Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

5-2005

Information Dissemination via Wireless Broadcast

Baihua ZHENG
Singapore Management University, bhzheng@smu.edu.sg

Dik Lun LEE
Hong Kong University of Science and Technology

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

Part of the Databases and Information Systems Commons, and the Numerical Analysis and Scientific Computing Commons

Citation

Information Dissemination via Wireless Broadcast

Baihua Zheng[†] and Dik Lun Lee[‡]

[†] School of Information Systems, Singapore Management University

bhzheng@smu.edu.sg

[‡]Department of Computer Science, Hong Kong University of Science and Technology

dlee@cs.ust.hk

Introduction

The advent of sensor, wireless and portable device technologies will soon enable us to embed computing technologies transparently in the environment to provide uninterrupted services for our daily life. With temperature and location sensors and wireless access points embedded in an environment, a person entering an environment can be automatically connected to the environment. All of his personal computing devices will then be adapted to the context, making all important information readily available for the user to tackle his tasks at hand. An important step towards the realization of this pervasive environment is to be able to disseminate timely and relevant information to the user anytime, anywhere. This paper provides an overview of current research on ubiquitous data access and dissemination on wireless networks, emphasizing the differences between the traditional database and mobile environments, and describing some recent research on data broadcast on wireless channels.

System Architecture

Today, many wireless technologies such as Bluetooth, Wi-Fi (802.11b) and 3G global communication networks are available. Although these technologies vary in many aspects, we can look at them as an abstract cellular model where users access information through access points. The abstract model consists of a base station, a number of clients and a number of channels. A client can acquire an

uplink channel to send a request to the base station and receive the result from a *downlink* channel. While we assume that many users can listen to the same downlink channel to achieve broadcasting from the base station, an uplink channel is dedicated to one client for transmission at any time.

On-Demand vs Broadcast

There are two basic approaches to disseminating data to mobile clients. In *on-demand access*, a mobile client submits a request to the server, which then returns the results to the mobile client directly via point-to-point connection. In *periodic broadcast*, data are periodically broadcast on a wireless channel. A mobile client listens to the broadcast channel and downloads the desirable data from the channel according to a query issued from the user or a stored profile of interest on the client.

In on-demand access, the client is actively pulling data from the server. The server is responsible for processing the query and returning the answer directly to the client. On the other hand, periodic broadcast is equivalent to the server actively pushing data to the clients. The server determines what data should be broadcast and its schedule on the channel. The client listens to the channel and decides what should be retrieved.

In a lightly loaded system, where the number of queries generated from the clients is relatively small, on-demand access provides fast service because the waiting time for available channel is small. However, its performance deteriorates rapidly as the system workload increases because of the contention for bandwidth among the clients.

Data broadcast is an attractive alternative to on-demand access because it can broadcast data simultaneously to a large number of clients at a fixed cost. It is suitable for location-based services which exhibit strong temporal and spatial locality in that clients within the vicinity and a certain time period tend to seek the same kind of information. For example, a museum may want to broadcast attractions, open hours, etc., to all of its visitors [2], whereas on-demand, point-to-point access is used only for fulfilling occasional, ad hoc requests.

Broadcast system can be further classified as pure broadcast or on-demand broadcast. Clients in pure broadcast do not transmit any query to the server but just listen passively to the channel for the interesting data items. This works in a closed data space such as a stock quote system or when a profile of user interest is known in advance. Microsoft's DirectBand Network, a low-bandwidth

broadcast built on FM stations, is a pure broadcast system for subscribers who had expressed their interests through a web site [3]. On-demand broadcast allows the user to transmit a query as in the on-demand access, but the result to the query is broadcast together with results from other queries on the same channel. The client has to filter out the result designated to him/her.

Performance Measures

The mobile computing environment has a few characteristics that distinguish itself from the traditional wired network. In addition to the traditional access latency, mobile clients have limited energy power. As a result, access techniques must take energy consumption into consideration.

In on-demand access, power consumption is dominated by the number of request transmissions since all the client needs to do is to transmit the query and wait for the result, leaving all of the processing work to the server. In this regard, effective caching methods are vital in the reduction of the number of queries needed. For broadcast, since the client has to scan the channel for the interested data item, its power consumption can be considered as directly proportional to the time it is listening actively to the channel, which is referred to as *tuning time*.

