b) Space Division (c) D-tree Structure

Figure 3: Index Construction Using the D-tree

Grid-partition index is specialized for NN problem [8]. It is motivated by the observation that an object is the NN only to the query points located inside its Voronoi Cell. Let $O = \{o_1, o_2, \cdots, o_n\}$ be a set of points. $\mathcal{V}(o_i)$, the *Voronoi cell* (VC) for o_i, is defined as the set of points q in the space such that $dist(q, o_i) < dist(q, o_j)$, $\forall\ j \neq i$. That is, $\mathcal{V}(o_i)$ consists of the set of points for which o_i is the NN. As illustrated in Figure 3(a), p_1, p_2, p_3, and p_4 denote the VCs for four objects, o_1, o_2, o_3, and o_4, respectively. It tries to reduce the search space for a query at the very beginning by partitioning the space into disjoint grid cells. For each grid cell, all the objects that could be NNs of at least one query point inside the grid cell are indexed, i.e., those objects whose VCs overlap with the grid cell are associated with that grid cell.

Figure 4(a) shows a possible grid partition for the running example, and the index structure is depicted in Figure 4(b). The whole space is divided into four grid cells; i.e., G_1, G_2, G_3, and G_4. Grid cell G_1 is associated with objects o_1 and o_2, since their VCs, p_1 and p_2, overlap with G_1; likewise, grid cell G_2 is associated with objects o_1, o_2, o_3, and so on. If a given query point is in grid cell G_1, the NN can be found among the objects associated with G_1 (i.e., o_1 and o_2), instead of among the whole set of objects. Efficient search algorithms and partition approaches have been proposed to speed up the performance.



(a) Grid Partition (b) Index

Figure 4: Index Construction Using the Grid-Partition

Conventional spatial index R-tree has also been adapted to support kNN search in broadcast environments [2]. For R-tree index, the kNN search algorithm would visit index nodes and objects sequentially as backtracking is not feasible on the broadcast. This certainly results in a considerably long tuning time especially when the result objects are located in later part of the broadcast. However, if the client knows that there are at least k objects in the later part of the broadcast that are closer to the query point than the currently found ones, they can safely skip the downloading of the intermediate objects currently located. This observation motivates the design of the enhanced kNN search algorithm which caters for the constraints of wireless broadcast. It requires each index node to carry a count of the underlying objects ("object count") referenced by the current node. Thus, clients do not blindly download intermediate objects.

Hilbert Curve Index (HCI) is designed to support general spatial queries, including window queries, kNN queries, and continuous nearest-neighbor (CNN) queries in wireless broadcast environments. Motivated by the linear streaming property of the wireless data broadcast channel and the optimal spatial locality of the Hilbert Curve (HC), HCI organizes data according to Hilbert Curve order [6, 7], and adopts B^+-tree as the index structure. Figure 5 depicts a 8 grid, with solid dots representing data objects. The numbers next to the data points, namely *index value*, represent the visiting orders of different points at Hilbert Curve. For instance, data point with $(1, 1)$ as the coordinates has the index value of 2, and it will be visited before data point with $(2, 2)$ as the coordinates because of the smaller index value.

The *filtering and refining* strategy is adopted to answer all the queries. For window query, the basic idea is to decide a candidate set of points along the Hilbert curve which includes all the points fallen within the query window and later to filter out those fallen outside the window. Suppose the rectangle shown in Figure 5 is a query window. Among all the points within the search range, the first point is point a and the last is b, sorted according to their occurring orders on the Hilbert curve, and both of them are lying on the boundary of the search range. Therefore, all the points inside this query window should lie on the Hilbert curve segmented by points a and b. In other words, data points with index values between 18 and 29, but not the others, are the candidates. During the access, the client can derive the coordinates of data points based on the index values and then retrieve those within the query window.

For kNN query, the client first retrieves those k nearest objects to the query point along the Hilbert curve and then derives a range which for sure bounds at least k objects. In the filtering phase, a window query which bounds the search range is issued to filter out those unqualified. Later in the refinement phase, k nearest objects are identified according to their distance to the query point. Suppose a NN query at point q (i.e., index value 53) is issued. First, the client finds its nearest neighbor (i.e., point with index value 51) along the curve and derives a circle centered at q with r as the radius (i.e., the green circle depicted in Figure 5). Since the circle bounds point 51, it is certain to contain the nearest neighbor to point q. Second, a window query is issued to retrieve all the data points inside the circle, i.e., points with index values 11, 32, and 51. Finally, the point 32 is identified as

the nearest neighbor. The search algorithm for CNN adopts a similar approach. It approximates a search range which is guaranteed to bound all the answer objects, issues a window query to retrieve all the objects inside the search range, and finally filters out those unqualified.

