Noname manuscript No.
(will be inserted by the editor)

A Hierarchical Grid Index (HGI), Spatial Queries in Wireless Data Broadcasting

Kwangjin Park and Patrick Valduriez

Received: date / Accepted: date

Abstract The main requirements for spatial query processing via mobile terminals include rapid and accurate searching and low energy consumption. Most location-based services (LBSs) are provided using an on-demand method, which is suitable for light-loaded systems where contention for wireless channels and server processing is not severe. However, as the number of users of LBSs increases, performance deteriorates rapidly since the servers' capability to process queries is limited. Furthermore, the response time of a query may significantly increase with the concentration of users' queries in a server at the same time. That is because the server has to check the locations of users and potential objects for the final result and then individually send answers to clients via a point-to-point channel. At this time, an inefficient structure of spatial index and searching algorithm may incur an extremely large access latency.

To address this problem, we propose the Hierarchical Grid Index (HGI), which provides a light-weight sequential location-based index structure for efficient LBSs. We minimize the index size through the use of hierarchical location-based identifications. And we support efficient query processing in broadcasting environments through sequential data transfer and search based on the object locations. We also propose Top-Down Search and Reduction-Counter Search algorithms for efficient searching and query processing. HGI has a simple structure through elimination of replication pointers and is therefore suitable for broadcasting environments with one-dimensional characteristics, thus enabling rapid and accurate spatial search by reducing redundant data. Our performance evaluation shows that our proposed index and algorithms are accurate and fast and support efficient spatial query processing.

Kwangjin is with the School of Electrical Electronics and Information Engineering, Wonkwang University, Iksan-Shi, Chunrabuk-do 570-749, Republic of Korea.
E-mail: kjpark@wku.ac.kr. ·
P. Valduriez is with INRIA and LIRMM 161 rue Ada 34392 Montpellier Cedex 5 France.
E-mail: Patrick.Valduriez@inria.fr.

Keywords Moving objects, mobile computing, wireless data broadcasting

1 Introduction

Location-based services (LBSs) refer to spatial query processing services and include map services, augmented reality (AR) technology, and navigation. These services determine the location of surrounding buildings using the Global Positioning System (GPS) and the geographical location of individuals who carry mobile devices such as smartphones. Because of the nature of mobile phones, which feature mobility and portability, LBSs have been forecasted as one of the "killer applications" since the beginning of mobile communication. Despite such prospects, the predecessors to smartphones, known as feature phones, failed to gain popularity for a number of reasons, including lack of technical support over the platform, lack of resources such as mobile handset batteries, communication speed and storage devices, and lack of infrastructure support for LBSs. However, feature phones have always been recognized as potentially successful services. The recent explosive growth and popularization of smartphones have made LBSs at the forefront of the market as one of the 'successful killer applications in the market', rather than a 'potential killer application'. To provide LBS, 4 or more coordinates are obtained at the receiving point using GPS, the most commonly used location-tracking device. Typically, satellite-based approaches provide accuracy to the level of tens of meters, while a Differential Global Positioning System (DGPS) provides accuracy to the level of a few meters. Meanwhile, approaches using communication methods that were quickly popularized with the advent of ubiquitous eras, such as Radio-frequency identification (RFID), Ultra-wide band (UWB), or Bluetooth, are also rapidly developing. Specifically, UWB is being used primarily for real-time location tracking systems, where the accuracy of the location is significant because of its advantage in providing high accuracy of up to a few centimeters in indoor position tracking. With these remarkable location-tracking technologies, a variety of application services involving the location of the user has emerged. These services include providing the location of the taxi drivers who are closest to the customers who called for a taxi cab, the so-called 'peace of mind' service that periodically reports children's locations (for crime prevention), and a service that locates friends who are within a 500-meter radius, where the friends are drawn from a contact list in the smartphones.

Various studies have considered a broadcasting environment to handle the recent explosive growth in smartphone users' query congestion and to protect users' personal information, such as locations and interests [1,2,3,4,5,6,7]. Wireless data broadcasting transfers a large amount of information through the push method without processing individual users' query requests. Thus, we can support high quality services even in environments that are concentrated with many users, such as schools, parks, or train stations.

Additionally, to allow users to conduct desired query processing without exposing their location to the server, privacy is provided. That is, the server pe-

riodically transfers information on buildings located at a specific place or moving objects through wireless broadcasting channels. For example, customers entering a large bookstore can use their smartphones to periodically find specific information regarding the books on their areas of interest, such as new books on a certain topic and the locations, titles, and content of books. In addition, a variety of services that consider the users' locations can be offered through wireless broadcasting environments, including weather information, mobile advertisements, traffic information, and the local news. As an example, mobile communications companies, such as AT&T and Verizon, provide location tracking services to keep their family and friends safe by tracking their real-time location to customer using any smart-phone by broadcasting methods. Broadcast dissemination has also been adopted by Microsoft Smart Personal Objects Technology (SPOT) to send timely, location-aware information to customers via the DirectBand network[15].

A wireless data broadcast can be viewed as "storage on the air" and saves power on the client side by avoiding power consuming uplink transmissions [9,5,4]. However, one of the limitations of the broadcast model is that it restricts data access to be sequential. Queries can only be fulfilled after all the required on-air data arrives [10]. Therefore, various studies that strengthen the advantages of broadcasting environments and support more efficient LBS are necessary.

Indexes are used for efficient spatial query processing in LBS. Indexes can be classified between 'disk-based' and 'air' indexes [10,11,12]. Generally, a disk-based index is used in point-to-point methods, where the server receives and processes the clients' requests in an on-demand environment. In other words, the server searches for the query position and objects that are requested and then transfers the results to the requesting client. In contrast, an air-index is used in the push method, where the server transfers the broadcast programming information and the actual broadcast contents in sequence via wireless broadcast channels and the clients use the received index data to selectively tune to the necessary broadcast. Studies regarding spatial query processing in an air-index environment that can simultaneously handle requests from many users and also consider the privacy of the individual users are underway [11, 13,14,15,16,17]. Disk-based indexes have the following characteristics:

- An index search at the time of the query request begins at the index starting point. Taking R-tree as an example, the index search always begins at the root, which is the topmost node.
- Depending on the characteristics of the random-access media, the occurrence of backtracking during the index search does not have a significant effect on the search time.
- A partial search of the index is possible. Taking R-tree as an example, a selective search is possible, which is limited to the potential locations of the query result.

Compared to disk-based indexes, air-indexes face three main issues:

- Probe wait[1] issue: Depending on the times of the query requests and the index transfers, the starting time for the index search can vary. Typically, the clients stand by at the index start point and then begin the index search.
- Backtracking issue: Backtracking of the index search with a tree structure has a significant impact on the query processing time. To hear again the past index data in the broadcasting environment, it is necessary to wait until the next period.
- Index search issue: Repeating the awake and sleep modes and waiting until the desired data are transferred through the channel must be performed, according to the sequential approach.

Therefore, novel indexed structures and search algorithms should be devised to overcome the problems in wireless broadcasting environments and process queries efficiently. This paper proposes a new hierarchical spatial index, called HGI (Hierarchical Grid Index), and a spatial query processing algorithm that exploits the objects' locations and distribution data.

We investigate the research problem of processing spatial queries in wireless broadcasting environments, which is useful in heavy-loaded location based service systems. HGI has a small cost in terms of tuning time and access time by using object ID encoding instead of pointers from Hilbert Curve-based indexing techniques. As a result, the data broadcast server transfers the object ID according to the grid order, which can be used to further prune unnecessary examinations of objects. HGI eliminates the partial replication pointers in multiple places in the broadcast channel that may incur redundant tuning time and access time, thus enabling rapid and accurate spatial search by reducing redundant data.

Moreover, we try to optimize LBS users' access time and tuning time with two different approaches Top-Down Search (TDS) and Reduction-Counter Search (RCS), respectively. TDS provides the optimum access time, whereas RCS offers the optimum tuning time.

The main contribution of this paper is with respect to the three main issues described above:

- Probe wait issue: The proposed index has the fully distributed non-redundant light-weight structure in order to reduce the probe wait time and to support the selective index search. Clients can determine the distance between the objects or the objects' locations through the object identifications, which consist only of the location data of the objects. Therefore, an efficient data search is supported by a quick index search and short information retrieval cycle.
- Backtracking issue: our proposed index overcomes the problem of backtracking through the server, setting the broadcasting order with a consideration for the object's location. Our Top-Down Search algorithm (TDS) provides fast query processing with minimum search costs by utilizing

[1] The average duration for getting to the next index segment is called the probe wait [5].

a location-based data transfer order that is transferred from the server. Meanwhile, our Reduction-Counter Search algorithm (RCS) utilizes a hierarchical structure of HGI to selectively tune to the desired data, providing energy-efficient query processing.
- Partial index search issue: Our HGI utilizes the hierarchical object identification (*oid*) based on the location of the object so that the object's geographical location can be determined just with partial index data. Therefore, a fast spatial query processing is possible through the search of the object's location without the probe waiting regardless of when the client awakes.

Furthermore, our performance evaluation shows the superiority of our solution compared to previous work related to air indexing.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 discusses the motivations for our proposed solutions. Section 4 introduces our system architecture. Section 5 describes our HGI spatial index structure. Section 6 describes our spatial query processing algorithms. Section 7 gives our performance evaluation. Finally, Section 8 concludes.

2 Related Work

In this section, we discuss existing techniques that can be used for supporting spatial queries in wireless broadcasting environments. We also point out their limitations. First, we examine the indexing techniques for reducing query response time and supporting efficient data search. Second, we discuss data broadcasting methods. Third, we examine various spatial query processing algorithms.

2.1 Indexing Techniques

There are two basic methods to deliver data to clients such as pull and push. In the pull method, all requests are explicitly made to the server. The server is responsible for processing the query and returning the answer directly to the client. On the other hand, in the push method, the server periodically transfers the data that a client requests through broadcasting channels. At each period, the index can be inserted with the transferred data to support the client's selective data tuning. However, because the client is unable to process queries during the time prior to tuning to the index data (i.e., the probe wait time), a solution is required. The $(1,m)$ index [5] proposes a method whereby the index is broadcast m times during a single broadcast cycle, i.e., the broadcast index is broadcast every fraction $\frac{1}{m}$ of the broadcast cycle. Selective tuning is accomplished by multiplexing an index with the data items in the broadcast. In general, the fastest access time[2] in a broadcast cycle is obtained when there

[2] Access time is the time elapsed from the time a client requests data to the point when all the required data is downloaded by the client [4].

is no index, since the size of the entire broadcast cycle is minimized at the expense of tuning time[3]. In this case, the average latency time is $\frac{N}{2} + d$, where N denotes the size of the transferred data objects and d denotes the download time for data objects. On the other hand, increasing the number of index segments in a single broadcast cycle reduces the average *probe wait* time, but increases access time because of the additional index information. In this case, the probe wait time for the next index is equal to $\frac{1}{2} \times (i + \frac{N}{m})$ and the wait time for the desired data object is equal to $\frac{1}{2} \times (N + i \times m)$, where i denotes the size of the index. Since the access time is the sum of the probe wait time for the index and the wait time for the desired data object, the average access time for the (1, m) index is equal to $\frac{1}{2} \times ((m + 1) \times i + \frac{1}{m} + 1 \times N) + d$ (see [5] for the details).