On-Disk vs On-Air Indexing

Portable devices have limited power resources due to their small size. It is inefficient in terms of power consumption for the client to retrieve data by continuously listening to the channels until the desired data arrives. Imagine a client wanting to monitor the price of a stock. The device has to actively listen to thousands of stock symbols on the broadcast just to pick up one of them to display. Indexing can be used to guide the client in the listening process so that it will activate only when the data arrives and thus reducing power consumption. To contrast, while a traditional database index maps a key to where the key value is stored in memory, an index for broadcast data maps a key to the time when the key value is broadcast.

In traditional database indexes, data is stored on disks or in main memories. The data is always available and can be accessed randomly. However, in wireless broadcast, data is available "on air" in the sense that the data is available on the wireless channel only transiently and has to be accessed sequentially as dictated by the broadcast program. Search algorithms and index methods

for wireless broadcast channels should avoid back-tracking. Existing database indexes obviously were not designed to meet this requirement and hence perform poorly on broadcast data.

When indexing is used, the broadcast cycle contains the index together with the actual data. When the client is looking for information, it tunes to the channel to locate the index for the broadcast cycle, from which it can obtain the broadcast time of the information and sleeps until it arrives. Compared to the total size of the data records, the index only contains the key names and thus is relatively small. It achieves the objective of reducing power consumption without significantly increasing the access latency.

Location-Based Services

Research on data broadcast is typically concerned with scheduling, indexing and caching. In general, broadcast data is assumed to be location independent. For example, the price of a stock is time dependent but in general not location dependent. That is, no matter where the data is stored and where the user is located, the answer to a query on stock price is the same. However, in mobile computing where users move around, location becomes an important dimension of data. The answer to a query depends not only on the data values but also on the location where the query was issued. These queries are called *location-dependent queries* (LDQs). The data involved in answering LDQs are called *location-dependent data* (LDD). In this paper, we will examine three types of LDQs:

1. *Nearest-Neighbor Queries:* Where is the nearest restaurant?

2. *k-Nearest-Neighbor Queries:* Where are the 10 nearest restaurants?

3. *Window Queries:* Where are the hotels within this county?

LDQs can be regarded as spatial queries, where location conditions are derived from the location of the user. For example, the location constraint for nearest-neighbor queries is the current location of the user and that for window queries is a rectangular region bound the country.

Spatial queries have been studied for a long time. It is natural to assume that traditional solutions to spatial queries can be well applied to broadcast data. Unfortunately, this is not the case, as is demonstrated later.

4

Valid Scopes

For LDQs, a data item has different data values in different geographic regions known as the *Valid Scopes* (VSs) of the data item. Take ZIP code as an example. For the query "Give me the ZIP Code of my current location", clients in Princeton will receive NJ $08544 - 2087$ while those in UCLA will get CA $90095 - 1361$. ZIP code can be regarded as a location-dependent data item, and its valid scope is defined by the ZIP code boundary map (see Figure 1(a)).

Valid scopes can be defined differently for different types of applications. The VS of ZIP code is defined by the postal office, whereas the VS of nearest-neighbor queries is defined by the Voronoi Diagram (VD) [1]. Figure 1(b) shows two international airports in the state of New Jersey. The state is partitioned into two parts by the Voronoi Diagram. For this simple example, it is the perpendicular bisector between the two airports. For clients in the region above the dashed line, the Newark airport is the nearest one, while the Atlantic City airport is the nearest one for the clients below the dashed line.

Valid scopes can also be defined symbolically. For example, when a user moves across cells in a cellular system, he needs to find out which cell his is in. The valid scope for this type of queries is the radio coverage of the cells, as shown in Figure 1(c).

In general, valid scope does not have to be a contiguous geographical region. For example, when a user asks for the temperature at his current location, the answer may be 40°F. The valid scope for this answer could consist of disjoint regions, in which the temperatures are all 40°F (shadowed parts of Figure 1(d)).