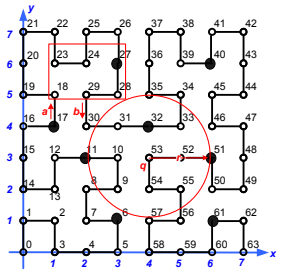


Figure 5: Hilbert Curve Index

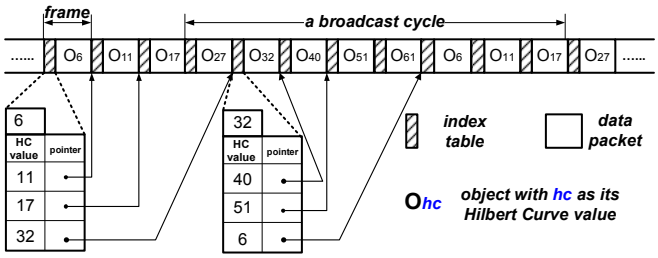


Figure 6: Distributed Spatial Index

All the indexes mentioned above are based on a central tree-based structure, like R-tree and B-tree. However, employing a tree-based index on a linear broadcast channel to support spatial queries results in several deficiencies. Firstly, clients can only start the search when they retrieve the root node in the channel. Replicating the index tree in multiple places in the broadcast channel provides multiple search starting points, shortening the initial root-probing time. However, a prolonged broadcast cycle leads to a long access latency experienced by the clients. Secondly, wireless broadcast media is not error-free. In case of losing intermediate nodes during the search process, the clients are forced to either restart the search upon an upcoming root node or scan the subsequential broadcast for other possible nodes in order to resume the search, thus extending the tuning time. *Distributed spatial index (DSI)*, a fully distributed spatial index structure, is motivated by these observations [4].

Very different from tree-based indexes, DSI is not a hierarchical structure. Index information of spatial objects is fully distributed in DSI, instead of simply replicated in the broadcast. With DSI, the clients do not need to wait for a root node to start the search. The search process launches immediately after a client tunes into the broadcast channel and hence the initial probe time for index information is minimized. Furthermore, in the event of data loss, clients resume the search quickly.

Like HCI, DSI also adopts Hilbert curve to determine broadcast order of data objects. Data objects, mapped to point locations in a 2-D space, are broadcast in the ascending order of their HC index values. Suppose there are N objects in total, DSI chunks them into n_F *frames*, with each having n_o objects ($n_F = \lceil N/n_o \rceil$). The space covered by Hilbert Curve shown in Figure 5 is used as a running example, with solid dots representing the locations of data objects (i.e., $N = 8$). Figure 6 demonstrates a DSI structure with n_o set to 1, i.e., each frame contains only one object.

In addition to objects, each frame also has an index table as its header, which maintains information regarding to the HC values of data objects to be broadcast with specific waiting interval from the current frame[2]. Every index table keeps n_i entries, each of which, τ_j, is expressed in the form of $\langle HC'_j, P_j \rangle$, $j \in [0, n_i)$. P_j is a pointer to the r^j-th frame after the current frame, where r (> 1) is an exponential base (i.e., a system-wise parameter), and HC'_j is the HC value of the first object inside the frame pointed by P_j. In addition to τ_j, an index table also keeps the HC values HC_k ($k \in [1, n_o]$) of all the objects obj_k that are contained in the current frame. This extra information, although occupying litter extra bandwidth, can provide a more precise image of all the objects inside current frame. During the retrieval, a client can compare HC_ks of the objects against the one she has interest in, so the retrieval of unnecessary object whose size is much larger than an HC value can be avoided.

Go back to the example shown in Figure 5, with corresponding DSI depicted in Figure 6. Suppose $r = 2$, $n_o = 1$, $n_F = 8$, and $n_i = 3$. The index tables corresponding to frames of data objects O_6 and O_{32} are shown in the figure. Take the index table for frame O_6 as an example: τ_0 contains a pointer to the next upcoming (2^0-th) frame whose first object's HC value is 11, τ_1 contains a pointer to the second (2^1-th) frame with HC value for the first object (the only object) 17, and the last entry τ_2 points to the fourth (2^2th) frame. It also keeps the HC value 6 of the object O_6 in the current frame. Search algorithm for window queries and kNN searches are proposed.

[2]This waiting interval can be denoted by delivery time difference or number of data frames apart, with respect to the current frame.

KEY APPLICATIONS

Location-based Service

Wireless broadcast systems, because of the scalability, provide an alternative to disseminate location-based information to a large number of users. Efficient air indexes enable clients to selectively tune into the channel and hence the power consumption is reduced.

Moving Objects Monitoring

Many moving objects monitoring applications are interested in finding out all the objects that currently satisfy certain conditions specified by the users. In many cases, the number of moving objects is much larger than the number of submitted queries. As a result, wireless broadcast provides an ideal way to deliver subscribed queries to the objects, and those objects that might affect the queries can then report their current locations.

CROSS REFERENCE

NEAREST NEIGHBOR QUERY
SPACE FILLING CURVES FOR QUERY PROCESSING
SPATIAL INDEXING TECHNIQUES

RECOMMENDED READING

[1] D. Acharya and V. Kumar. Location based indexing scheme for days. In *MobiDE*, pages 17–24, 2005.

[2] B. Gedik, A. Singh, and L. Liu. Energy efficient exact knn search in wireless broadcast environments. In *GIS*, pages 137–146, 2004.

[3] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air - organization and access. *TKDE*, 9(3), 1997.

[4] W.-C. Lee and B. Zheng. Dsi: A fully distributed spatial index for wireless data broadcast. In *ICDCS*, pages 349–358, 2005.

[5] J. Xu, B. Zheng, W.-C. Lee, and D. L. Lee. The d-tree: An index structure for location-dependent data in wireless services. Technical Report 12, TKDE, 2002.

[6] B. Zheng, W.-C. Lee, and D. L. Lee. Spatial queries in wireless broadcast systems. *ACM/Kluwer Journal of Wireless Networks (WINET)*, 10(6):723–736, 2004.

[7] B. Zheng, W.-C. Lee, and D. L. Lee. On searching continuous k nearest neighbors in wireless data broadcast systems. *IEEE Transactions on Mobile Computing*, 6(7):748–761, 2007.

[8] B. Zheng, J. Xu, W.-C. Lee, and D. L. Lee. Grid-partition index: A hybrid method for nearest-neighbor queries in wireless location-based services. *VLDB Journal*, 15(1):21–39, 2006.