However, the (1,m) index has a serious disadvantage. Although copying multiple indexes and transferring to a single broadcast cycle reduces the probe wait time, the entire broadcasting cycle increases because of the inserted index, thus increasing access time. In [5], the authors propose a distributed index that improves over the (1,m) technique. Instead of transferring the complete index for the entire transfer of data, only partial indexes are transferred, thus saving data transfers. However, because the same information is still repeated in the index structure, the broadcast cycle can increase and the time to start query processing gets delayed. In [18], the authors address the issues of multi-attribute based queries on wireless data broadcast channels. They propose three different power conservative indexing techniques, such as index tree, signature, and hybrid. They discuss two simple multi-attribute queries: conjunction and disjunction. In [19], the authors also deal with broadcast-based spatial index. In this paper, the authors propose techniques for scheduling a spatial index tree for broadcast in a single and double channel environment. The algorithms executed by the clients aim at minimizing latency and tuning time. In [20], the authors address the trade-off between access latency and tuning time. To improve the efficiency of energy consumption on mobile devices, they present a parametrized index scheme, called exponential index, to meet different application requirements in terms of access latency and tuning time. The distributed property of the exponential index enables a search to start quickly from an arbitrary index table in the broadcast. The access latency and tuning time of the exponential index can be adjusted by two tuning knobs: index base and chunk size.

The techniques discussed above still cause search delay resulting from the probe wait time and additional index data. Therefore, a solution is required that can reduce the size of the index and simultaneously perform query processing as soon as the client awakes.

[3] Tuning time is the amount of time spent by a client listening to the channel. This allows determining the power consumed by the client to retrieve the required data [4].

2.2 Wireless Data Broadcasting

Delivering data through a broadcast channel provides simultaneous access by an arbitrary number of mobile users and thus allows efficient usage of scarce bandwidth. In [1], the authors propose a system architecture, called Broadcast Disks, that exploits the relative abundance of downstream communication capacity in asymmetric environments. The central idea is that the servers exploit their advantage in bandwidth by broadcasting data to multiple clients. Groups of pages, such as hot and cold groups, with different broadcast frequencies are multiplexed on the same channel. Then, items stored on the faster disks are broadcast more often than items on the slower disks. In [21], the authors introduce the Broadcast on Demand (BoD) model to provide timely broadcast according to user requests. The goal is to maximize performance with respect to satisfaction of deadline constraints and achieve efficient use of available bandwidth. In [22], the authors investigate how to efficiently generate the broadcast schedule in a wireless environment. They consider the access pattern formed by all the requests where the data dependency and access frequency can be represented by a weighted directed acyclic graph. Then, they propose heuristic methods that can be classified according to the underlying strategies: level-oriented and greedy. In [23], the authors notice that in general, clients are grouped based on location, with the members of each group having similar demands. Then, they propose a mechanism that exploits locality of demand in order to increase the performance of wireless data broadcast systems.

The techniques discussed above focus on efficient data transfer by analyzing users' requests. However, none of them consider LBS, as we do in this paper.

2.3 Location-based Spatial Query Processing

In this section, we describe the different methods for spatial query processing.

In [32], the authors introduce a new data structure, called SD-tree, to preserve network connectivity and distance information for the duration of a query. Then, the authors propose a Continuous Mobile Network-Distance-based Range query algorithm to support continuous range query processing with a large number of moving POIs in metro road networks. In [33], the authors present a R-tree-based index and query processing method for supporting geographic information systems. The authors introduce the multi-dimensional index structure to support efficient query processing through the grid processing of data. The proposed method reduces the spatial overlap and improves query efficiency by using the cross-indexed file structure that is free from the database system. In [24], the authors propose a branch and bound R-tree [4] traversal algorithm to support NN (Nearest Neighbor) queries. They discuss

[4] The R-tree is a classical spatial index structure. The basic idea is to approximate a spatial object with a minimal bounding rectangle (MBR) and to index the MBRs recursively [25, 26]

metrics for optimistic and pessimistic search ordering schemes. They also extend their work to deal with kNN queries.

In [27], the authors propose a sharing-based nearest neighbor query model, called MAPLE, which is designed for sharing of query results that are cached in the local storage of mobile clients. In [14], the authors address the CkNN (Continuous k-NN) search problem in wireless broadcasting environments. They discuss the issues in adopting a spatial air index in wireless data broadcast systems and propose a search algorithm to support CkNN search based on the Hilbert Curve index. The authors propose a query window partitioning strategy to improve the spatial locality of the Hilbert curve and three index organizations to facilitate the processing of different queries. In [28], the authors propose a spatial index, called Distributed Spatial Index (DSI) to support location-based queries in wireless data broadcast systems. In [36], the authors extend the previous work in [28] to develop indexing and query processing algorithms to answer complex queries, such as window queries and NN queries. The authors provide an analytical model to study the performance of a primitive search algorithm, namely Energy Efficient Forwarding (EEF), in both error-free and error-prone environments. DSI has a distributed structure that mixes multiple search paths into a linear index structure that is distributed into the broadcast cycle. In DSI, a pointer of each data instance is repeated as many times as the number of entries of an index in a broadcast cycle to facilitate multiple search paths. However, DSI increases access time because of the load from the duplicated pointers that contains the next upcoming frame. Moreover, DSI increases the search cost since Hilbert Curve-based indexing techniques require a mapping procedure (i.e., to map from Hilbert Curve values to real coordinates of points) in order to obtain real locations of the data objects. Indeed, the client suffers from probe wait time for referring to exponential pointers in broadcasting environments.

In [29], the authors discuss the effects of different broadcast organizations on search performance and challenge the traditional use of Depth-First organization. They address the problem of exact kNN search on R-trees in wireless broadcasting environments. They propose a technique that improves the tune-in time of kNN search and discuss the tradeoffs involved in organizing the index on the broadcast medium. In a previous paper [12], we proposed a novel broadcast-based spatial data dissemination and selective tuning scheme, namely ESS (Exponential Sequence Scheme), which provides clients with the ability to perform selective tuning and assists in reducing the client's tuning time. The basic idea is to use exponential pointers from each data item. The exponential pointer facilitates the index replication by sharing links in different search trees and enables a search to start quickly from an arbitrary index table in broadcasting environments [20].

With the exponential pointer, each data object contains pointers that contain the IDs, locations, and arrival times of the data items that will subsequently be broadcast. Each client utilizes an exponential pointer from each data item for the purpose of reducing energy consumption. In [30], we proposed an algorithm to support dynamic, continuous nearest neighbor queries in

wireless broadcasting environments. To enable clients to find the exact answer for moving queries, we defined the GR (Guaranteed Region), which divides a query line segment into disjoint lines where the nearest neighbor of any point inside a GR is the same.

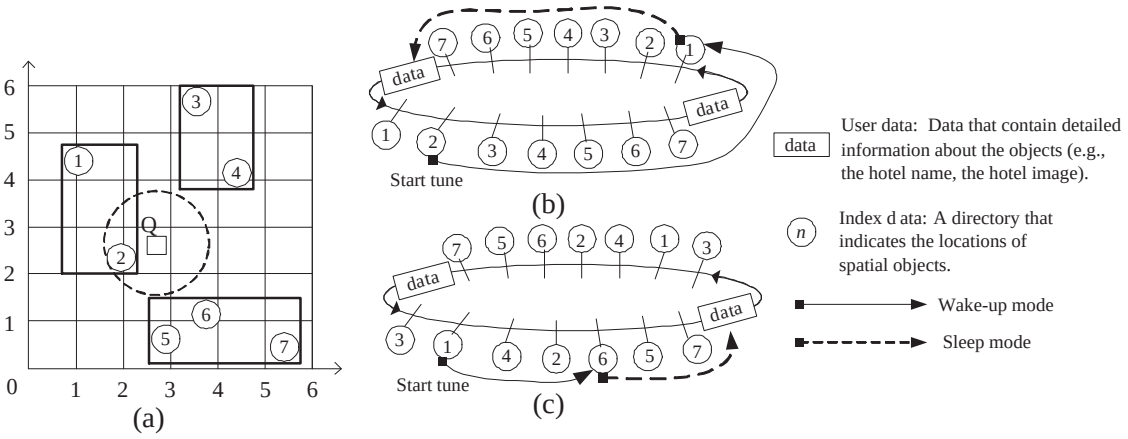
The works discussed above focus on spatial query processing to support LBSs. However, they do not provide treatment of both the index structure and the data transfer schemes to reduce the clients' energy consumption and improve query processing time.

3 Motivations

In this section, we discuss the influence of access time and tuning time with an index structure and a broadcasting order in broadcasting environments.

Suppose that only data that are indexed are transferred through the single broadcasting channels. Let N be the size of the transferred data. Then, the access time and the tuning time for obtaining the desired information can be expressed as access time= $\frac{N}{2}$ and tuning time=$\frac{N}{2}$, respectively. In this case, an ideal access time can be guaranteed because the time for waiting and reading the index is not necessary. However, a substantial amount of energy is consumed since the clients remain in awake mode until obtaining their desired information. Therefore, selective tuning can be provided through transferring the index along with the data, which allows for the identification of the data type being requested and when the data will be transferred. As mentioned before, we refer to the time taken to read and transfer the index and data together as the 'probe wait'. We also refer to the time taken to read the index and obtain the desired data as 'bcast wait'. Let the size of the transfer index be '*index*', the probe wait is then $\frac{N+index}{2}$ and the bcast wait is $\frac{N+index}{2}$. Because the access time for obtaining the desired data is probe wait+bcast wait, access time becomes $N+index$. In this case, the most ideal tuning time can be guaranteed because the client can understand the desired data by reading the index so that the client needs to be awake and tuning only at the time when the data are transferred. However, the probe wait and bcast wait require too much time for the client to obtain the desired data.

