Design of Spatial Data Broadcast for Mobile Navigation Applications

Shou-Chih Lo

Department of Computer Science & Information Engineering
National Dong Hwa University
Hualien 974, Taiwan
Email: sclo@mail.ndhu.edu.tw

Abstract

Existing mobile navigators can readily display any static data related to the area surrounding a user. However, their ability to display any dynamic data is limited. In this paper, we enable the viewing of dynamic data on a navigator using wireless data broadcast. The dynamic data are periodically broadcast via base stations of wireless systems, and can be filtered by navigators fetching desired data. We address several crucial issues related to the design of this application, including data organizing, indexing, caching, and querying. We simulate the performance of the resulting system by measuring the time and energy costs associated with retrieving the dynamic data. Moreover, we demonstrate a prototyping system for the mobile navigation application.

Index Terms: Wireless Networks, Mobile Applications, Data Broadcast, Query

Processing, Cache Management,

1 INTRODUCTION

The rapid growth of portable devices and the ubiquitousness of the Global Positioning System have lead to mobile navigation systems becoming a popular means of guiding a person to a destination. A user can view the road map of his/her surrounding area on a navigator as he/she moves. Typically, the map data are downloaded to the navigator before they are retrieved, and hence existing navigators have limited ability to display the real-time status of the surrounding area. One possible application is to display the numbers of vacant spaces in nearby parking lots or the current traffic conditions on certain road segments, in which case the real-time information needs to be updated dynamically. The providing of such dynamic data has attracted some research efforts in vehicular ad hoc networks (VANETs) [3][16].

We define a *spatial object* to be an object in physical space that has data attributes associated. The data attributes can be either static (e.g., address or telephone number) or dynamic (e.g., traffic conditions or temperature). Typically, the data about static attributes (or named static data) can be predownloaded into a navigator. However, how to efficiently provide the data about dynamic attributes (or named dynamic data) becomes challenging in mobile environments. Wireless data broadcast allows users to retrieve data with a markedly lower energy requirement (due to no user transmission being required) and with a cost that is independent of the number of users. This motivates us to serve dynamic data of spatial objects using data broadcast (in particular, spatial data broadcast). The main limitation of data broadcast is that the data can only be accessed sequentially, and hence how to manage the broadcast data for efficient access is an important issue.

Location-dependent information services (LDISs) [12] have received considerable attention in recent years. LDISs typically answer location-related queries such as “find the nearest hotel to me” (a point query) or “find all restaurants within 100 meters” (a window query) on demand via a spatial database, but they can also be answered via a spatial data broadcast. Research efforts on spatial data broadcast have focused on data placement [22], index construction [4][13][21][23][25], and cache management [17][20][24] for reducing the cost (both time and energy) of answering the queries. Our approach differs from the previous work in that we consider the broadcasting of dynamic rather than static data.

In this paper we focus on the application of viewing dynamic data of spatial objects that are located within a specified sized area centered at the current location of a user. Example queries include “display all five-star hotels within 500 meters that have at least two vacant rooms” and “display all sections of roads within 200 meters on which the average vehicle speed is lower than 30 km/hr”. To obtain a global understating of the whole design, we address many of the important design issues: broadcast data organization, index construction, cache management, and query processing. We propose new grid-based index structures and new cache replacement policies. We provide a mixed-type data filtering mechanism and a cache invalidation mechanism

using bit maps. Moreover, we implement a prototyping system to realize our design.

The remainder of this paper is organized as follows. Section 2 briefly surveys the related work on data broadcast. Section 3 discusses the main design issues. Section 4 demonstrates our prototyping system of the mobile navigation application. We simulate the performance in Section 5. Finally, we draw conclusions in Section 6.

2 RELATED WORK

Data broadcast can serve users to perform different processes, such as (1) nonspatial queries, (2) spatial queries, and (3) read-only transactions. The common operation in these processes is retrieving one or more data items from the broadcast channel to answer/execute a query/transaction. In most cases, a user is equipped with a battery-driven device to retrieve broadcast data. Moreover, the data broadcast has the essential feature of sequential access for any user. Hence, the amounts of time and energy consumed in accessing broadcast data represent the main cost metrics to be evaluated.

Queries to data items that are broadcast more often or close together can be answered more quickly than other data items. This corresponds to the *data placement problem* in data broadcast. Moreover, if any index information is embedded with data broadcast, a sequential scanning of each data item can be avoided when answering a query. This corresponds to the *index construction problem* in data broadcast. The presence of local storage on the user side to cache downloaded data items raises the *cache management problem*. An appropriate cache replacement policy can increase data reusability in the cache and hence reduce the cost of downloading from the data broadcast. If dynamic data are broadcast, update consistency should be maintained in the cache.

The data placement in data broadcast has two separate goals: (1) increasing the data availability of frequently accessed data items and (2) arranging the broadcast data into a sorted order based on certain criteria. The latter goal is particularly necessary when an index structure is built into the broadcast data. Data placement techniques for nonspatial query and read-only

transaction processes have been widely reported [1][8][11][15][19]. Two common rules are used in data placement: (1) the same data item has equal spacing on the broadcast channel and (2) data items with higher access frequencies are broadcast more frequently. The optimal placement of spatial objects for answering window queries was studied in [22], with the main drawback of the reported method being the high computation cost incurred on set operations.

Index construction in data broadcast for nonspatial queries can be based on tree-based indexing [7][9][10], hashing [6], and signature [5] techniques. The former two techniques support data search using key values, while the last one supports data search using keywords. Index construction for spatial queries has been studied extensively by Zheng et al. [13][21][23][25]. A paged binary search tree (called D-tree) is used in [21] to answer point queries. In [23], the spatial objects are first arranged into a linear sequence according to the Hilbert curve, from which a B^+-tree is constructed. That work addresses continuous nearest-neighbor queries. In [13], the B^+-tree is replaced by a distributed index structure – which is based on [7] – to answer window and nearest-neighbor queries. In [4], the R* tree is used to answer window queries. The recent work of Zheng et al. [25] verified that using a grid file provides good performance over other tree-based indexes (e.g., D-tree or R-tree). We also use a grid file in this paper but propose new approaches to generate a grid file.

The update consistency can be achieved by either a multiversion data broadcast [14] or an invalidation report [2], where the latter method can provide better currency and be used to invalidate out-of-date data in the cache. We also adopt invalidation reports but use a more cost-effective representation and allow weak consistency in cache invalidation. For cache replacement, the access frequency of a cached data item is typically used to select a victim to be replaced. However, more information can be referred to in LDISs. The FAR (furthest-away replacement) policy is proposed in [17], which selects a victim from those data items that are not in the direction of movement and are furthest from the user. This method is useful when the movement is predictable. A combination method that takes the access frequency, distance, and the size of valid scope into account is proposed in [24] for specific location applications. In this

paper, we propose new replacement polices that are suitable to our application.

3 SPATIAL DATA BROADCAST

In this section we describe the main issues involved in the design of spatial data broadcast, focusing on the application of providing mobile users with dynamic data information. Consider the example environment with four cells shown in Figure 1. A navigator displays the road map and the dynamic data of spatial objects that are within the *view scope*, which is a rectangular area centered at the current location of the user. As the user moves, the navigator periodically refreshes the screen to show the new statuses within the current view scope. Basically, a base station (BS) needs to broadcast all dynamic data of spatial objects located within the serving cell. However, a view scope may cover several cells, as shown in the figure. Since a navigator can listen to the broadcast from only a single BS at any one time, a BS needs access to additional broadcast parts of dynamic data belonging to the surrounding cells.

