Client-Side Caching for Nearest Neighbor Queries

Kwangjin Park and Chong-Sun Hwang

Abstract: **The Voronoi diagram (VD) is the most suitable mechanism to find the nearest neighbor (*NN*) for mobile clients. In *NN* query processing, it is important to reduce the query response time, since a late query response may contain out-of-date information. In this paper, we study the issue of location dependent information services (LDISs) using a VD. To begin our study, we first introduce a broadcast-based spatial query processing methods designed to support *NN* query processing. In further sections, we introduce a generic method for location-dependent sequential prefetching and caching. The performance of this scheme is studied in different simulated environments. The core contribution of this research, resides in our analytical proof and experimental results.**

Index Terms: **Data caching, energy conservation, geographical information systems.**

I. INTRODUCTION

Over the last two decades, several studies have been conducted with emphasis on spatial database processing. Location-dependent information services (LDISs) is one of the applications which is gaining increasing attention. LDISs represent new innovating ways to satisfy customer needs, such as traffic reports, hotel information and weather broadcasts [1]. A core concept of LDISs is the *NN* query, which retrieves the data closest to a query point [2].

In broadcast environments, the server continuously sends data to the clients, without the any specific client having to send an actual request. Any number of clients can monitor and retrieve data from the broadcast. Therefore, the client is responsible for filtering specific desirable data. If the data is efficiently organized to represent the needs of the clients, such a scheme makes effective use of low wireless bandwidth and is ideal for achieving maximum scalability. Two key requirements for data access in wireless environments are power conservation and the minimization of client waiting time. In push-based systems, the mobile clients must wait until the server broadcasts the desired data. In this case, the client waiting time is determined by the overall usage of the broadcast data channel [3]. A technique used to address this issue, called *air indexing*, operates by interleaving indexing information among the broadcast data items. At the same time, the client device can reduce its battery power consumption through the use of selective tuning [4], [5]. The *air indexing* technique can be evaluated in terms of the following factors. First, *access time (AT)*: The average time elapsed from the moment a user issues a query to the client to the moment when the required data item is received by the client. Second, *tuning time (TT)*: The amount of time spent by a client listening to the channel. Then, *AT* consists of two separate components, namely: *probe wait*: The average duration for getting to the next index segment. If we assume that the distance between two consecutive index segment is L, then the *probe wait* is $L/2$. *Bcast wait*: The average duration from the moment the index segment is encountered to the moment when the required data item is downloaded. *AT* is the sum of the *probe wait* and *Bcast wait*, and these two factors work against each other [4], [5].

In general, the fastest *access time* in a broadcast cycle is obtained when there is no index, however, this increases the *TT*. The number of indices in the broadcast cycle has to be optimized by taking into consideration the trade off between the *TT* and the *AT*. Consequently, in order to achieve efficient indexing on air, it is necessary to simultaneously minimize both the *TT* and the *AT*. A mobile environment, in addition to the active usage time, energy consumption can also be influenced by sleep duration, since the client also consumes battery power while it stays in doze mode. Several studies have been conducted relating to reducing energy consumption, most of them have focused on reducing the client's *tuning time*. However, little attention has been given considering the *access time* for energy consumption. This point will be examined further.

Fig. 1 shows the example of the $(1, m)$ indexing technique. This figure shows an example of the client attempting to download data item 9 from the wireless broadcast channel. The client first tunes the data item 3 and obtains the pointer to the next occurrence of the index segment from the data item 3, since each data item has the offset to the beginning of the next index segment [5]. The client switches to doze mode until the next index segment arrives. The client tunes and accesses the index segment to find out when to tune into the broadcast channel to get the required data item. Then, switches to doze mode until the desired data item arrives. Finally, the client tunes to the channel and downloads the data item 9. The $(1, m)$ index scheme uses indices to conserve battery power. However, this method has the worst possible latency, because even if the desired data is just in front of them, the clients have to wait until the beginning of the next broadcast cycle, as shown in the Fig. 1. Therefore, we conclude that the existing indexing methods are unsuitable for LDISs.