We can see that some applications require the valid scopes to be known (e.g., in the ZIP code and temperature examples), whereas some applications are primarily interested in object locations (e.g., the locations of the airports or base stations). In the latter, queries can be computed purely based on the object locations. However, the major benefit of knowing the valid scope is two-fold. First, it leads to the development of fast indexing and query processing algorithms, which will be described later. Second, once a query is answered, the client does not have to ask the same query again as long as he stays within the valid scope. This can be used as a spatial caching scheme to significantly reduce the number of queries submitted and is mandatory for the support of continuous queries [6, 7].
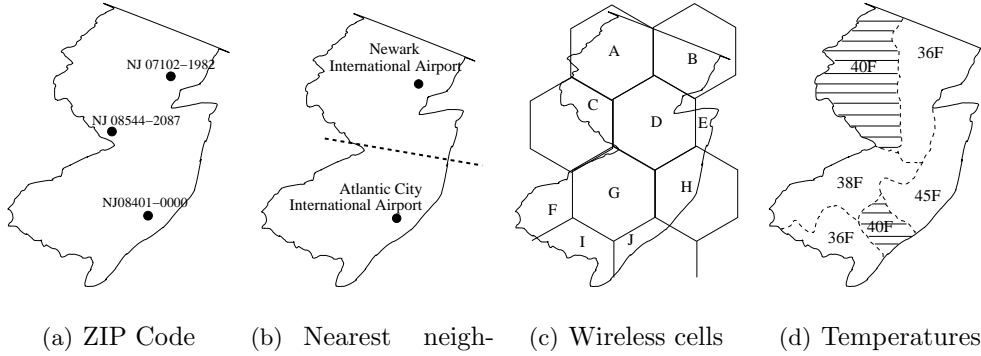
(a) ZIP Code (b) Nearest neigh- (c) Wireless cells (d) Temperatures
bors

Figure 1: Examples of Valid Scope (New Jersey)

Object vs Valid Scope Indexing

Indexing methods for LDD can be divided into two categories, namely, object-based and solution-based indexes. An object-based index is built on object locations, like traditional R-tree which records the coordinates of each object in the tree. An object-based index can be used to answer a broad range of queries, including *NN*, *kNN*, and window queries, since it records the raw location information about the objects and thus supporting any spatial queries.

A solution-based index is built for a specific type of queries. A pre-computed solution space is first obtained for the type of queries. Then, an index is built on the solution space. For the LDQs discussed in this article, the solution space is the valid scope, as illustrated in Figure 1. A solution-based index essentially indexes the boundaries of the valid scopes so that given a query point, it can identify the valid scope containing the query point and return the associated data (e.g., the ZIP code information or temperature).

Mapping Points to Valid Scopes

We can see that LDQs require an index to map a query point into one of the valid scopes and return the associated data of the valid scope. Since the valid scope of a query can be viewed as a set of polygons, the index essentially maps a query point to one of the polygons. Besides meeting the linear search requirement of wireless broadcast, the index must be small so that it won't lengthen the broadcast cycle and thus the access latency. Most spatial indexes were developed for the traditional disk-based environment and as such are not optimized for any of these requirements.

D-tree is a design motivated by the above considerations [5]. The basic idea is to index VSs

based on the divisions that form VSs' boundaries. For a space containing a set of VSs that are disjoint and complementary, it recursively partitions the space into two sub-spaces having similar number of VSs until each space only contains one VS. Figure 2(a) shows a VD for fours objects. Polyline $pl(v_2, v_3, v_4, v_6)$ partitions the original space into P_5 and P_6, and $pl(v_1, v_3)$ 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. Each node of the D-tree contains some header information (e.g., partition dimension), the pointers to the children, and the partition in the form of a polyline. The search algorithm starts from the root, and recursively follows either the left pointer or the right pointer according to the partition and the query point until a data pointer is reached. The D-tree has been shown to have a much better performance than the traditional indexes in terms of tuning time and access latency [5].

We can also see that D-tree's storage requirement is minimal in the sense that the polylines are the minimum information needed for the reconstruction of the *VS*. Furthermore, given a query point, the search from the root node to the target leaf node is fixed and traverses only one branch, making it suitable for the linear broadcasting environment.
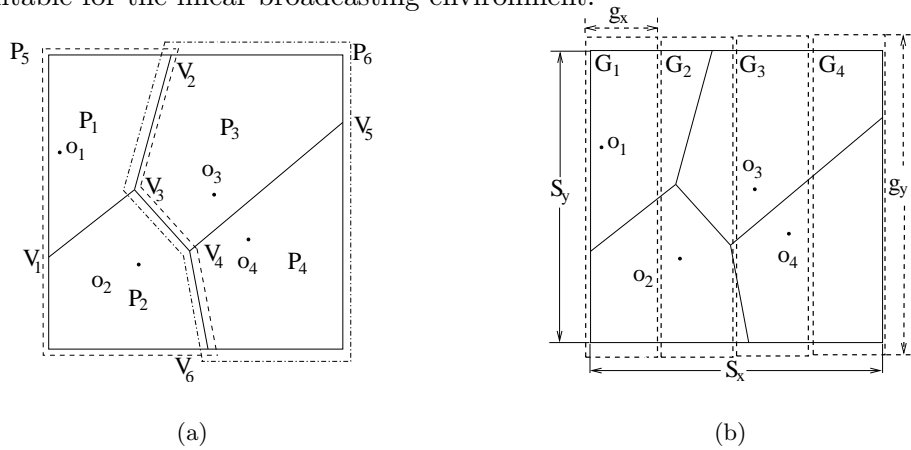