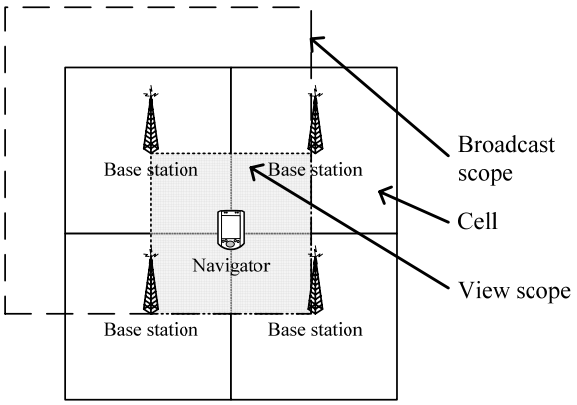


Figure 1. Schematic of the environment.

We define a *broadcast scope* to be the area containing all the spatial objects whose dynamic data have to be broadcast by a BS. We represent a cell area by the minimal bounded rectangle that encloses the cell. Without loss of generality, we use a square area to represent a certain scope of area throughout this paper. As can be seen in Figure 1, the side length of a broadcast scope is equal to the side length of a cell plus the side length of the largest view scope that a user can specified. The issue of how to collect these dynamic data is discussed in Section 4.

3.1 Data Organization

We assume that each spatial object is associated with a universal identifier called the OID (object identifier) and an object type (e.g., hotel or hospital). The OID can be used to combine dynamic and static data into a spatial object. The dynamic data of a spatial object are described by a list of <attribute, value> pairs, such as <speed, 30 km/hr> and <vacancy, 10>.

We group those dynamic data of spatial objects that are located within an area of a certain size (called the *block scope*) into a *data block*. A data block has data fields: BID, Type Signature, Size, and a list of spatial objects with dynamic data. The BID (block identifier) field contains an identifier number that is associated with the data block. The Type Signature field contains a bit vector that assists in rapidly ascertaining whether a data block contains the information about spatial objects of a particular type. This bit vector can be obtained by summarizing all the type values of spatial objects in the data block using the signature technique [5]. The Size field indicates the number of spatial objects that are attached to the block.

The data blocks are generated by dividing the broadcast scope of a BS into block scopes. The division methods are discussed in Section 3.2. One complete transmission of all data blocks from one broadcast scope on a broadcast channel is called a *broadcast cycle*. In our navigation application, a navigator needs to download those data blocks that contain dynamic data of spatial objects that are within the current view scope. In the absence of any guidance, all the data blocks in a broadcast cycle have to be scanned to find the required data blocks. To avoid this scanning, we insert index blocks (as explained in Section 3.2) into a broadcast cycle.

An index or data block is transmitted using a frame structure on a broadcast channel, where each frame has a maximum length (in bytes) and is a basic transmission unit of broadcast data. An index or data block that cannot be transmitted using a single frame is fragmented into several frames. A frame comprises mainly Header, Payload, and FEC (forward error correction) fields. The Header field contains the following information: (1) the preamble for channel synchronization and data decoding (Preamble), (2) the sequence number of a frame within a broadcast cycle (Sequence Number), (3) the type of block (i.e., index or data) that the frame

holds (Type), (4) the current frame that holds the last fragment of a certain block (More Fragments), (5) the sequence number of the next nearest index frame (Index Number), and (6) the time period after which the next nearest index frame will appear (Duration). The Payload field holds one fragment of an index or data block, and the FEC field holds the FEC code for recovering possible bit errors in the frame.

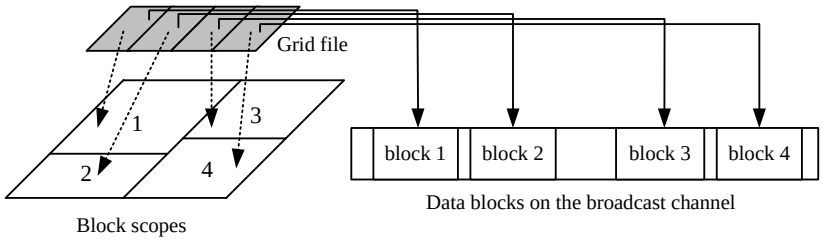


Figure 2. Index structure using a grid file.

3.2 Index Organization

We use a grid file as our index structure, since it has been verified to have good performance over other tree-based indexes in [25]. A grid file contains a series of grids, each of which guides the retrieval of a certain data block on the broadcast channel. A descriptive example is shown in Figure 2, where each grid maps to one block scope. We use an index block to carry an entire grid file. An index block has data fields: Update Version, Invalidation Bitmap, Size and a list of grids. The Update Version and Invalid Bitmap fields are used for cache invalidation, and are explained in Section 3.4. The Size field indicates the number of grids associated with an index block. Each grid has Boundary and Pointer fields. The Boundary field contains the longitude and latitude coordinates of the upper-left and lower-right corners of a block scope that is mapped by the grid. The Pointer field is denoted by the triplet <BID, Seq, and Dur>, which indicates that the data block pointed to by this grid has an identifier of BID, is located at the frame with a sequence number of Seq, and will appear after a period of Dur.

A navigator will download an entire index block before retrieving any data block on the broadcast channel. We do not further discuss the internal indexing technique for searching the grids, since an index block is smaller than all the other data blocks. To increase the availability

of an index block, we insert several identical copies of the index block into a broadcast cycle. We follow the (1, *m*) indexing scheme proposed in [7] to place *m* copies of the index block into the broadcast cycle with equal spacing.

We now discuss how to divide a broadcast scope into block scopes. We introduce five approaches based on the type of grid: fixed, semi-, dynamic, binary, and semi-dynamic grids. The first three approaches have been mentioned in [25] and the last two approaches are newly proposed in this paper. Figure 3 shows a series of examples for these approaches. In the fixed grid, the entire broadcast scope is uniformly divided into equal-sized block scopes. In a semigrid, the broadcast scope is first equally divided into stripes along one dimension (vertically in our example). Each stripe is then partitioned in another dimension into block scopes such that each block scope contains a similar number of spatial objects, where this number is no more than a predefined block size. In a dynamic grid, the entire broadcast scope is recursively divided by alternating vertical and horizontal partitions into two complementary subareas containing a similar number of spatial objects. The partitioning continues until the number of spatial objects associated with each subarea is less than a predefined block size. Each finally divided subarea is a block scope.