Figure 1 illustrates this scenario with an example. In Figure 1(a), 7 objects are expressed as 3 minimal boundary rectangles (MBRs). Assume that the client awakes at a random time for a Nearest Neighbor (NN) query and begins tuning from the second index data. Assume that the tree-based approach transfers the index in the order 1, 2, 3, 4, 5, 6, and 7, while the location-based approach transfers the index in the order 3, 1, 4, 2, 6, 5, and 7, from top to bottom, according to the location of the objects. Let x be the amount of data transferred during a single broadcast cycle, x' the amount of data from the starting point of the data transfer to the point where the desired data arrive, i the number of index entries, and j the number of data items. In Figure 1(b), the tree-based index transfer technique requires tuning to all of the index data (i.e. a complete index data) for accurate spatial query processing. Thus, an



(a) Objects with MBR (b) Tree-based index broadcast (c) Location-based index broadcast

Fig. 1 Comparison Between Location-based and Tree-based Data Transfer Methods to Retrieve Data in Wireless Broadcasting Environments

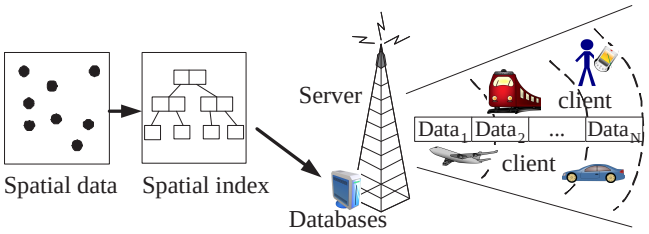


Fig. 2 Our System Model

NN result of object 2 can be identified only after tuning to 6 index entries (i.e. index entries 2, 3, 4, 5, 6, 7) and data items, and then tuning to index 1 in the next cycle. Thereafter, the client converts to sleep mode and awakens at the time when the data arrives, to finally obtain the desired data. Accordingly, access time= $i\times13+x+x'$ and tuning time=$i\times7+1\times j$. In contrast, in Figure 1(c), the location-based index misses the first index, which was 3. However, even if the tuning began with index 1, we could confirm, after tuning to index 6, that the remaining data 5 and 7, which are more distant from the Y-coordinate, are not the targets of the query result. This scenario results from having an index structure that transfers sequentially according to the object's location. Thereafter, the client can be certain that the final NN result is 2 even without tuning to the remaining index data in the next cycle. Therefore, it converts to sleep mode until the time that the data arrives without additional index data and is then awake when the data arrive, to receive the desired data. Accordingly, access time=$i\times6+x'$ and tuning time=$i\times4+1\times j$.

Therefore, to support rapid query processing using an index in a broadcasting environment, the client should be able to perform spatial query processing

solely by waking up at a random time and reading the partial index data. In addition, the ideal spatial query processing index should support selective tuning only through the data that are transferred without a probe wait. The objective of this paper is to provide optimal access time and tuning time for spatial query processing in broadcasting environments.

4 System Architecture

This section describes our basic system model, assumptions, and components for spatial query processing.

The system model in Figure 2 distinguishes between three major entities: server, client, and data.

Server: The server transfers the data to a random number of clients through public channels that are allocated for data broadcasting. To perform this task, the server gathers and analyzes the locations of the objects on the map and transfers signals to the clients through wireless broadcasting channels. A stream of data that is transferred through the broadcast is called the broadcast stream, and the server repeatedly broadcasts the broadcast stream. The server identifies and analyzes the locations of the objects on the map, then structures them into indexes. Thereafter, the data transfer program is formed for transferring data to the clients through wireless broadcasting channels.

Client: The client selectively receives the desired data on the broadcast channels without any request made to the server for specific data. Then, the client performs spatial query processing, using the received index. The client can be classified according to the characteristics of the resources. For example, minimizing the limited battery consumption in LBSs may be an important issue for mobile terminals such as smartphones, whereas energy consumption may not be a significant issue for a navigation system installed on a vehicle. In this case, rapid query processing is more important than anything else for accurate query results, depending on the characteristics of the fast-moving users. Based on the relevant resource characteristics, the clients are classified into the following two groups:

- Limited Resource Device (LRD): awakes at the time when the desired data arrive and tunes to the data but is asleep (sleep mode) the rest of the time.
- Adequate Resource Device (ARD): in a standby state to tune to the data (awake mode).

In other words, LRD converts from a standby mode (sleep mode) to a tuning mode (awake mode) for query processing to tune to the index that is transferred from the server through the wireless broadcasting channels. After using the index data to identify the time when the desired data will arrive, it only awakes again when those data arrive and tunes to the data. ARD, however, does not take into account the constraints of the battery limits and continues to stay awake to tune to the data that are transferred from the server until the desired data arrive through the broadcast channels. Thus,

LRD tunes to the data with consideration for both access time and tuning time simultaneously, while ARD tunes to the data with consideration only for access time. In this paper, the client's mobility pattern follows the Random Waypoint Mobility Model [31], which is widely used.
Data: Data that are transferred from the server can generally be classified into index data and user data, as follows.

- User data: Data that contain detailed information about the objects (e.g., the hotel name, the hotel image, a reservation status, or pricing information with respect to information concerning other hotels).
- Index data: A directory that indicates the locations of spatial objects in a hierarchical structure. Data that contain information such as the objects' coordinate information, the amount of data, and the transfer time for the user's data.

The broadcast cycle is composed of N, the amount of the user data, and m, the number of indexes. The lowest level (leaf level) of the index, which has a hierarchical structure, is composed of pointer data that include the real information for the specific object.

5 HGI Spatial Index Structure

In this section, we introduce HGI, a hierarchical index structure that considers the broadcasting environment.We start by describing the basic structure of HGI. Then, we introduce how to construct HGI based on the distribution of the objects.

Unlike traditional indexes, HGI forms an index only with an identification of the existing objects. Thus, it provides an efficient search environment by removing unnecessary formation data. Additionally, it provides a linear order of data objects for broadcasting while maintaining maximal locality among nearby data objects [28]. To this end, HGI represents spatial objects in the form of a grid according to the objects' locations and distributions. The objects' identification numbers, through grid division, are used as references for the locations of the objects. The first region is the highest root grid. When the entire region is considered as a square shape, it is quartered by a cross shape and divided into root grids 'R_0, R_1, R_2, and R_3' while making a 'z' from the top left side as the basis (see Figure 3). Then, additional divisions are made into child nodes through the repetition of a quadrisection, with the same rule as that for the root grid, according to the objects' distribution until only the final object is included. The regional division is influenced by the distribution of the objects. As can be seen in Figure 3, although grid 10 in R_1 and grid 33 in R_3 both contain 2 objects, the grids are divided into 4 and 7 grids, respectively, to accurately display the objects' locations according to the locations of the objects and the distances between the objects. The object information that corresponds to a leaf node represents the pointers to data tuples containing information such as a physical location, name, and price,

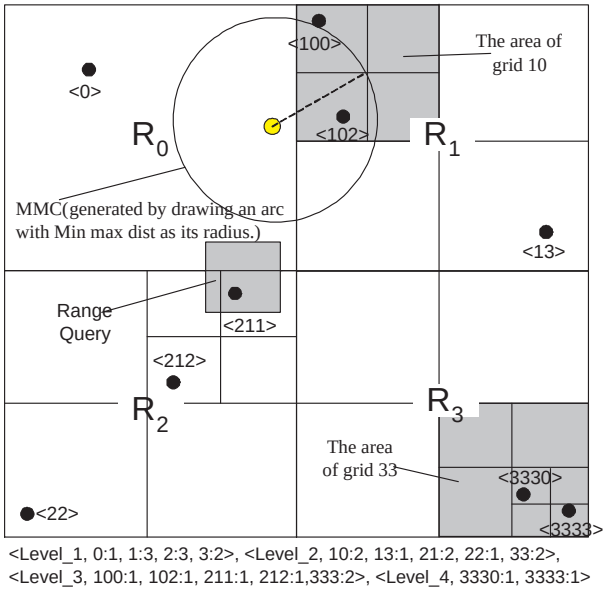


Fig. 3 Spatial Query Processing Using HGI

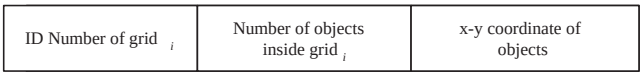


Fig. 4 HGI Basic Structure

etc. Each object is assigned a unique identification number according to the location and distribution of the surrounding objects. For the entire set of data of size N, Figure 4 shows the structure of HGI.

The size of the index can be expressed as the number of grid nodes with respect to the entire region+the number of objects. Let p^n the child node for the root grid node R_n, k^n the number of grid nodes belonging to R_n, and pID the grid identifier ID that is included in R_n. Let $\Re_{nn}$ denote the number of objects included in grid nodes that belong to the root grid R_{nn} and k'^n the number of nodes in R_n excluding the leaf grid. Then, we have the following expression:

$$Index\_size = (\sum_{p^0=0}^{k^0} pID^0 + \sum_{p^0=0}^{k'^0} \Re_{n0}) + (\sum_{p^1=0}^{k^1} pID^1 + \sum_{p^1=0}^{k'^1} \Re_{n1})$$
$$+(\sum_{p^2=0}^{k^2} pID^2 + \sum_{p^2=0}^{k'^2} \Re_{n2}) + (\sum_{p^3=0}^{k^3} pID^3 + \sum_{p^3=0}^{k'^3} \Re_{n3})$$

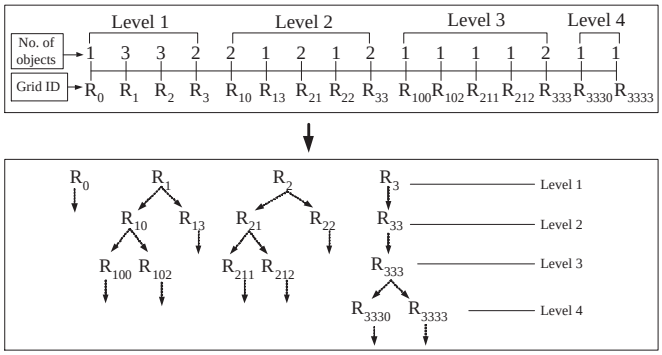


Fig. 5 Hierarchical Grid and the Transfer of a Number of Objects in the Grid

$$= \sum_{R_n=0}^{3} (\sum_{p^n=0}^{k^n} pID + \sum_{p^n=0}^{k'^n} \Re) \qquad (1)$$

Let us now give the division process of HGI, which depends on the distribution of the objects. In Figure 3, the root grid, R_0, is order of grid 1, in which one object is present; therefore, an additional division does not occur. By contrast, an examination of R_1 shows the order of grid 2 (grid ID 11, 12, 13) and 3 (shaded area of R_1) to be such that a total of 7 divisions into child nodes occur. The digits in the grid identification number represent the order of the grid: 4 digits in the identification number mean a grid order of 4, with a 4-level division from the highest root grid 0 to the leaf grid 3 (see objects 3330 and 3333 of R_3 in Figure 3). As such, each of the regions can have different orders according to the distribution of the objects in the HGI, and the clients can verify the exact location and distribution of the objects using only the objects' identification number during each object search. Furthermore, the clients' selective tuning is supported without additional information in a sequential index structure required to express the object's location. We discuss the hierarchical search algorithm in the next section. To further explain this algorithm and using Figure 3 as an example, R_1 and R_3 were excluded from the search range, while the query regions and non-overlapping regions (e.g., 20, 22, and 23 from R_2) were also excluded from the search range in R_2 during the range query processing.

Figure 5 hierarchically shows the sequential grid ID and the number of objects included in the grid (see the grid structure of Figure 3).