(a)   (b)

Figure 2: (a) D-tree and (b) Grid-Partitioned Indexes

A Hybrid Index for NN queries

In principle, an LDQ can always be answered using a solution-based approach because for any type of query a solution space must exist and can be pre-computed if we don't consider the construction, update and storage costs. However, the storage cost of a solution-based approach is definitely larger than that of an object-based approach. This is because a VS typically has many edges. It requires

much more space to index the polygons than the objects, thus leading to poor search speed and high update cost if the index structure is not designed appropriately.

The *grid-partition index* is an efficient index for *NN* queries. Given a *VD*, the grid-partition index first partitions it into disjoint grid cells. For each grid cell, it identifies and stores the objects that are *potential NN*s of any query point that falls into the grid cell. In Figure 2(b), the *VD* is partitioned into grid cells of equal size, namely G_1 to G_4. Take G_1 as an example, a query point within G_1 can only have two potential *NN*s, namely, O_1 and O_2. As such, this association, $\langle G_1, O_1, O_2 \rangle$ can be kept in the index.

The next problem for the realization of the grid-partition index is to identify the potential *NN*s for a grid cell. For *NN* search, each subspace in VD defines the valid scope of that object. Hence, only the objects whose VSs overlap with that grid cell can be a potential nearest neighbor to a query point in that cell. However, this method requires the *VD* to be first constructed, which is expensive to create and maintain. To avoid the use of *VD*, a new algorithm was developed to find all potential *NN*s to a grid cell using the Delaunay Triangulation (DT), the straight-line dual of *VD*s [1]. The Delaunay Graph is a graph having the objects as vertices and edges connect two objects if and only if their *VS*s are adjacent. It has some nice properties: it can be computed without first obtaining the *VD*, thus avoiding the high cost of constructing the *VD*; it can be computed incrementally, thus lending itself to efficient updates.

The grid-partition index can be considered as a hybrid index, combining the advantages of solution-based and object-based indexes. It is a solution-based index because it indexes the valid scope of a query. However, instead of storing the boundaries of the valid scope, it stores the object locations. As a result, it is both fast and storage efficient.

General Spatial Queries

Although work has been done in finding valid scopes for *kNN* and window queries [6], the overheads in computing and broadcasting the valid scopes for different query types could be expensive. In this section, we look at a different approach that is based on object indexing. Since valid scopes are not pre-computed, it can support both *kNN* and window queries at the same time.

Since most spatial queries incur searching on objects that are close to each other, we can apply space-filling curves to arrange objects on the broadcast so that spatially near objects are put

8

together on the broadcast. Figure 3 shows the basic Hilbert Curve of order 1 and 2. To derive a curve of order i, each vertex of the basic curve is replaced by an order $(i-1)$ curve, which may be strategically rotated and/or reflected to fit the new curve.

The numeric labels represent the positions of the objects on the Hilbert Curve. For example, point $(1,1)$ has the index value 2 in the order 2 curve shown in Figure 3(b). After the Hilbert Curve is obtained, an index (e.g., a B+-tree) is built on the labels and serves as the index of the broadcast objects.

Given a query window, we can identify the first point and the last point within the window that the Hilbert Curve visits [4]. In the example shown in Figure 3(b), the rectangle in dotted line is the query window, and points a and b are, respectively, the first and last point within the query window visited by the Hilbert Curve. We can conclude that points lying before a and after b along the Hilbert Curve are not inside the query window. Consequently, the client only needs to check the objects between a and b inclusive on the curve (i.e., points between 2 and 13 inclusive), compute their actual Euclidean distances, and eliminate those that are not lying within the window (i.e., objects 10, 11 and 12).