A binary grid is very similar to a dynamic grid except that two equal-sized subareas are divided for each partition. The detailed process is described below:

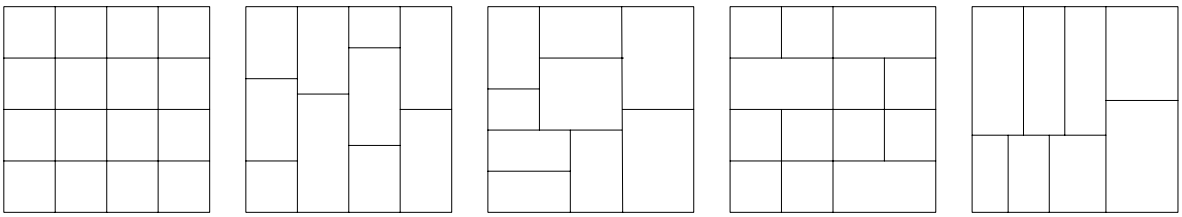
```
Binary_Grid(Area B, Block_Size S, Direction D)
//NumObj(B): Return the number of spatial objects within B;
{ If NumObj(B) ≤ S    Return;
  Else { Divide B into two equal-sized subareas B1 and B2 along the direction D;
         Change D to the other direction;
         Binary_Grid(B1, S, D); Binary_Grid(B2, S, D);
       }
}
```

In a semi-dynamic grid, we first use the dynamic grid partition to a certain depth on the

recursion and then transform the partition to a uniform one which divides a subarea into equal-sized block scopes in number of spatial objects. The detailed process is described below:

```
Semi_Dynamic_Grid(Area B, Block_Size S, Direction D, Depth H)
//NumObj(B): Return the number of spatial objects within B;
//level: static variable with initial value of zero;
{ If NumObj(B) ≤ S    Return;
  Else { If (level == H)
            Divide B into subareas Bi with each NumObj(Bi) ≈ S along the direction D;
         Else { Divide B into two subareas B1 and B2 with equal number of spatial objects
                along the direction D;
                Change D to the other direction;
                level = level +1;
                Semi_Dynamic_Grid(B1, S, D, H); Semi_Dynamic_Grid(B2, S, D, H);
         }
  }
}
```

Assume that a broadcast scope with area size $W \times W$ contains N spatial objects. The complexity of dividing k blocks is as follows: fixed grid = O(N), binary grid = O(kN), dynamic grid = semi-dynamic grid = semigrid = O(kWN); more performance comparisons are given in Section 5. The data blocks generated using any of these division methods are arranged into a broadcast cycle in the same order as they are generated.



(a) Fixed grid (b) Semigrid (c) Dynamic grid (d) Binary grid (e) Semi-dynamic grid

Figure 3. Block-division approaches.

3.3 Query Processing

A user has to specify the types of spatial objects that should be displayed, after which the navigator will refresh its screen at a fixed interval (denoted by Δr) with the dynamic data of spatial objects that are located within the view scope and are of the types requested by the user. The navigator has to collect the related data blocks from the broadcast channel for the screen updates as often as possible. This can be considered to be a continuous window query process, where the window area is equal to the view scope.

We use the notation $\vec{P} = (P_x, P_y)$ to denote a geographic location. A view scope can be represented by $< \vec{S_1}, \vec{S_2} >$, which denotes the locations of the upper-left and lower-right corners, respectively. Suppose that the current and next screen refreshes are at times T and $T+\Delta r$, respectively. Assume that a user is located at position $\vec{P}$ and has a motion vector $\vec{V}$ at time T, and hence we can predict that the user will be located at $\vec{P} + \vec{V} \times \Delta r$ at time $T+\Delta r$. The view scope centered at the predicted location is $< \vec{S_1}, \vec{S_2} > = < \vec{P} + \vec{V} \times \Delta r + \vec{W}, \vec{P} + \vec{V} \times \Delta r - \vec{W} >$, where $\vec{W} = (-\frac{1}{2} L_v, \frac{1}{2} L_v)$ with L_v being the side length of a view scope.

During the time interval Δr, a navigator has to prepare the data of those spatial objects that are within this predicted view scope. Actually, the precise location of the user at time $T+\Delta r$ may be different from the predicted one, and hence there may be some discrepancies between the correct data and what are displayed on the navigator. However, these discrepancies would be acceptable if Δr is sufficiently short.

To increase the data reusability and reduce the downloading cost, we assume that a navigator has a cache for storing previously downloaded data blocks. Those spatial objects in the predicted view scope that can be retrieved from the cache will be displayed without requiring broadcast data to be downloaded. The following algorithm lists the main steps used to process a window query:

Algorithm: Window Query Processing

Input: A window area $< \vec{S_1}, \vec{S_2} >$ and the types of spatial objects the user would like to be displayed on the navigator.

Output: The data blocks that are within the window area and contain the desired types of spatial objects.

Begin

1) Tune into the broadcast channel and download one frame. Check if the downloaded frame is of type index and contains the first fragment of an index block: if it does, proceed to the next step; otherwise, wait for a period indicated by the Duration field and then look for the index frame with the sequence number indicated by the Index Number field.
2) Check if the downloaded frame has more fragments: if it does, download all the fragments. Examine the index block and collect those grids whose mapping block scopes are within window area $< \overrightarrow{S_1}, \overrightarrow{S_2} >$ into set *TARGET*.
3) Remove any invalid data block in the cache based on the indicated values in the Invalid Bitmap field of the index block. Remove the grids from set *TARGET* if their recorded BIDs can be found in the cache.
4) According to the guidance provided by each grid in set *TARGET*, wait for an indicated duration and then prepare to download each desired data block. If the currently downloaded data block is confirmed as having no desired types of spatial objects, terminate the downloading of this data block.

End.

Step 1 of the above algorithm describes how to find the nearest index block when tuning into the broadcast channel at any time. In Step 2, the grids that are relevant to the window area are collected. These grids contain data about how to download from the broadcast channel the data blocks that are within the window area. We can easily collect the relevant grids by checking the Boundary field of each grid. Step 3 first removes all invalid data blocks in the cache (as

discussed in Section 3.4) and then removes the grids pointing to data blocks that can be accessed in the cache. To avoid missing any desired data block during this processing step, the broadcast data will continue to be downloaded into a temporary buffer until the beginning of the next step. Finally, the desired data blocks are downloaded (parts of them may be retrieved from the temporary buffer) from the broadcast channel based on the guidance provided by the grids in Step 4.

Whilst a data block is being downloaded, we will check whether it contains desired types of spatial objects. We first transform the desired types into a bit vector using the same technique used to compute the Type Signature field in a data block. We then combine the bit vector after transformation with the bit vector in the Type Signature field of the current data block using bitwise AND operations. A combination bit vector with a value of 0 confirms that the current data block contains no desired types of spatial objects, with a nonzero value indicating that the current data block may contain the desired types of spatial objects. In the former case, we can terminate the downloading of other fragments of this data block so as to reduce energy consumption.

3.4 Cache Management

Since we assume that a navigator has a cache for storing previously downloaded data blocks, we need mechanisms to maintain the cached data – particularly their invalidation and replacement. Here we use a data block instead of a spatial object as the basic storage unit of the cache so as to reduce the maintenance cost. Each downloaded data block is stored in the cache by adding the Expired Time field, which records the time after which the cached data block should be deleted.

Cache invalidation involves removing any invalid data blocks, and cache replacement selects a victim to be replaced by a new incoming data block. Here we define a data block in the cache as being *invalid* if one of the following two conditions is satisfied:

(1) The dynamic data value for one of the spatial objects in the cached data block differs

from the current value by more than a certain threshold.

(2) The entire cached data block has exceeded the expired time.

We consider a cached value of one spatial object to be referable if the deviation from the real value is small. This can prolong the lifetime of a cached data block. The cached data block is invalidated by an explicit notification under the first condition, and is invalidated by an implicit notification under the second condition.

The explicit notification is implemented using invalidation reports [2] that tell the users which data blocks have become invalid and should be removed from the local cache. An invalidation report is embedded in each index block and consists of two components: Update Version and Invalid Bitmap. The Update Version is a number that is increased by 1 (and made equal to 0 if the maximal value is reached) whenever the content of Invalid Bitmap is updated. The Invalid Bitmap contains a bit vector whose *i*th bit is 1 if the data block pointed to by the *i*th grid in the index block has become invalid, and is 0 if this data block is valid. The using of bit map can incur less overhead than traditional invalidation reports using text-based descriptions. Within the same broadcast cycle, all invalidation reports have the same content.