The following defines the HGI data structure.

Definition 1 *(Hierarchical Grid Index) In a given set of data regions, grid G_i repeats the hierarchical division process until only 1 object is included. The HGI tree has an unbalanced tree structure, and the leaf node grid contains a single object.*

The following defines the number of objects contained in G_i for an HGI.

Definition 2 *(Number of objects inside G_i) Let num_obj_G_i be the number of objects inside G_i, which has child nodes and num_obj_G_i' be the number of objects inside G_i', which has not child nodes. Then, num_obj_$G_i \geq 1$ and num_obj_G_i' ≥ 0.*

HGI provides a partial search of the index for spatial query processing via wireless broadcast channels. With HGI, since the data objects broadcast by the server are sequentially ordered based on their locations, it is not necessary for the client to wait for the beginning of the next index segment. Even the client starts to tune an index in the middle of a broadcast cycle, it is not necessary for the client to wait for the beginning of the next broadcast cycle. Therefore, HGI provides a fast spatial query processing without the probe waiting regardless of when the client awakes.

6 Spatial Query Processing

This section presents our solution to spatial query processing using HGI, based on two algorithms. The first algorithm is Top-Down Search (TDS), which is targeted for ARD. TDS transfers the object data from the top to the bottom according to its location, thereby minimizing the index size and reducing query processing time. The second algorithm is Reduction-Counter Search (RCS), which is targeted for LRD. Although there is an additional transfer of data based on the number of objects contained in each grid, TDS supports the clients' selective data tuning, which in turn reduces tuning time. The specific index search method is as follows.

1. ***TDS*** (provide the optimal access time): the client processes the queries using the minimum identification information regarding the data objects.

- The server transfers the object identification numbers according to the grid order rather than the objects' coordinate information. The order of the transfer is in top-bottom direction from the object location as the basis. For example, in Figure 3, the information is transferred in the order 100, 0, 102, 13, 211, 212, 3330, 22, and 3333. In particular, if the heights are equal, the object on the left is transferred first.
- Regardless of the hierarchical structure of the HGI grid, TDS transfers the identification number of the lowest level grid that contains the objects according to the objects' location. Therefore, the data are not transferred by the grid levels in the HGI; rather, the grid identification numbers at various levels are transferred simultaneously according to the location and distribution of the objects.
- Selective tuning is possible for the clients via the object identification number information that is received from the server. Consider Figure 3 for instance. During a range query, the client is able to discover that the final query result is 211 after tuning to the object data (up to 211), that is, tuning the remaining object data to 212, 22, 3330, and/or 3333, which are outside of the y-coordinate range (we note that the objects are sorted by

y-coordinates or x-coordinates based on the analysis of the distribution of the objects on the map), is unnecessary.
- Because TDI transfers data according to the location of the objects, the data can be obtained quickly without additional index information. In contrast, this can incur unnecessary energy consumption because the client must be in the awake mode until the object that can confirm the query result arrives.

2. ***RCS*** (provide the optimal tuning time): the client processes the queries using the information on identification number and the number of objects in the grids with respect to the data objects.

- RCS includes the number of objects that are contained in each grid to support the clients' selective data tuning.
- According to the hierarchical structure of HGI, the server transfers the grid identification number with the number of the objects included in each grid's 'Z' sequence.
- If the last digit of each level is 1, the child node does not remain, and search can proceed until the end of the level reaches 1. This structure supports selective choice. Consider Figure 5. Because the grid number 0 (R_0) at Level 1 had a value of 1 as the number of objects, we know that additional objects will not be found at Level 2. In contrast, because the grid number 1 (R_1) has objects in numbers 10 and 13, we know that additional data are present at Level 2.
- The coordinate data of the objects are transferred in batches at the end, after the Hierarchical sequential grid data.
- As for the search method, the search reaches the end when the number of objects in the grid contained in the Minimum/Maximum/distance Circle (MMC: generated by drawing an arc with Min max dist as its radius. The Min max dist is the distance to the furthest point from the grid that contains the objects based on the query point.) becomes 1 and satisfies the threshold value, which is the sum of the grids that overlap with the MMC (T_{min}: the value of the predefined threshold for adjusting the selective tuning tolerance boundary). For example, when $T_{min} \leq 4$, if the number of objects in grid 102 that are included with the MMC becomes 1 and the sum of the number of objects in grids 0, 100, and 102 that overlap with the MMC is smaller or equal to 4, as shown in Figure 3, then the search ends. Subsequently, the client finds the final query result within 0, 100, and 102. Because the number of objects indicates that the objects are not present in the remaining overlapping regions of 103 and 12, they are excluded from the search targets.

6.1 Top-Down Search for ARD

In this section, we describe our spatial query processing algorithm for ARD. The order of transfer for TDS is determined according to the location of the

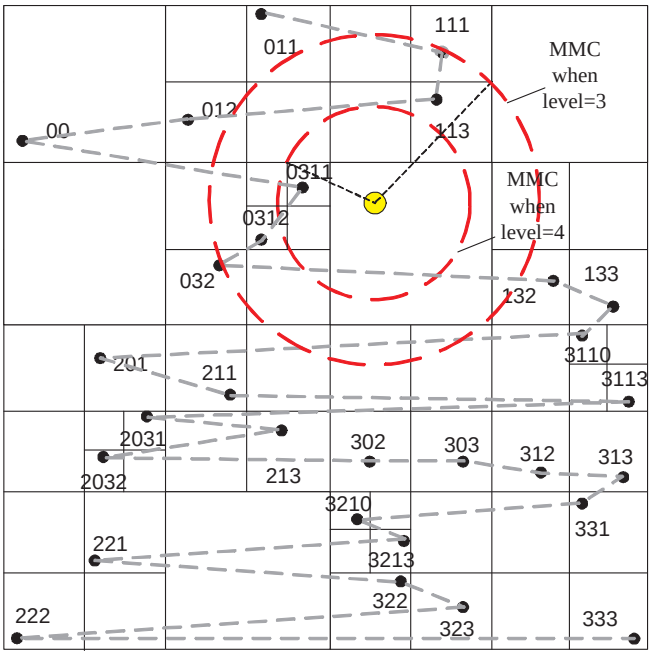


Fig. 6 The order of Transfer for TDS

objects, regardless of the hierarchical structure of the HGI grid. Depending on the location of the objects, the transfer is sequential from the object located at the highest region to the object located at the lowest region. For the objects at the same height, the object located on the left is transferred first.

In Figure 6, the order of data transfer for TDS is as follows: 011, 111, 113, 012, 00, 0311, 0312, 132, 133 ... etc. The client identifies the object ID numbers that are transferred from the server, to determine the location and the size of the grid (grid level) that contains the objects. Then, a Minimum/Maximum/distance Circle (MMC) is generated by drawing an arc with Min_max_dist as its radius. The Min_max_dist is the distance to the furthest point from the grid that contains the objects based on the query point. The respective MMCs are shown when the grid levels are 3 and 4. Among the objects for transfer, the objects that are located at the bottom of the MMC (e.g., 3110, 201, 211.., 3113..., when the MMC level is 4) or are not contained in the MMC are excluded from the search target. As shown in Figure 6, the client tunes the object information to 133, and because the coordinates for object 3110 that were transferred thereafter are located lower than the MMC (top-down search), the client stops tuning and performs the final query processing. After confirming the coordinate data of the objects that belong in grids 113, 0311, and 0312, which are included in or overlapped with the MMC, the final query result is 0311.

Definition 3 *(Final NN decision for TDS) While the data objects are sequentially broadcast in horizontal order, that is, from the top coordinate to the bottom coordinate, if the y-coordinate of the top of grid of G_{i+n} is lower than MMC, then G_{i+n} and the rest of the broadcast data objects are located outside of the NN range.*

TDS can improve data access time by performing query processing with the data according to the object location and hierarchy without using additional information. The TDS NN search algorithm is the same as in Figure 7.

```
Algorithm for NN Search of TDS
Input: locations of query point and the ID number of grid G_i
Output: NN
1:At the initial stage, n=1, set of NN_candidate={G_i}, and MMC be MMC of G_i. (Step 1)
2: for each object inside G_i ∈ S
3:    read ID number of G_{i+n}
4:       if y-coordinate of the top of grid of G_{i+n} is lower than MMC
5:          stop tuning
6:       if grid of G_{i+n} does not intersect or included in MMC of G_i (Step 3)
7:          G_{i+n} cannot be NN-candidate (Step 5)
8:       else
9:          set of NN_candidate ∪ G_{i+n} (Step 4)
10:            if the radius of MMC of G_{i+n} is smaller than MMC (Step 7)
11:               MMC <= MMC of G_{i+n}
12:      increase the value of n
13:   find NN from the set of NN_candidate (Step 2 and Step 6)
14return NN
```

Fig. 7 Pseudo-code of TDS

We now describe the steps taken by the client to process the NN query with TDS.
Step (1): The ID number of object G_i is read, and the MMC is drawn with the grid that contains G_i and the query point as the radius. At the initial stage, n=1, NN_candidate=$\{G_i\}$, and MMC be MMC of G_i.
Step (2): G_{i+n} is read. If G_{i+n} is EOF, then the coordinates of the NN candidate objects are read, and the final query results are determined. Otherwise, go to step (3).
Step (3): If the grid that contains object G_{i+n} is included in the MMC or is overlapped with MMC , then go to step (4); otherwise, go to step (5).
Step (4): NN-candidate$\cup G_{i+n}$ (G_{i+n} can be the possible object for the final result of NN). Go to step (7).
Step (5): G_{i+n} cannot be the final result of NN. If the grid that contains object G_{i+n} is located at the coordinate below the MMC of the G_i, go to step (6); otherwise, go to step (2).
Step (6): Stop tuning. In this case, G_{i+n} and all of the objects transferred afterwards cannot be the final query result for NN. Read the coordinates of the NN candidate objects, and determine the final query result.
Step (7): Draw the MMC of G_{i+n}. If the MMC of G_{i+n} is smaller than the

MMC of G_i, then MMC <=MMC of G_{i+n}; otherwise, MMC <=MMC of G_i. Increase the value of n. Go to step (2).

6.2 Reduction-Counter Search for LRD

We now describe our spatial query processing algorithm for LRD. TDS must continuously tune to the data in awake mode until obtaining the final query result for query processing. Therefore, although partial selective tuning is possible using MMC, a more effective search algorithm is necessary to optimize energy efficiency. RCS supports the client's selective data search using a pruning algorithm through a hierarchical search. Furthermore, a selective spatial query is handled using the number of objects present in each grid region.