To answer a *kNN* query, we can first determine the smallest possible window containing the k nearest neighbors. Then, we can check the objects within the window to find the actual k nearest neighbors. This method requires scanning the Hilbert Curve twice. In the first scan, we identify the k objects around the query point on the Hilbert Curve. Based on these objects, we can determine the smallest window centered at q that encloses these k objects. It can be proven that this window is guaranteed to contain the *kNN* objects of q in the Euclidean space. The window approximates the search range within which further search can be done to obtain the actual answers.

In the second scan, the objects within the window are fetched and their Euclidean distances from the query point is computed to obtain the k nearest neighbors. The example in Figure 3(b) shows a *4NN* query issued at point q. The first scan returns objects 5, 6, 8, and 9, which are the immediate objects before and after q on the Hilbert Curve. However, it is clear that they are not actually the four nearest objects to the query point in the 2-dimensional Euclidean space but they can be used to approximate the search range containing the results (see the dotted square circle centered at q in Figure 3(b)). The second scan will return all of the objects within the search window. This is similar to the window query described earlier, but in this case the client checks the

Euclidean distances to find out the four actual nearest objects (i.e., objects 2, 4, 6, and 8 within the dashed circle).
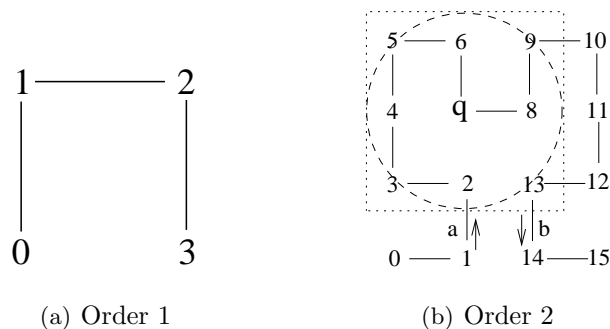

(a) Order 1   (b) Order 2

Figure 3: Window Query and *k*NN Search based on Hilbert Curve

We can see that space filling curves can answer these common spatial queries efficiently and are suitable for broadcast data. *kNN* queries require navigating the index information twice. This can be done by broadcasting the index twice within a broadcast cycle.

Conclusion

Wireless networking and portable digital devices provide people with unrestricted mobility. Consequently, location becomes a very important property of data and introduces a new dimension to the design of data access methods. Traditional data access methods are not suitable for the mobile environment since they were designed for wired environment where users do not usually move. In this paper, we analyze the differences between data access methods in these two environments and provide an overview of recent research in dealing with spatial queries in the mobile environment with a particular emphasis on the broadcast data. The combination of wireless broadcast, sensor networks, data/user mobility and context awareness will sure open up a lot of interesting research issues towards the realization of pervasive computing environments.

Glossary

Voronoi Diagram   Based on n given point objects, Voronoi Diagram partitions the space into n sub-spaces, with each sub-space containing exact one object. For any point within a sub-space, the corresponding object is its nearest neighbor.

Delaunay Triangulation   It is the dual graph of Voronoi Diagram. The objects in Voronoi Diagram are the vertexes, and any two objects whose corresponding sub-spaces are adjacent are connected using a segment.

Hilbert Curve   Hilbert Curve is one kind of space-filling curve, which visits all the objects in an n-dimensional space only once without crossing itself. The basic curve is depicted in Figure 3(a) with order 1. To derive a curve of order i, each vertex of the basic curve is replaced by an order $(i-1)$ curve, which may be strategically rotated and/or reflected to fit the new curve.

References

[1] M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*, chapter 7. Springer-Verlag, New York, NY, USA, 1996.

[2] K. Cheverst, K. Mitchell, and N. Davies. The role of adaptive hypermedia in a context-aware tourist guide. *Communications of the* ACM, 45(5):47–51, 2002.

[3] Microsoft Corporation. What is the directband network? URL at http://www.microsoft.com/ resources/spot/direct.mspx, 2003.

[4] D. Moore. Hilbert curve. URL at http://www.caam.rice.edu/ dougm/ twiddle/Hilbert.

[5] J. Xu, B. Zheng, W.-C. Lee, and D. L. Lee. Energy efficient index for querying location-dependent data in mobile broadcast environments. In *Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE'03)*, pages 239–250, Bangalore, India, March 2003.

[6] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. Lee. Location-based spatial queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'03)*, San Diego, CA, 2003.

[7] B. Zheng and D. L. Lee. Semantic caching in location-dependent query processing. In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD'01)*, pages 97–116, Los Angeles, CA, USA, July 2001.