We show how to set the values of Invalid Bitmap at each new broadcast cycle in more detail. Figure 4 shows the change of one dynamic data value of spatial object o_i that belongs to data block b_i. A new broadcast cycle begins at each time t_i. Suppose that the *i*th bit of Invalid Bitmap at time t_0 is set to 1. At times t_1 and t_2, the *i*th bit of Invalid Bitmap is set to 0. At time t_3, the difference between the current value and the smallest value (occurring at time t_2) has exceeded the threshold. Therefore, the *i*th bit of Invalid Bitmap is set to 1 again.

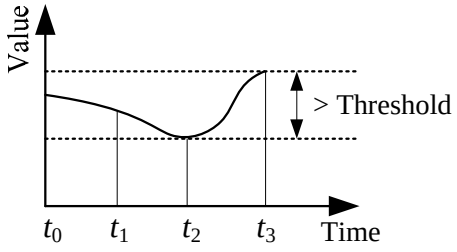


Figure 4. Setting of an invalid bit.

If the generated Invalid Bitmap differs from that at the immediately preceding broadcast cycle, the Update Version is increased by 1. All entries in the Invalid Bitmap should be set to 1 when a broadcast scope is redivided into different block scopes. A navigator has to periodically tune into the broadcast channel and download at least one index block for each broadcast cycle and refresh the cache accordingly. When the number in the Update Version field is not successively changed when a navigator is powered on or is reconnected to the network after a short period of disconnection, all the cached data blocks have to be deleted.

The implicit notification is implemented by a time-out mechanism, whereby a cached data block is automatically deleted when the time recorded in the Expired Time field has passed. The expired time is calculated by adding the maximum time interval that a data block can be kept in the cache to the time when the data block was downloaded. The cache is scanned and all the expired data blocks are deleted whenever a window query is processed.

Due to the small size of the cache in a typical navigator, we need a replacement policy to discard appropriate victims from the cached data blocks so as to accommodate new incoming data blocks. The presence of additional spatial relationships between our cached data blocks may make traditional replacement policies nonoptimal for our application. Based on the observation, we propose five possible types of replacement policies, and we compare them in Section 5.

(1) **Time-based policy**: We replace the cached data block that has not been used for the longest period of time. This is actually the standard LRU (least recently used) policy.

(2) **Frequency-based policy**: We replace the cached data block that has been referenced the least number of times. This is actually the standard LFU (least frequently used) policy.

(3) **Distance-based policy**: We replace the cached data block whose corresponding block scope is furthest from the user, since this data block will not be referenced immediately. The distance is measured from the current position of the user to the center position of the corresponding block scope.

(4) **Area-based policy**: We replace the cached data block whose corresponding block scope has the smallest area, since a user will remain longer in a block scope that is larger.

(5) **Distance-angle-based policy**: We extend the distance-based policy by involving the direction in which the user is moving. We measure the angle between the distance vector of a cached data block and the motion vector of the movement of the user, where a large angle means that the user is moving away from the corresponding block scope. We replace the cached data block that has the largest value of $distance^{\alpha} \times angle^{1-\alpha}$, where α is a parameter between 0 and 1.

4 PROTOTYPING SYSTEM

We have implemented a prototyping system for the mobile navigation application. The proposed system architecture is shown in Figure 5. The data server is responsible for data storing and accessing on the associated database. The data gateway can accept external requests and then translate them into query commands which will be submitted to the data server. The external requests may be issued from data proxies or external dynamic data collectors. The data proxy which is co-located with a BS directly interacts with mobile users. A data proxy both acts as a forwarding node which forwards data to the servers/clients and a broadcast server which periodically advertises user related messages.

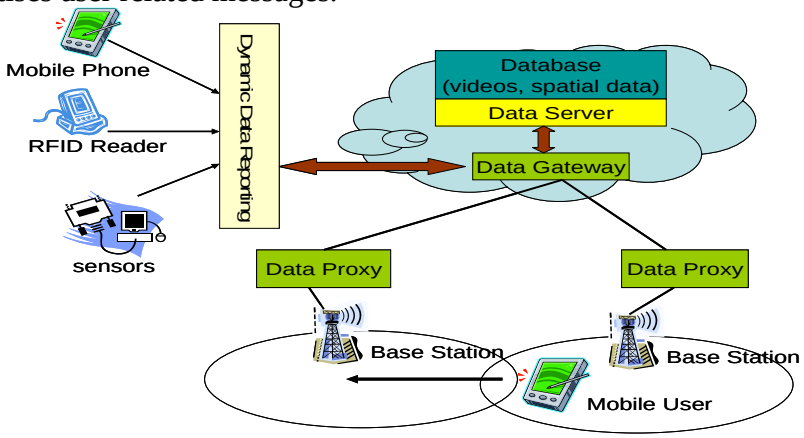


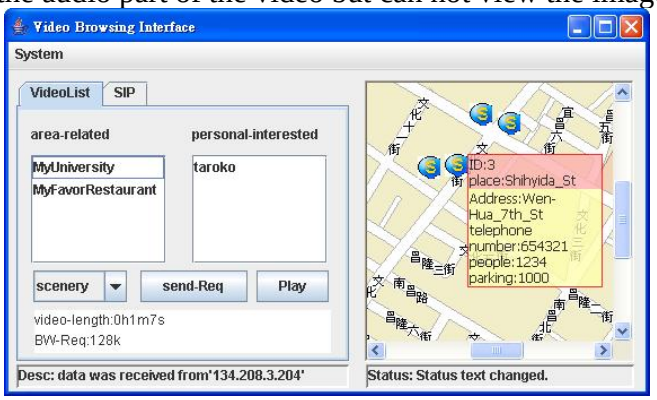
Figure 5. System architecture.

The functionality of the system can be divided into three parts: data collecting, data processing, and data rendering. In data collecting, any spatial object with its associated data attribute values or any shared data published by a mobile user is reported to our system via

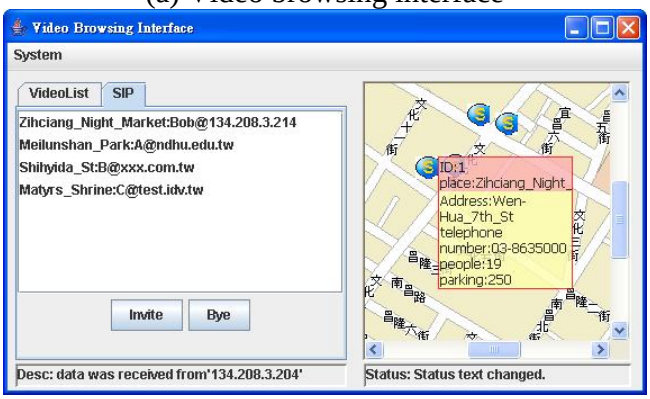
wired/wireless communication, or cellular SMS/MMS services delivery. The dynamic data of a spatial object can be captured automatically by devices such as RFID readers and sensors. In data processing, the three components: data server, data gateway, and data proxy deal with data request and dissemination. In data rendering, we provide video and map browsing services.