NN query processing is as follows.
The variable Min_max_dist, which is the largest distance among the grids that contain the objects that are closest to the query point, is used as the radius to draw the MMC. Locations not included in the arc are excluded from the search targets. In addition, selective tuning is performed using the HGI identification numbers and the information on the number of objects that are transferred hierarchically. Figure 3 shows an example of the selective NN query processing using the MMC. The MMC is drawn at the third level with the query point and the furthest point in Grid 102 as its radius. Objects 13, 212, 211, 22, 3330, and 3333, which are not included in the MMC, are excluded from the search range. The NN query processing confirms the possibility of selective tuning and draws the MMC by each level from Level 1 to Level m. Then, the client attempts a selective search using the MMC and the HGI identification numbers as well as the information on the number of objects in the grid. Figure 8 shows RCS' tree structure.

Let us introduce the following definition used for selective tuning with RCS during NN query processing.

Definition 4 *(Selective tuning with RCS) Grid G_i that is neither contained nor intersected with the MMC cannot contain an NN object.*

The following definition ensures the earliest decision time that the client stops tuning and identifies the nearest neighbor candidates.

Definition 5 *(NN candidate for NN query with RCS) While the data objects are sequentially broadcast according to HGI order, if the number of objects inside G_i that are included in MMC is 1, G_i, which completely contains or intersects in MMC, can be the final nearest neighbor candidate.*

As observed in Definition 5, we introduce the following definition for selective tuning with HGI.

Definition 6 *(Selective tuning with HGI) It is not necessary to check whether Grid G_i is contained or intersected with the MMC if x-coordinate of rightmost of G_i < x-coordinate of leftmost of MMC or x-coordinate of leftmost of G_i*

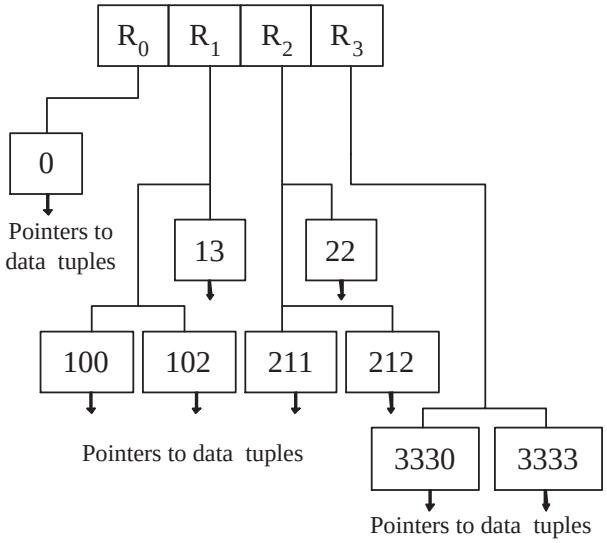


Fig. 8 RCS Tree Structure: Grid-based hierarchical search

> x-coordinate of rightmost of MMC or y-coordinate of bottom of G_i > y-coordinate of top of MMC, or y-coordinate of top of G_i < y-coordinate of bottom of MMC.

Figure 9 illustrates how NN search proceeds according to each level. The MMC is used to divide a pruning region that does not need to be searched (shaded area). Selective tuning is supported by excluding the pruning region from the search range. The pruning process draws the MMC that has the radius of the most distant point within the grid box (where the object is present) that is closest to the query point. Meanwhile, the objects that are out of the MMC range are excluded from the query search target. At this time, the number of objects contained in the grid contributes to the selective tuning by the client (to be discussed later). In Figure 9, 2 objects are assumed to be in grid 12, and 1 object is assumed to be in grid 122. The search comes to an end when the number of objects in the grid contained in the MMC becomes 1 and the sum of the objects in the grid that overlap with the MMC is below the threshold value (T_{min}: the value of the predefined threshold). We note that the example in the figure shows the pruning process after a maximum of 28 objects. However, as the number of objects located on the map increases, the search costs for the unnecessary objects can be reduced more effectively than with the existing index structure.

Completing the search for the RCS occurs when the MMC has a grid that contains only 1 object (Grid 102 in Figure 3). The RCS NN search algorithm used is the same as in Figure 10.

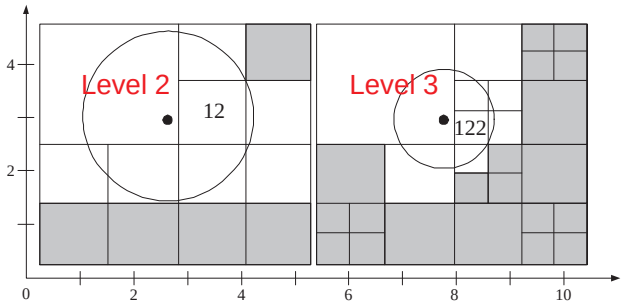


Fig. 9 Pruning Processing with the Minimum/Maximum/Distance Circle (MMC)

Algorithm for NN Search of RCS
Input: locations of query point, the ID number of grid and the number of objects inside the grid
Output: NN
1:At the initial stage, n=1, set of NN_candidate=$\{G_{i+n}\}$, and MMC be MMC of G_{i+n}. (Step 1)
2: **for** each level
3: **for** each grid G_i
4: read ID number of G_{i+n} and the number of object inside G_{i+n}
5: **if** the number of object inside of G_i which is included in MMC is 1
AND No. of objects drops below threshold T_{min}, (Step 3)
6: find the final result according to the value of NN-candidate
7: **if** grid of G_{i+n} does not intersect or included in MMC of G_{i+n} (Step 4)
8: G_{i+n} cannot be NN_candidate (Step 6)
9: **else**
10: set of NN_candidate $\cup$ G_{i+n} (Step 5)
11: **if** the radius of MMC of G_{i+n} is smaller than MMC (Step 7)
12: MMC <= MMC of G_{i+n}
13: increase the value of n (Step 7)
14: find NN from the set of NN_candidate (Step 3)
15**return** NN (Step 3)

Fig. 10 Pseudo-code of RCS

We now describe the steps taken by the client to process the NN query with RCS.

Step (1): At the initial stage, n=1, NN_candidate=G_{i+n}, and MMC be MMC of G_{i+n}.

Step (2): For the first level, read ID number of G_{i+n}, then draw MMC centered at query point q which completely contains G_{i+n}.

Step (3): For each level, read G_{i+n}, which is intersected or included in MMC. Read the number of object inside G_{i+n}. If the number of objects inside G_i that is completely located in MMC is 1 and the number of objects inside G_i that are intersected in MMC drops below threshold T_{min}, find the final result according to the value of NN-candidate. Else, go to step (4).

Step (4): If the grid that contains object G_{i+n} is included in the MMC or overlaps with MMC , go to step (5), else go to step (6).

Step (5): NN-candidate$\cup G_{i+n}$ (G_{i+n} can be the possible grid that contains

the final result of NN). Go to step (7).
Step (6): G_{i+n} cannot be the final result of NN. Go to step (3).
Step (7): Draw the MMC of G_{i+n}. If the MMC of G_{i+n} is smaller than the MMC of G_{i+n-1}, MMC <=MMC of G_{i+n}. Increase the value of n. Go to step (3).

7 Performance Evaluation

In this section, we give a performance evaluation of our TDS and RCS algorithms, based on analysis and experiments. We compare our algorithms with Distributed Sequential Search (namely DSS, a new algorithm that combines the characteristics of both DSI[28] and ESS[12]).

7.1 Performance Analysis

In this section, we analyze TDS, RCS and DSS[5]. We choose DSS that combines the characteristics of both DSI[28] and ESS[12] and each index table has exponential pointers as mentioned in Section 2.3. DSI and ESS are proposed recently to decrease access time and tuning time in broadcasting environments.

DSS uses distributed exponential pointers from each data item. With the exponential pointer, each data object contains pointers that contain the IDs, locations, and arrival times of the data items that will subsequently be broadcast.

Each entry of the index table contains pointers to subsequent data. For example, index no. 0 contains the pointers to subsequent data items 1, 2, 4, 8 as shown in Figure 11. These exponential pointers, from the first tuned index, provide fast access to both nearby and distant data. To identify the object with DSS, the client listens to the current data and follows the pointer to the furthest data that does not exceed the target object.

DSS provides a fully distributed structure that allows query processing to start quickly. However, performance degradation is caused by the redundant structure of the pointers that represent the objects. In contrast, HGI uses a hierarchical tree structure that is only composed of existing object information and non-redundant pointers to reduce the index size and lessen the unnecessary repeated tuning process by the clients.

7.1.1 Access Time for DSS, RCS, and TDS

We now compare the access time of DSS, RCS, and TDS. Because all three algorithms have a distributed index structure, the probe wait time can be ignored.

[5] Even until recently, DSI and ESS have been a representative study of index for supporting spatial query processing in wireless broadcast environments. In this paper, performance assessment was conducted on DSS that assumed the most similar environments as the proposed technique.

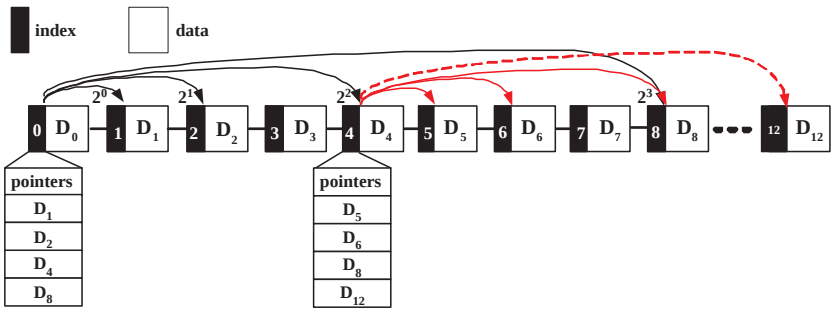


Fig. 11 Distributed exponential pointers for DSS

– DSS: e represents the exponent, N represents the total number of data, and d represents the total number of the next pointer in each index table[6], HC represents the Hilbert Curve order, exp represents the total size of the exponential pointer during a broadcast cycle, where $exp = \sum_{d=1}^{N}(log_e N)$ and oid represents the object ID. Then, the access time for DSS is as follows:

$$DSS = ((HC + oid) \times N + exp + data \times N)/2 \tag{2}$$

$\Re$ denotes the number of objects that are contained in the grid nodes that belong to the root grid R, p^n represents the child node of the root grid node R_n, k^n represents the number of grid nodes that belong to R_n, pID represents the grid identifier included in R_n, and $k^{'n}$ represents the number of grid-nodes included in the R_n (because the number of leaf nodes in $\Re$ is 0, it is not necessary to indicate it, and therefore, the lowest grid node is excluded from the number). Then, the access time for RCS is

$$RCS = (\sum_{R^n=0}^{3}\sum_{p^n=0}^{k^n} pID + \sum_{R^n=0}^{3}\sum_{p^n=0}^{k^{'n}} \Re + data \times N)/2 \tag{3}$$

where p^{n}' represents the leaf grid node of the root grid node R_n[7], $k^{n'}$ represents the number of leaf grid nodes that belong to R_n, and pID represents the grid identifier contained in R_n. Then, the access time for TDS is

$$TDS = (\sum_{R^n=0}^{3}\sum_{p^{n'}=0}^{k^{n'}} pID + data \times N)/2 \tag{4}$$

[6] Each index table in the DSI and ESS has the next pointers that increase exponentially within the range of N, the total r amount of data. For example, if N=1024, a single index table has 9 next pointers of 1, 2, 4, 8, and up to 1024.

[7] The leaf grid node refers to the node that is located at the lowest place in the tree structure. That is, the node does not have any more child grid nodes

– TDS eliminates the probe wait time and processes the spatial queries only with a leaf grid ID.

7.1.2 Tuning Time for DSS, RCS, and TDS

We now compare the tuning time for DSS, RCS, and TDS to prove the energy efficiency of our proposed algorithms during spatial query search. While DSS uses the exponential pointer, which has a redundant structure, our algorithm that uses HGI allows selective tuning with the ID and the number of objects without the exponential pointer.

Let exp' represent the number of indexes from j to N', which the client wakes up and listens to until the desired data are obtained. We have the following:

$$DSS = HC + exp' + data \tag{5}$$

, where exp'=$\sum_{j=1}^{N'}(log_e N)$.

Let L^m denote the number of grid nodes in R_n up to the grid level, m, until the query processing is completed and L^{m*} the number of grid nodes excluding the leaf grid node in the R_n up to the grid level, m, until the query processing is completed. Thus, we have:

$$RCS = \sum_{R^n=0}^{3} \sum_{p^n=0}^{L^m} pID + \sum_{R^n=0}^{3} \sum_{p^n=0}^{L^{m*}} \Re + data \tag{6}$$

The tuning time for TDS is almost the same as the access time; however, only one data point, which is the final query result, arrives through the indexed data. This yields the following formula:

$$TDS = (\sum_{R^n=0}^{3} \sum_{p^{n'}=0}^{k^{n'}} pID)/2 + data \tag{7}$$

7.2 Performance Experiments

In this section, we evaluate the efficiency of our algorithms through their implementation with a Pentium 3.16 GHz CPU. We assume that the client's mobility pattern follows the widely used random waypoint mobility model [31]. A mobile client begins by staying in one location for a certain period of time t. We assume two energy states: doze and active mode. In a processor, the doze mode has extremely low power consumption. We assume that the broadcast channel has a bandwidth of 2 Mbps as the same applied in [34], [35]. We use a real dataset (hereafter called $\mathcal{D}1$ and $\mathcal{D}2$) containing 39,231 data objects

of MBRs for Montgomery County roads and containing the 5922 data objects of cities and villages of Greece, and a uniform dataset (hereafter called $\mathcal{D}3$) with 10,000 points; the former was extracted from the dataset available at www.rtreeportal.org (see Figures 12(a) and 12(b)).

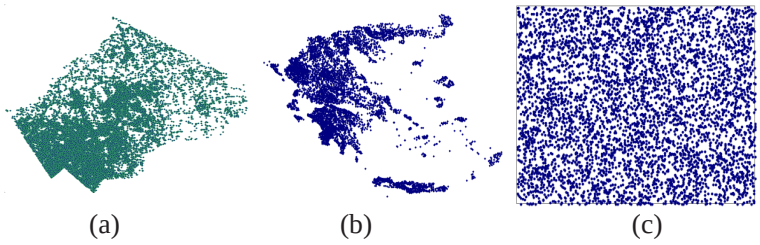
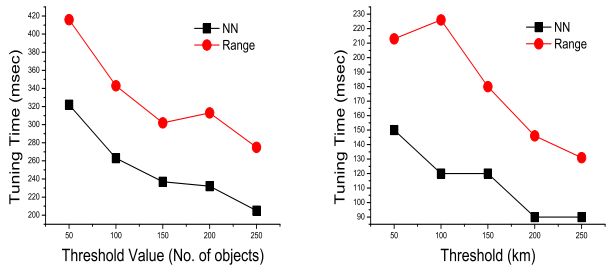


Fig. 12 Datasets for Performance Evaluation. (a) REAL dataset1 ($\mathcal{D}1$), (b) REAL dataset2 ($\mathcal{D}2$), (c) UNIFORM dataset ($\mathcal{D}3$).

For default setting, we assign the 10 bytes for each pointer which also includes two dimensional coordinates. We assign 32 bytes for the spatial index since it needs more space to maintain spatial information and pointers in index. The default parameter setting in our synthetic data set test is: number of objects = 10,000 ($\mathcal{D}3$), Size of data = 128 bytes, exponential value $e = 2^8$, and $T_{min} = 50$.

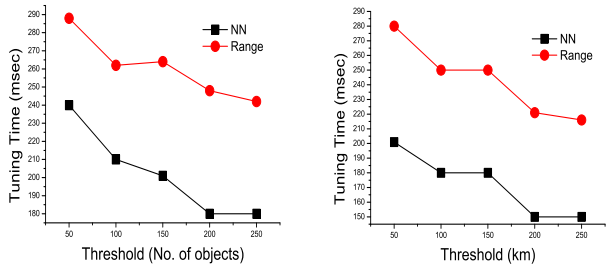
The first experiment (Figures 13(a) and 13(b)) shows the tuning time results following an increase in the threshold value when processing the NN queries and the range queries using Reduction-Counter Search (RCS) in $\mathcal{D}1$. The data size is assumed to be 128 bytes. The client-set threshold value has a

[8] The exponential value allows indexing pointers to be exponentially increased at any base value e. For example, if the number of objects and the value of e are set to 8 and 2, respectively, a data item 0 has four forward pointers such as $1(2^0)$, $2(2^1)$, $4(2^2)$, and $8(2^3)$.



(a) (b)

Fig. 13 Tuning time results following an increase in the threshold value in $\mathcal{D}1$. (a) NN queries. (b) Range queries.



(a) (b)

Fig. 14 Tuning time results following an increase in the threshold value in $\mathcal{D}3$. (a) NN queries. (b) Range queries.

direct impact on the results of tuning time and access time. If the threshold value is met while the client is tuning to the broadcast channels, then the index does not require further selective tuning. The client awakes and constantly tunes to the channels until the desired data comes. In other words, the amount of search conducted for the index decreases as the threshold value increases, whereas the amount of data that the client must awake and listen to may increase. When the threshold value sets 1 for NN query processing, the client continues to read the pointers and attempt to perform selective tuning until 1 object is found in the closest proximity. Thus, adjusting the appropriate threshold value depending on the size or the number of data points or records is significant. As the figure shows, the index search cost decreases as the threshold value increases. The tuning time decreases to some extent.

The second experiment (Figures 14(a) and 14(b)) shows the tuning time results following an increase in the threshold value when processing the NN queries and the range queries using RCS in $\mathcal{D}3$. Similar to the above experiment, the tuning time decreases to some extent as the threshold value increases.

The next experiment evaluates for each query the performance results according to the changes in data size. First, the access time for the TDS, RCS, DSS, and (1,m) index during NN query processing in $\mathcal{D}1$ is measured (see Figure 15(a)). Subsequently, we performed a comparative evaluation of tuning time for RCS and DSS. Because RCS has better tuning time than TDS in the tuning time experiment, TDS is excluded from the evaluation. The data size is increased from 64 bytes to 1024 bytes. In Figure 15(a) (an horizontal dash line represents HGI TDS, while an horizontal solid line represents HGI RCS in the figure), TDS is found to exhibit the optimal access time. This is because query processing is performed with only the location of the data and without the index, thus reducing the data search time and broadcast cycle to guarantee fast query processing. Although the difference in performance is not shown significantly, TDS increasingly outperforms the others as the size of

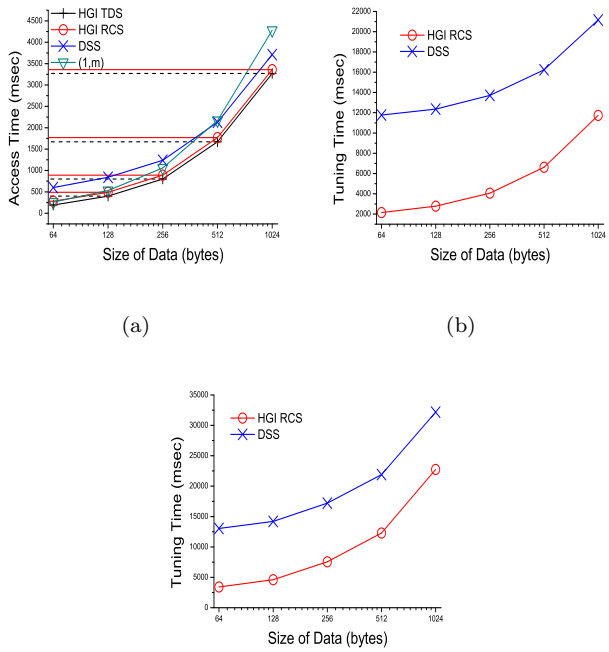


Fig. 15 Results according to the changes in data size in $\mathcal{D}1$. (a) Access time for NN queries. (b) Tuning time for NN queries. (c) Tuning time for range queries.

pointers increases because TDS allows query processing without an exponential pointer. We study this aspect in more details in the following experiment. In Figures 15(b) and 15(c), RCS exhibits better tuning time than DSS in both the NN and range queries. The reason is that, even without the additional data, such as the exponential pointer in DSS, RCS supports selective tuning through the sequential and hierarchical structure of the index. That is, RCS supports the object search within a small data size by reducing the number of the unnecessary index pointer data. In addition, RCS is not dominated by specific index structures (e.g., R-tree and Hilbert curve) and expresses the location and distribution of the spatial objects with numbers that have hierarchical meaning.

In Figure 16(a), the access time for TDS, RCS, DSS, and (1,m) index during NN query processing in $\mathcal{D}3$ is measured (an horizontal dash line represents HGI TDS, while an horizontal solid line represents HGI RCS in the figure). For the same reasons as above, TDS exhibits the best access time in $\mathcal{D}3$, as spatial query processing is possible without depending on the index. Additionally, in the case of tuning time, RCS exhibits the optimal tuning time for the same reason as in the experiments in Figures 16(b) and 16(c).