Data Rendering

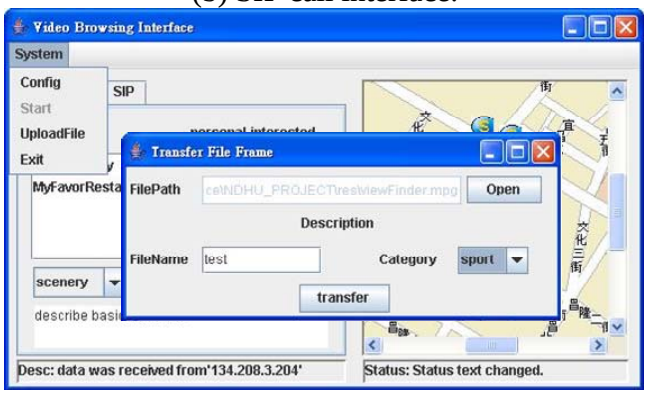
Figure 6 shows a series of user interfaces of our mobile navigation application that is currently developed in JAVA programming. The right hand side window in Figure 6(a) displays the road map and spatial objects with their data attribute values. This part is written by the UrMap [18] APIs. The VideoList page provides an interface for users to view video clips. We separate two kinds of videos: area-related and personal-interested ones. The area-related videos are tour guide videos that introduce the scenic spots in a specified area, and are categorized as commonly interested data in the system. The personal-interested videos are those ones uploaded by users and are categorized as personally interested data. The SIP (Session Initiation Protocol) page in Figure 6(b) lists all available SIP URLs (or addresses) associated with the spatial objects in a specified area. A user can click one SIP URL to establish a SIP call for any contact. This function is implemented using the JAIN SIP library (JAVA API for SIP signaling). The video browsing is served by the video streaming technique using the RTP (Real-time Transport Protocol). More importantly, we incorporate quality-of-service management techniques on video streaming, which enables resource-aware services. When the network bandwidth is not sufficient, a user can only listen to the audio part of the video but can not view the image part.



(a) Video browsing interface



(b) SIP call interface.



(c)Video upload interface

Figure 6. User interfaces.

Data Collecting

Any user-provided data are uploaded to our system via an ftp like interface with an additional text annotation tool as shown in Figure 6(c). Moreover, some of the data can be uploaded by mobile phones. For example, we provide data feedback services by which a user can score any place that has ever been visited or served by sending a short message to our system. Also, a user can upload pictures or videos taken from one scenic spot by sending multimedia short messages to our system. These functions are implemented using the J2ME programming and over the 3G cellular network.

In the following, we introduce how different system-provided data, particularly for spatial objects, are collected. We assume that the static data of spatial objects are already there and can

be accessed locally on the user side. We provide different kinds of methods to collect dynamic data.

Web pages: We write programs to periodically retrieve some web pages of on-line reservation systems and parse dynamic data associated.

RFID: We involve a RFID system to manage parking lots such that the current number of vacant lots can be reported to our system. The RFID system which operates at the 13.56 MHz band contains a reader and several tags that can be written with the status of entering or leaving.

Sensor: We deploy sensors to monitor the temperature and humidity of one area. These sensors support the Zigbee (IEEE 802.15.4) protocol and can report data to a gateway that in turn forwards them to our system.

Data Processing

In our implemented system, three database tables with their associated attributes are created as follows:

ObjectTable: O_ID | Name | Region | Type | Longitude | Latitude | Address | Telephone | SIP URL| Dynamic_Data

VideoTable: V_ID | File Name | Region | Type | Server_IP | Port | Requirement

ObjectTable stores static and dynamic data of all spatial objects. The Region attribute records the region (administrative or network domain) where the spatial object is belonging to. In this primitive version of the system, we allow each spatial object having at most one dynamic attribute. We assume that a copy of ObjectTable with the Dynamic_Data attribute having null values is pre-downloaded into the user device. VideoTable stores a list of video files with their descriptions. The combination of Server_IP and Port can uniquely identify a video stream. The Requirement attribute specifies the minimal bandwidth required to play the video.

The data processing flow is along the path “mobile user – data proxy – data gateway – data server”. A data storing request happens for the following cases: updating the dynamic data of a spatial object and uploading a user-provided video file. A data accessing request happens for the

following cases: playing a video file, querying interested videos for a mobile user, and retrieving area-related data for a data proxy.

To facilitate user access and reduce the usage of wireless bandwidth, we make use of data broadcasting to disseminate commonly interested data. In the system, the following area-related data will be broadcast periodically by each data proxy: video list (V_ID | File Name | Server_IP | Port | Requirement) and spatial objects with dynamic data (O_ID | Dynamic_Data). Each broadcast data item is represented by a list of attribute-value pairs. Figure 7 shows how a data proxy gets area-related data from the data server. To keep the broadcast data up-to-date, a data proxy has to retrieve these data periodically.

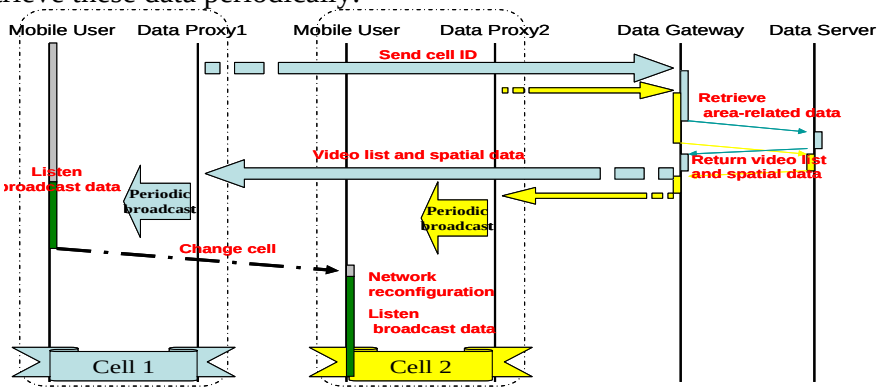


Figure 7. The retrieval and broadcast of area-related data.

For video browsing, we separate the audio and image streams from a video file. The audio stream will be played always but the image stream will be played dependently. At the data proxy, we check whether the remaining available bandwidth of the associated BS satisfies the minimal resource requirement of playing the video. If it is not, the data proxy will notify the data gateway to filter out any packet on the port used by the image stream. This port will be re-opened when having sufficient bandwidth. The playing of streams and packet filtering are implemented by the software tools vlc and iptables, respectively.

5 Performance Evaluation

We evaluated the performance of our system using simulation. We considered the worst-case

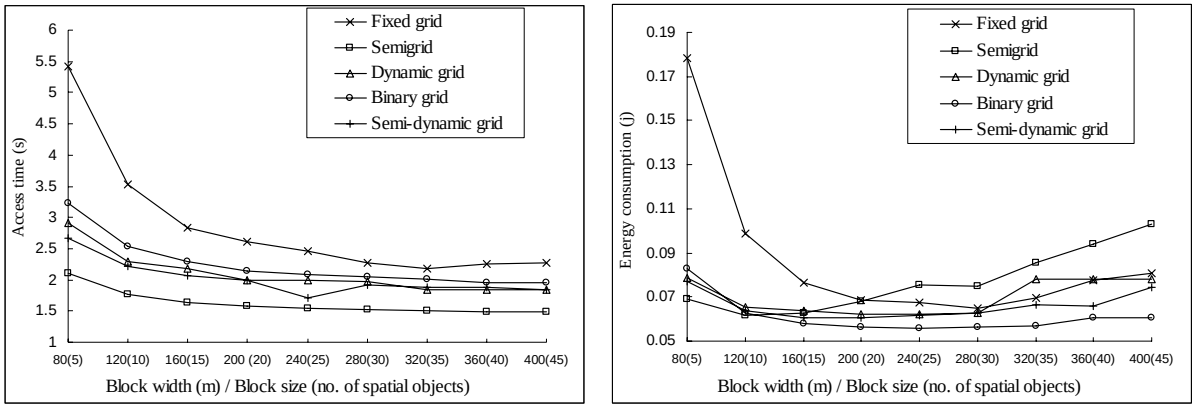
scenario by assuming that a user expects to view all types of spatial objects. Also, we assumed that each spatial object has only one dynamic attribute. The metrics used in our performance evaluation were the time and energy consumed in answering a window query (the size of a window area is equal to that of a view scope), which are relevant to the access and tuning times defined in [7]. The access time is defined as the time that elapses from the moment a window query begins being processed to the moment when all data blocks in the window area have been downloaded, and the tuning time is the time spent by a navigator downloading data from the broadcast channel during the processing of a window query. Assuming that a navigator consumes W_a watts in active mode (i.e., when downloading data) and W_s watts in silent mode, the total energy consumption in answering a window query is $W_a \times$tuning time + $W_s \times$(access time−tuning time).