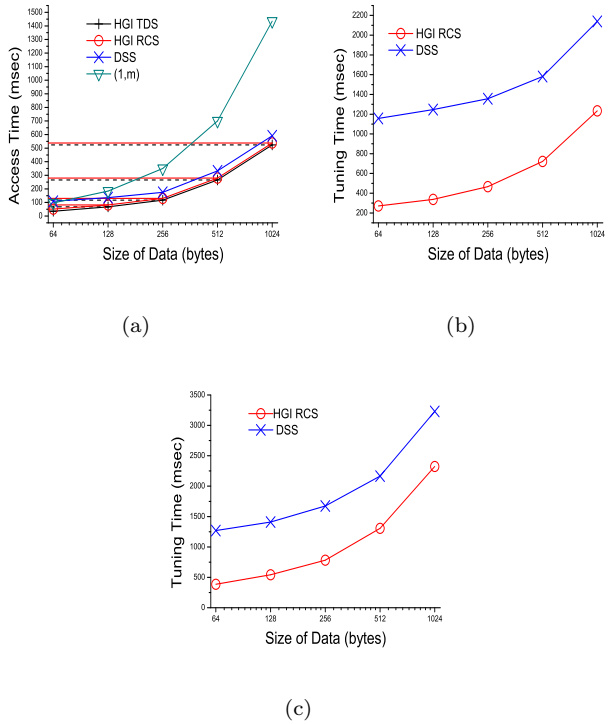


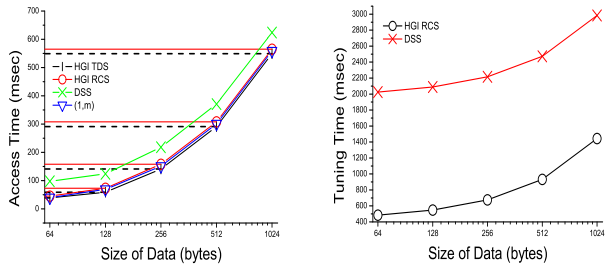
Fig. 16 Results according to the changes in data size in $\mathcal{D}3$. (a) Access time for NN queries. (b) Tuning time for NN queries. (c) Tuning time for range queries.

In Figure 17(a) (an horizontal dash line represents HGI TDS, while an horizontal solid line represents HGI RCS in the figure), the access time for TDS, RCS, DSS, and (1,m) index during NN query processing in $\mathcal{D}2$ is measured. For the same reasons as above, TDS exhibits the best access time in $\mathcal{D}2$, while RCS exhibits the optimal tuning time as in the experiments in Figures 17(b) and 17(c).

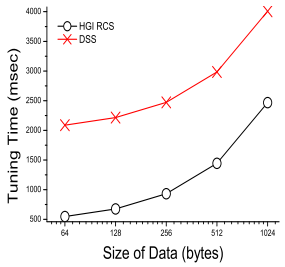
Figures 18(a), 18(b), and 18(c) show the results of the access time and tuning time following an increase in the amount of data during the NN queries and the range queries in $\mathcal{D}3$.

In Figure 18(a) the index size increases as the amount of data increases, thus increasing the index search time. Therefore, TDS, which does not depend on the index structure, exhibits the optimal performance. The data size in the experiment was set at 128 bytes. However, the performance of TDS becomes far superior as the difference between the pointer and data size get smaller.

Figures 18(b) and 18(c) show the experimental results following an increase in the amount of data. RCS, which has a simpler and smaller index structure and size, exhibits an enhanced performance compared to DSS.



(a) (b)



(c)

Fig. 17 Results according to the changes in data size in $\mathcal{D}2$. (a) Access time for NN queries. (b) Tuning time for NN queries. (c) Tuning time for range queries.

Figures 19(a) and 19(b) show the outcome of the experiments on $\mathcal{D}1$ and $\mathcal{D}3$ for the tuning time results, following the increase in the region of the range query during range query processing. RCS exhibits excellent performance in both data sets, $\mathcal{D}1$ and $\mathcal{D}3$, through its efficient index structure and search process. The figures show the experimental results regarding the outcome of query processing following an increase in the pointer size.

The access time for the proposed technique is not affected significantly by an increase in pointer size, which depends on the size and the number of the data. In contrast, DSS is significantly influenced by an increase in pointer size because it uses as many pointers as the size of the log for all of the data. Furthermore, because the tuning time selectively tunes to the data, the size of the pointer has a significant impact on the performance results. In particular, when the data size is relatively small,TDS and RCS exhibit superior query processing performance when compared to DSS.

Figure 20 shows the access time following an increase in pointer size for NN query processing in $\mathcal{D}1$, where TDS has the optimal access time.

Figures 21(a) and 21(b) show the tuning time for the NN and range queries following an increase in pointer size in $\mathcal{D}1$. RCS exhibits excellent tuning time

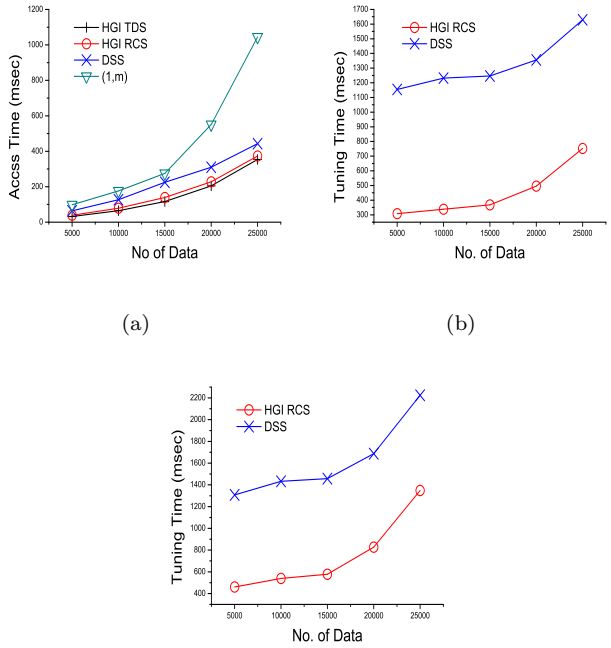


Fig. 18 Access Time and Tuning Time Following an Increase in the Amount of Data in $\mathcal{D}3$. (a) Access time for NN queries. (b) Tuning time for NN queries. (c) Tuning time for Range queries.

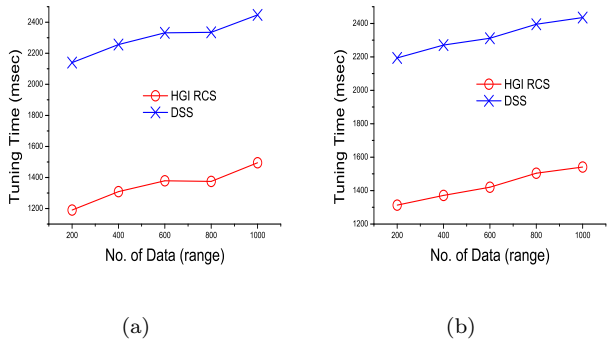


Fig. 19 Result of the Experiments on $\mathcal{D}1$ and $\mathcal{D}3$. (a) Tuning Time in $\mathcal{D}1$. (b) Tuning Time in $\mathcal{D}3$.

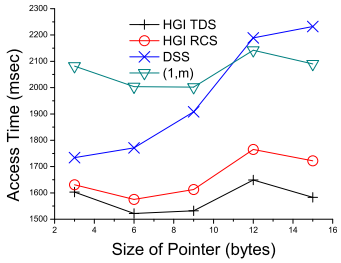
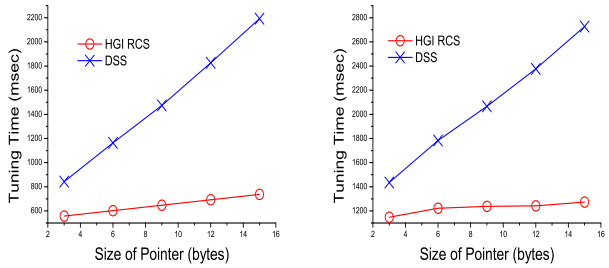


Fig. 20 Access Time Following an Increase in Pointer Size for NN Query Processing in $\mathcal{D}1$.



(a) (b)

Fig. 21 Tuning Time Following an Increase in Pointer Size in $\mathcal{D}1$. (a) Tuning Time for NN queries. (b) Tuning Time for Range Queries.

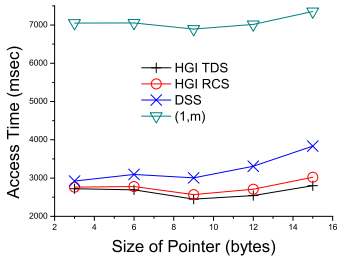
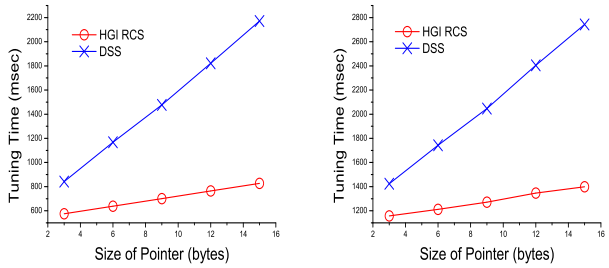


Fig. 22 Access Time Following an Increase in Pointer Size for NN Query Processing in $\mathcal{D}3$.

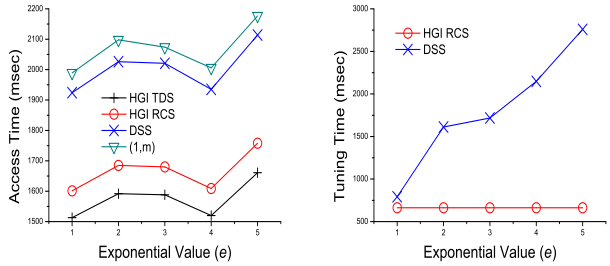
because the pointer size that was allocated to show the location of the spatial objects has increased.

Figure 22 shows the access time following an increase in pointer size for NN query processing in $\mathcal{D}3$. For the same reason as in the $\mathcal{D}1$ experiment, TDS exhibits the optimal performance.

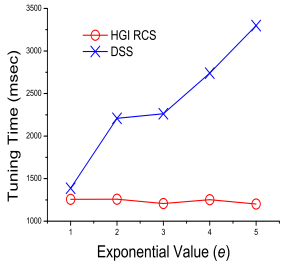


(a) (b)

Fig. 23 Tuning time for NN and range queries following an increase in pointer size in $\mathcal{D}3$. (a) NN queries. (b) Range queries.

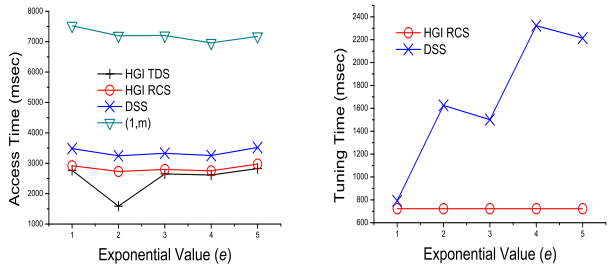


(a) (b)

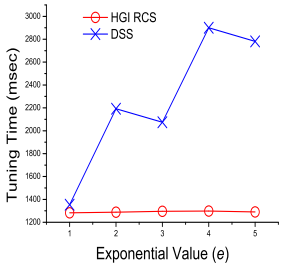


(c)

Fig. 24 Results of an experiment for increasing the exponent value of DSS in $\mathcal{D}1$. (a) Access time for NN query processing. (b) Tuning time for NN queries. (c) Tuning time for Range queries.



(a) (b)



(c)

Fig. 25 Results of an experiment for performance evaluation following an exponential value increase of DSS in $\mathcal{D}3$. (a) Access time for NN query processing. (b) Tuning time for NN queries. (c) Tuning time for Range queries.