We limit the simulation to a single-cell environment with an area of 2000 m × 2000 m, an available data rate of 144 Kbps, and a view scope of 500 m × 500 m. The navigator screen refresh interval is 2 s. The energy consumption parameters are W_a = 250 mW and W_s = 50 μW. The movement of a user is modeled as follows: A user randomly picks a destination in the cell and moves towards that destination at a fixed speed of 60 km/hr. Once the user reaches the destination, he/she picks another destination randomly. We uniformly insert six index blocks into a broadcast cycle. A total of 1000 spatial objects are distributed into the cell according to the cluster model: Spatial objects are grouped into five clusters in the space that are randomly distributed throughout the cell. The distance between a spatial object and the center of its associated cluster follows an exponential distribution with a mean value of 100 m.

Experiment A: Comparisons of block-division approaches

In this experiment the navigator does not have a data cache. In a fixed grid, we divide the area into square block scopes with the specified block width. In a semigrid, we first divide the area into stripes with widths of 100 m. The depth is set to three in a semi-dynamic grid. The maximal number of spatial objects that can be contained in each data block in a non-fixed grid is determined by the block size.

Figure 8 shows that a semigrid exhibits the shortest access time and a binary grid exhibits the lowest energy consumption. The access time is directly proportional to the size of a broadcast cycle, which in turn is relevant to the number of data blocks. A semigrid requires fewer data blocks to accommodate all the spatial objects, so the access time is low. A binary grid results in more data blocks than does a dynamic grid, because the number of spatial objects in each partition is not equal. Therefore, the access time is higher for a binary grid than for a dynamic grid. A fixed grid – which is less suited to partitioning nonuniformly distributed spatial objects – exhibits the worst performance. The tuning time represents a tradeoff with the size of a data block. More energy is consumed when a large data block is downloaded, but fewer data blocks need to be downloaded when answering a window query. The average number of spatial objects in a binary grid is less than that in a semigrid, a dynamic grid, or a semi-dynamic grid, and hence the tuning time is shorter.



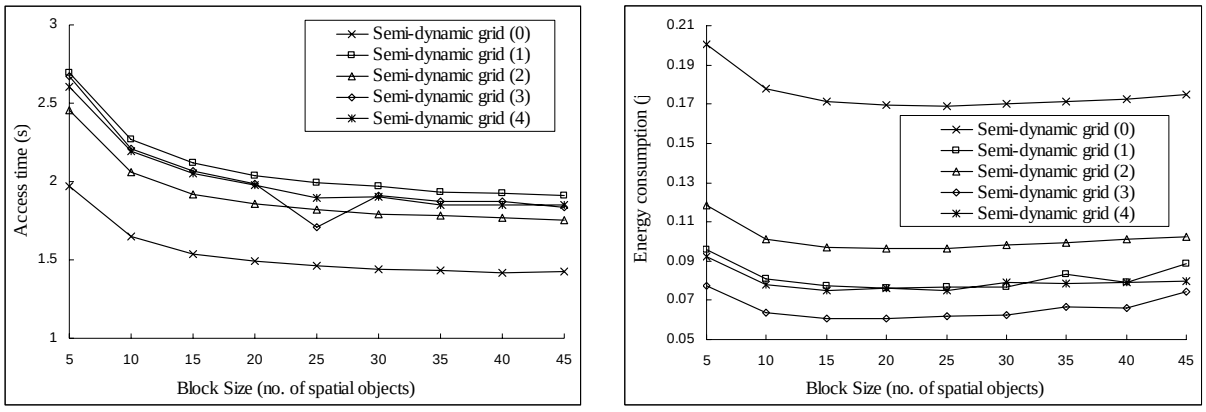
(a) (b)

Figure 8. Comparisons of block-division approaches. The *x*-axes show both the block width and the block size (within parentheses).

These experimental results appear to indicate that the semigrid exhibits the best performance, particularly in terms of the access time. However, a semigrid might perform worse when the dimension in which the stripes are partitioned does not match the main dimension of the cluster distribution. In a semigrid, the best division is found by adjusting the dimension of partitioning stripes, the width of a stripe, and the block size, and this is associated with a high computing cost.

A dynamic grid is also associated with overhead on each division of two subareas containing similar numbers of spatial objects. In contrast, a binary grid is an efficient approach for the division, and also exhibits good performance in terms of the energy consumption.

A semi-dynamic grid inherits both the features of a semigrid and a dynamic grid. With a small depth value, a semi-dynamic grid looks like a semigrid; while, with a large depth value, a semi-dynamic grid looks like a dynamic grid. We individually observe the performance of semi-dynamic grids by changing the depth after which a uniform partition is performed instead. As shown in Figure 9, a semi-dynamic grid has the lowest access time when the depth is zero and has the lowest energy consumption when the depth is three. Hence, a semi-dynamic grid has the flexibility to lower access time or energy consumption accordingly.



(a) (b)

Figure 9. Influence of the depth on semi-dynamic grids.

Experiment B: Effects of the number of index blocks

We now investigate the settings for the number of index blocks (the *m* value) within a broadcast cycle. We use the same settings in Experiment A, but focus on the performance of a binary grid with a block size of 25. Figure 10 shows the access time for different *m* values. When there is no index block, the access time is equal to the delivery time of the entire broadcast cycle. A larger *m* results in a shorter waiting time for an index block on the broadcast channel but also a longer broadcast cycle. Hence, the setting of *m* represents a tradeoff with the access time. We found that the optimal value was either 5 or 6 when the number of spatial objects was between

600 and 2000. When there is no index block, the tuning time is equal to the access time, and the energy consumption is about 0.53 joules. The tuning time becomes invariant if *m* is more than 1, for which the energy consumption is about 0.055 joules. As a result, using index blocks can greatly reduce the energy consumption (by about 89%) but produce little improvement (at most 2.1%) in the access time. This conclusion is also true for the other grid division approaches, since the factor is mainly dominated by the quantity of index blocks but not the structure of index blocks.

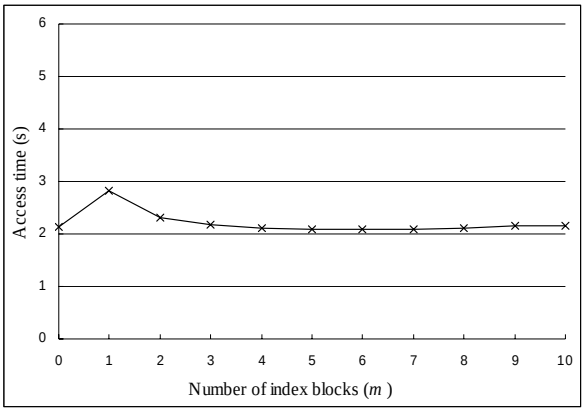


Figure 10. Access time vs. the number of index blocks

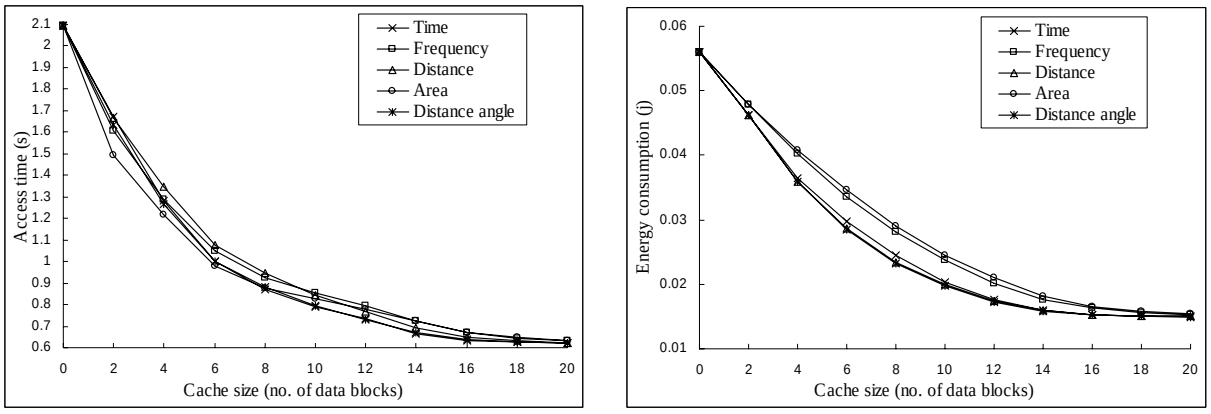
Experiment C: Comparisons of cache replacement policies

In this experiment, we evaluate the performances of cache replacement policies based on time, frequency, distance, area, and distance angle. Parameter α is set to 0.5 in the distance-angle-based policy. We fix the block size to be 25 and use the cluster model. Any newly cached data block will expire after 30 s.

Figure 11 shows the performance when a binary grid is used (note that a dynamic grid produced similar results). The average size of a data block is 543 bytes. We found that the access time and energy consumption can be reduced by 21.9% and 17.4%, respectively, if the cache size is 2 (about 1086 bytes). The best policy for the access time is the area-based one when the cache size is less than 8 (about 4344 bytes); otherwise it is the distance-angle-based one. The best policy for the energy consumption is the distance-angle-based one. The order for the variances of

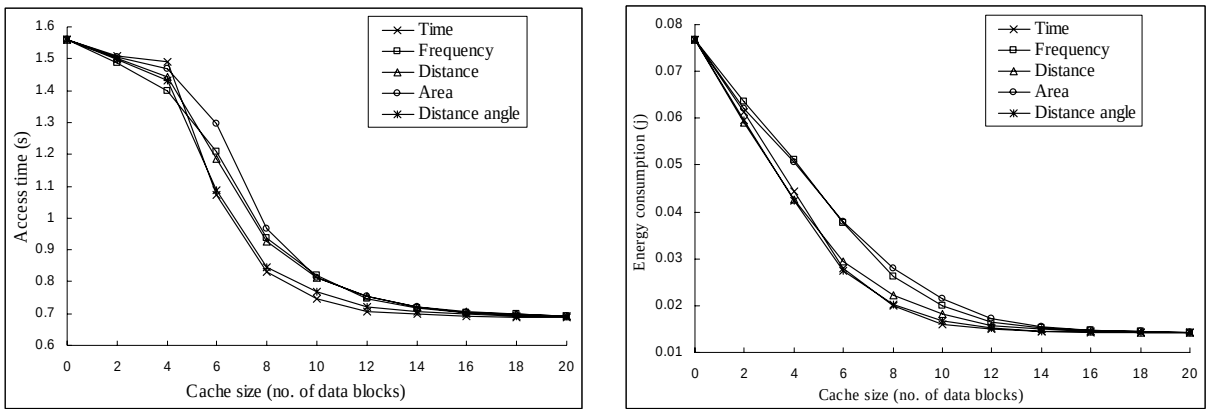
the areas of the generated data blocks is binary grid > dynamic grid >> semigrid. Hence, the area information can be used to differentiate among data blocks divided by binary and dynamic grids. The reusability of a cached data block with a large area is definitely high, but this is also associated with a high downloading cost. Consequently, the area-based policy exhibits a good access time but poor energy use. In a binary grid, the divided block scopes tend to be randomly distributed in space such that cache replacement is influenced more by the distance and angle than by the time.

Figure 12 shows the performance for a semigrid as a function of the cache size. The average size of a data block is 715 bytes in this experiment. We found that the access time and energy consumption can be reduced by 3.7% and 22.3%, respectively, for a cache size of 2 (about 1430 bytes). The best policy for the access time is the frequency-based one if the cache size is less than 4 (about 2860 bytes); otherwise it is the time-based one. The best policy for the energy consumption is the distance-angle-based one if the cache size is less than 6; otherwise it is the time-based one. When using the semigrid, the divided block scopes are stacked in each stripe. As a user moves away from the area of one stripe, all the included data blocks in that stripe will have a lower probability of being accessed again. Hence, the distance factor has less effect than the time factor on the cache replacement.



(a) (b)

Figure 11. Cache performance using a binary grid.



(a) (b)

Figure 12. Cache performance using a semigrid.

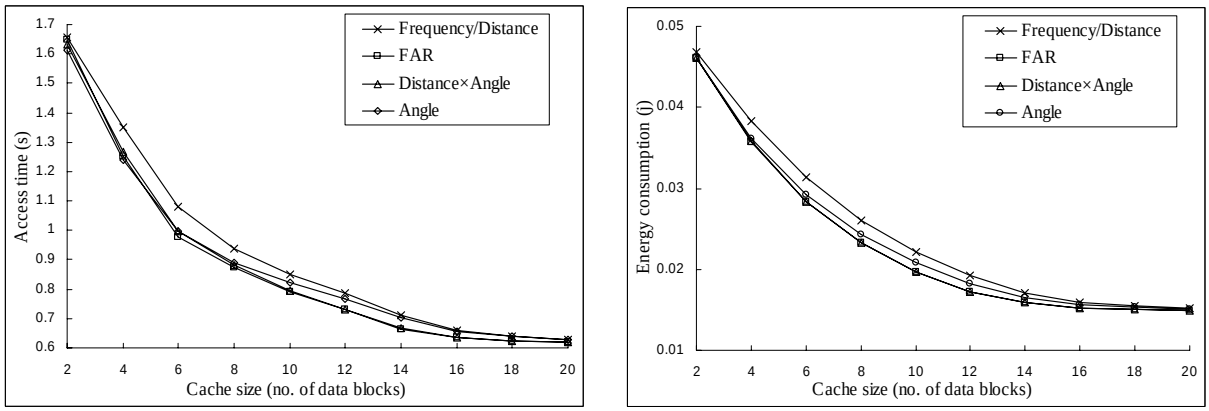


Figure 13. Comparison of cache performance with other policies.