Figures 23(a) and 23(b) demonstrate the tuning time for NN and range queries following an increase in pointer size in $\mathcal{D}3$. RCS exhibits superior performance in this case, for the same reason as in the $\mathcal{D}1$ experiment.

Next are the results of an experiment when increasing the exponent value in DSS.

Figure 24(a) shows the access time for the NN query processing results following an exponential value increase of DSS in $\mathcal{D}1$. TDS exhibits the optimal access time. In contrast, RCS demonstrates superior tuning time in both the NN queries and range queries (see Figures 24(b) and 24(c)). Through these experimental results, following an increase in exponent size, DSS is more impacted by the tuning time than by the access time. The reason is that as the exponent increases in DSS, the number of the forward pointers for each pointer decreases, whereas the number of times that the client has to perform repeated waking and sleeping increases the tuning time.

Figure 25(a) shows the access time for NN query processing following an exponential value increase of DSS in $\mathcal{D}3$. Figures 25(b) and 25(c) show the tuning time for NN and range query processing following an exponential value

increase of DSS in $\mathcal{D}3$. For the same reason as in the $\mathcal{D}1$ experiment, TDS exhibits the optimal access time. RCS also demonstrates superior performance of the tuning time for the NN queries and range queries, for the same reason as in the $\mathcal{D}1$ experiment.

In conclusion, RCS and TDS exhibit superior access time and tuning time compared to the existing techniques in various experimental environments. TDS can minimize the access time that is required to obtain the data after the request because it performs spatial query processing by the order of transfer of the objects without the index data. Meanwhile, RCS can support energy-efficient query processing because it uses the non-redundant light-weight index structure that expresses the object location as numbers, which reduces the index search time as well as the time that the client is awake.

8 Conclusion

In this paper, we proposed the Hierarchical Grid Index (HGI), a light-weight sequential location-based index structure for broadcasting environments. HGI analyzes the location and the distribution of the objects and then classifies the spatial objects into a form of hierarchical grids. In HGI, the ID number and the numbers of the objects contained in each grid are stored to support the clients' selective data tuning. Furthermore, object identification is made possible with minimum costs by granting a hierarchical identification number to each grid region (because the number of digits in the identification increases, the object is known to be located in the smaller grid) and expressing into grids only the regions where the objects are present.

HGI has two main advantages of probe wait compared to conventional spatial index. First, HGI provides a partial search of the index for spatial query processing via wireless broadcast channels. Second, with HGI, since the data objects broadcast by the server are sequentially ordered based on their locations, even if the client starts to tune an index in the middle of a broadcast cycle, it is not necessary for the client to wait for the beginning of next index segment. Therefore, HGI provides fast spatial query processing without the probe waiting regardless of when the client awakes.

We proposed two types of search algorithms to support spatial query processing. First, TDS provides the optimum access time. In this case, the server uses HGI to sequentially transfer the object identification number by the grid number order according to the location of the objects. The client reads the object identification data that are transferred from the server to identify the desired spatial objects. Because only the objects' location data are transferred by the transfer order, the client can obtain the best access time. Second, RCS offers the optimum tuning time. In this case, the server uses HGI to transfer the objects' identification number, the total number of objects contained in the grids, and the arrival time. Through the data transferred from the server, the client selectively wakes up only at the time when the necessary data arrive

to obtain the desired data. The client can use light-weight index data to obtain the desired information at a low search cost.

Finally, through performance analyses and experiments, we showed that HGI and its algorithms support accurate, rapid, and energy-efficient spatial query processing.

References

1. [1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communications Environments", *In Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD 95)*, pages 199-210, May 1995.
2. [2] A. Datta, A. Celik, J.K. Kim, D. VanderMeer, and V. Kumar, "Adaptive Broadcast Protocols to Support Power Conservation Retrieval by Mobile Users", *In Proceedings of IEEE International Conference Data Engineering (ICDE97)*, pages 124-133, April 1997.
3. [3] A. Datta, D. E. VanderMeer, A. Celik, and V. Kumar, "Broadcast protocols to support efficient retrieval from databases by mobile users", *ACM Transactions on Database Systems (TODS)*, 24(1), pages 1-79, March 1999.
4. [4] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Energy efficiency indexing on air", *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 25-36, May 1994.
   T
5. [5] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Data on Air - Organization and Access", *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 9(3), pages 353-372, May-June 1997.
6. [6] J. Shanmugasundaram, A. Nithrakashyap, R. M. Sivasankaran, and K. Ramamritham, "Efficient Concurrency Control for Broadcast Environments", *In Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD99)*, pages 85-96, June 1999.
7. [7] B. Zheng and D. L. Lee, "Information dissemination via wireless broadcast", *Communication of the ACM*, 48(5), pages 105-110, 2005.
8. [8] M. Gruteser and D. Grunwald, "Anonymous Usage of Location-Based Services through Spatial and Temporal Cloaking", *In Proceedings of the 1st ACM/USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys03)*, pages 31-42,June 2003.
9. [9] K. C. K. Lee, W.-C. Lee, J. Winter, B. Zheng and J. Xu, "CS cache engine: data access accelerator for location-based service in mobile environments", *In Proc. of Special Interest Group on Management of Data (SIGMOD)*, pages 787-789, 2006.
10. [10] W.-. Ku, R. Zimmermann, H. Wang, "Location-Based Spatial Query Processing in Wireless Broadcast Environments", *IEEE Trans. Mob. Comput.*, 7(6), pages 778-791, 2008.
11. [11] K. Mouratidis, S. Bakiras, D. Papadias, "Continuous Monitoring of Spatial Queries in Wireless Broadcast Environments", *IEEE Trans. Mob. Comput.*, 8(10), pages 1297-1311, 2009.
12. [12] K. Park, H. Choo, "Energy-Efficient Data Dissemination Schemes for Nearest Neighbor Query Processing", *IEEE Trans. Computers 56(6)*, pages 754-768, 2007.
13. [13] J. Xu, B. Zheng, W.-C. Lee, and D. L. Lee, "Energy efficient index for querying location-dependent data in mobile broadcast environments", *In Proc. of ICDE*, pages 239-250, 2003.
14. [14] 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), pages 748-761, 2007.
15. [15] 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 J.*, 15(1), pages 21-39, 2006.

16. [16] Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search", *In Proc. of VLDB*, pages 287-298, 2002.
17. [17] K. Park, H. Choo, P. Valduriez, "A scalable energy-efficient continuous nearest neighbor search in wireless broadcast systems", *Wireless Networks*, 16(4), pages 1011-1031, 2010.
18. [18] Q. Hu, W.-C. Lee, and D.L. Lee, "Power Conservative Multi-Attribute Queries on Data Broadcast", *In Proc. Int'l Conf. Data Eng. (ICDE '00)*, pages 157-166, 2000.
19. [19] S.E. Hambrusch, C.-M. Liu,W. Aref, and S. Prabhakar, "Query processing in broadcasted spatial index trees", *In Proc. Of Symposium on Spatial and Temporal Databases (SSTD)*, pages 502-521, 2001.
20. [20] J. Xu, W.-C. Lee, and X. Tang, "Exponential Index: A Parameterized Distributed Indexing Scheme for Data on Air", Proc. *In Proc. of MobiSys*, pages 153-164,2004.
21. [21] P. Xuan, S. Sen, O. Gonzalez, J. Fernandez, and K. Ramamritham, "Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments", *In Proc. of IEEE Real Time Technology and Applications Symposium*, pages 38-48, 1997.
22. [22] C.-M. Liu and K.-F. Lin, "Disseminating dependent data in wireless broadcast environments", *Distributed and Parallel Databases*, 22(1), pages 1-25, 2007.
23. [23] P.Nicopolitidis, G.I.Papadimitriou and A.S.Pomportsis, "Exploiting Locality of Demand to Improve the Performance of Wireless Data Broadcasting", *IEEE Transactions on Vehicular Technology*, vol.55, no.4, pages 1347-1361, July 2006.
24. [24] N. Roussopoulos and F. V. S. Kelley, "Nearest neighbor queries", *In Proc. of SIGMOD*, pages 71-79, 1995.
25. [25] A. Guttman, "R-trees: A dynamic index structure for spatial searching", *In Proc. of Special Interest Group on Management of Data (SIGMOD)*, pages 47-57, 1984.
26. [26] H. V. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang and A. Zhou, "VBI-Tree: A Peer-to-Peer Framework for Supporting Multi-Dimensional Indexing Schemes", *In Proc. of International Conference on Data Engineering (ICDE)*, pages 34-34, 2006.
27. [27] W.-S. Ku, R. Zimmermann, C.-N. Wan, and H. Wang, "MAPLE: A Mobile Scalable P2P Nearest Neighbor Query System for Location-based Services", *In Proc. of International Conference on Data Engineering (ICDE)*, page 160 (demo), 2006.
28. [28] W.-C Lee and B. Zheng, "DSI: A fully distributed spatial index for location-based wireless broadcast services", *In Proc. of ICDCS*,Columbus, USA, June 6-10, pages 349-358, 2005.
29. [29] B. Gedik, A. Singh, and L. Liu, "Energy Efficient Exact kNN Search in Wireless Broadcast Environments", *In Proc. Ann. ACM Int'l Workshop Geographic Information Systems (GIS '04)*, pages 137-146, 2004.
30. [30] K. Park, P. Valduriez, and H. Choo, "Mobile continuous nearest neighbor queries on air", *In Proc. of ACM SIGSPATIAL international conference on Advances in geographic information systems (ACM-GIS)*, page 65, 2008.
31. [31] T. Camp, J. Boleng and V. Davies, "A Survey of Mobility Models for Ad Hoc Network Research", *Wireless Communication and Mobile Computing (WCMC)*, 2(5), pages 483-502, 2002.
32. [32] Haojun Wang and Roger Zimmermann, "Processing of Continuous Location-Based Range Queries on Moving Objects in Road Networks", *IEEE Trans. Knowl. Data Eng*, 23(7), pages 1065-1078, 2011.
33. [33] Wei Zhang, Xingguang Yang, Wanzhen Wu, and Gang Xiang, "An Optimized Query Index Method Based on R-Tree", *Proc. of International Joint Conference on Computational Sciences and Optimization*, pages 1007-1011. 2011.
34. K. Mouratidis, S. Bakiras, and D. Papadias, "Continuous Monitoring of Spatial Queries in Wireless Broadcast Environments", *IEEE Trans. Mob. Comput.*, 8(10), pages 1297-1311, 2009.
35. P. Chowdhury, S. Sarkar, and A. Reaz, "Comparative cost study of broadband access technologies", *Proc. of Advanced Networks and Telecommunication Systems*, pages 1-3, 2008.
36. B. Zheng, W.-C. Lee, K. Lee, D. Lee, and M. Shao, "A distributed spatial index for error-prone wireless data broadcast", *VLDB J.*, 18(4), pages 959-986, 2009.