We now compare the proposed distance-angle-based policy with the FAR policy [17] and that proposed in [24]. In the FAR policy, the evaluated value can be transformed according to

$$value = \begin{cases} 1/distance, & \text{if } angle < \pi/2 \\ -distance, & \text{otherwise} \end{cases}.$$

In [24], the evaluated value is *frequency*/*distance*. Both policies will replace the cached data block with the smallest evaluated value. In Figure 13, we label our policy as *distance*×*angle* when α is set to 0.5 and *angle* when α is set to 1. Note that we found that the evaluated value of *frequency*/*distance* provides no benefits. The performance of the policy with *distance*×*angle* is similar to that of the FAR policy, and *angle* outperforms the others when the cache size is less than 4. This is due to all of the cached data blocks being recently downloaded, and hence are

near to the current location of the user, with distance having a lesser effect on cache replacement. Our proposed policy can thus be adjusted by modifying the α value.

6 CONCLUSIONS

In this paper, we present an application that provides a dynamic view of the surrounding area on a navigator screen. Spatial data broadcast is used to disseminate the dynamic data in order to serve large volumes of mobile users with a reduced server overhead. We have addressed many of the important design issues. In the data and index organization, we first group dynamic data into data blocks and then construct a grid file as the index structure. A comparison of five block-division approaches revealed that (1) a fixed grid exhibits worse performance when the dynamic data are not distributed uniformly, (2) a semigrid exhibits a good access time but a high computing cost, (3) a dynamic grid exhibits a slightly better access time than a binary grid but also a high computing cost, (4) a binary grid exhibits a low computing cost and good energy saving, and (5) a semi-dynamic grid has the flexibility to lower access time or energy consumption accordingly. In terms of cache management, invalidation reports are disseminated with the broadcast data for the purpose of cache invalidation, and five cache replacement polices were assessed. The best policy was found to vary with the cache size and block-division approach, with the distance-angle-based policy (a parameterized approach) generally performing well.

In the future, we will consider a heterogeneous wireless networking environment and study the design of a hierarchical broadcast in which various data blocks with different data resolutions are broadcast according to the available data rates. For example, we could disseminate two versions of data broadcasts: one with full data and one with simplified data. When the user is moving quickly (resulting in a lower data rate), he/she could tune into the broadcast channel containing simplified data so as to obtain a rough view and, when the information looks useful, the user could slow down and tune into the other broadcast channel to obtain the details.

REFERENCES

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data management for Asymmetric Communication Environments," *Proc. ACM SIGMOD Conf.*, pp. 199-210, May 1995.

[2] D. Barbara and T. Imielinski, "Sleepers and Workaholics: Caching Strategies for Mobile Environments," *Proc. ACM SIGMOD Conf. Management of Data*, pp. 1-12, May 1994.

[3] M. Caliskan, D. Graupner, and M. Mauve, "Decentralized Discovery of Free Parking Places," *Proc. 3rd Intel. Workshop on Vehicular Ad Hoc Networks*, pp. 30-39, 2006.

[4] S. Hambrusch, C. M. Liu, W. Aref, and S. Prabhakar, "Query Processing in Broadcasted Spatial Index Trees," *Proc. Seventh Intel. Symposium on Spatial and Temporal Databases*, pp. 502–521, July 2001.

[5] Q. Hu, W. C. Lee, and D. L. Lee, "Power Conservative Multi-Attribute Queries on Data Broadcast," *Proc. International Conference on Data Engineering*, pp. 157-166, February 2000.

[6] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Energy Efficient Indexing on Air," *Proc. ACM SIGMOD Conf.*, pp. 25-36, May 1994.

[7] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Data on Air: Organization and Access," *IEEE Trans. on Knowledge and Data Engineering*, vol. 9, no. 3, pp. 353-372, May/June 1997.

[8] S. Jiang, N. H. Vaidya, "Satellite-Based Information Services: Response Time in Data Broadcast Systems: Mean, Variance and Tradeoff," *Mobile Networks and Applications*, vol. 7, no. 1, pp. 37-47, January 2002.

[9] S. C. Lo and A.L.P. Chen, "Optimal Index and Data Allocation in Multiple Broadcast Channels," *Proc. 16th IEEE International Conference on Data Engineering*, pp.293-302, February 2000.

[10] B. Lee and S. Jung, "An Efficient Tree-Structure Index Allocation Method over Multiple Broadcast Channels in Mobile Environments," *Database and Expert Systems Applications*, pp. 433-443, 2003.

[11] G. Lee, S. C. Lo, and A.L.P. Chen, "Data Allocation on the Wireless Broadcast Channel for Efficient Query Processing," *IEEE Trans. On Computers*, vol. 51, no. 10, pp. 1237-1252, October 2002.

[12] D. L. Lee, J. Xu, B. Zheng, and W. C. Lee, "Data Management in Location-Dependent Information Services," *IEEE Pervasive Computing*, pp. 65-72, 2002.

[13] W. C. Lee and B. Zheng, "DSI: A Fully Distributed Spatial Index for Location-Based Wireless Broadcast Services," *Proc. 25th Intel Conference on Distributed Computing Systems*, pp. 349–358, June 2005.

[14] E. Pitoura and P. K. Chrysanthis, "Multiversion Data Broadcast," *IEEE Trans. on Computers*, vol. 51, no. 10, pp. 1224-1230, October 2002.

[15] W. C. Peng, J. L. Huang, and M. S. Chen, "Dynamic Leveling: Adaptive Data Broadcasting in a Mobile Computing Environment," *ACM Mobile Networks and Applications*, vol. 8, no. 4, pp.355-364, August 2003.

[16] F. Picconi, N. Ravi, M. Gruteser, and L. Iftode, "Probabilistic Validation of Aggregated Data in Vehicular Ad-Hoc Networks," *Proc. 3rd Intel. Workshop on Vehicular Ad Hoc Networks*, pp. 76-85, 2006.

[17] Q. Ren and M. H. Dunham, "Using Semantic Caching to Manage Location Dependent Data in Mobile Computing," *Proc. ACM MOBICOM*, pp. 210-221, August 2000.

[18] UrMap: http://www.urmap.com/SearchEngine/api/.

[19] N. H. Vaidya and S. Hameed, "Scheduling Data Broadcast in Asymmetric Communication Environments," *Wireless Networks*, vol. 5, no. 3, pp. 171-182, May 1999.

[20] J. Xu, X. Tang, and D. L. Lee, "Performance Analysis of Location-Dependent Cache Invalidation Schemes for Mobile Environments," *IEEE Trans. on Knowledge and Data Engineering*, vol. 15, no. 2, pp. 474-488, March 2003.

[21] J. Xu, B. Zheng, W. C. Lee, and D. L. Lee, "Energy Efficient Index for Querying Location-Dependent Data in Mobile Broadcast Environments," *Proc. 19th IEEE International Conference on Data Engineering*, pp.239-250, March 2003.

[22] J. Zhang and L. Gruenwald, "Efficient Placement of Geographic Data Over Broadcast Channel for Spatial Range Query Under Quadratic Cost Model," *Proc. Third ACM Intel Workshop on Data Engineering for Wireless and Mobile Access*, pp. 30–37, September 2003.

[23] B. Zheng, W. C. Lee, and D. L. Lee, "Search Continuous Nearest Neighbors on the Air," *Proc. 1st IEEE International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pp.236-245, August 2004.

[24] B. Zheng, J. Xu, D. L. Lee, "Cache Invalidation and Replacement Strategies for

Location-Dependent Data in Mobile Environments," *IEEE Trans. on Computers*, vol. 51, no. 10, pp. 1141-1153, October 2002.

[25] B. Zheng, J. Xu, W. C. Lee, and D. L. Lee, "Grid-Partition Index: A Hybrid Method for Nearest-Neighbor Queries in Wireless Location-Based Services, " *VLDB Journal*, vol. 15, no. 1, pp.21-39, January 2006.