In a recent paper [6], we proposed the concept of data sorting for broadcasting called the broadcast-based location dependent data delivery scheme (BBS). Preliminary simulation-based results showed that the BBS significantly reduces the *AT*. In this paper, we attempt to reduce both the *TT* and *AT* in the wireless mobile computing environment. After pointing out the limitations of the existing indexing schemes, we present various schemes that can overcome these problems. We believe this is the first work in which *NN* query processing without an index segment is proposed, providing the optimum *access time*. Throughout this paper, we assume that the data objects are in 2-dimensional space and are static, such as restaurants, hospitals,

Manuscript received July 30, 2005; approved for publication by Kung Yao, Guest Editor, November 30, 2005.

The authors are with the Dept. of Computer Science and Engineering, Korea University, 5-1, Anam-dong, Seongbuk-Ku, Seoul, 136-701, Korea, email: {kjpark, hwang}@disys.korea.ac.kr.

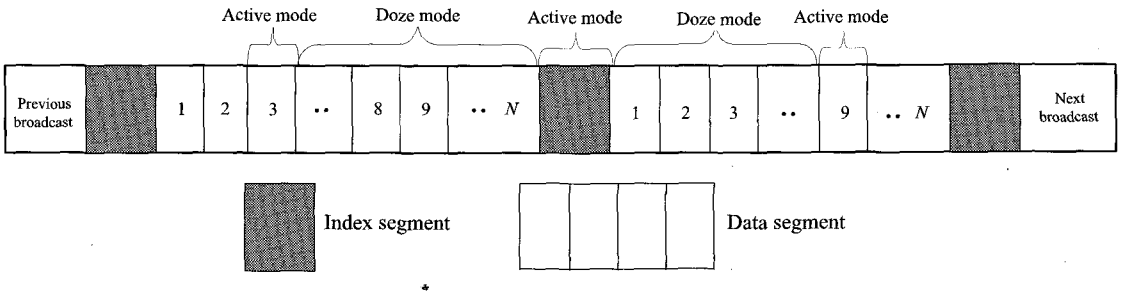


Fig. 1. Data and index organization.

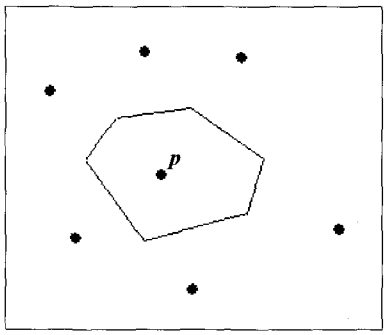


Fig. 2. All the points in the polygon have the same *NN*, namely, point p.

and hotels. The mobile clients can identify their locations using systems such as the global positioning system (GPS).

II. RELATED WORK

Spatial databases have been studied extensively during the last two decades, from this research, several spatial access methods have been proposed. In this section, we provide some background on the location model, caching models and air index schemes, which we adapt in this study.

A. Voronoi Diagrams (VDs)

VDs are fundamental tools used to express the proximity of geometric objects. The VD divides a space into disjoint polygons where the nearest neighbor of any point inside a polygon is the generator of the polygon. The VD for n objects on a plane can be constructed at $O(n \log n)$ complexity using a simple sweep algorithm. However, the maintenance cost is high, especially for high dimensional space or moving objects, thus hindering the application of the VD structure for complex and dynamic data sets [7]. Let $N := A\{n_1, n_2, \cdots, n_n\}$ be a set of n distinct objects in the map. We define the $\mathcal{V}(p)$ as the subdivision of the map into p cells, one for each site in N, with the property that a point q lies in the cell corresponding to a site pi, if and only if $dist\ (q, pi) < dist\ (q, pj)$ for each $pj \in N$ with $j \neq i$. Fig. 2 shows an example of VD for *NN* searching.

The general definition of the Voronoi cell (VC) of a point in the d-dimensional space $\Re^d$ follows.

Theorem 1: If n is a d-dimensional point, N is a set of n points in the d-dimensional space $\Re^d$, then $\mathcal{V}(p)$, the VC of the point n given set N, is defined as the unique convex polygon which contains all the points in the set $\mathcal{V}(p)$

$$\mathcal{V}(pi) = \{q \in \Re^d \mid \forall\, pi \in N, dist\ (q,\ pi) < dist\ (q,\ pj)\} \quad \text{where } j \neq i. \quad (1)$$

B. Caching Methods for LDISs

The client's cache stores frequently accessed information so that queries can be answered without connecting to a server. However, frequent disconnection and the natural mobility of clients may cause cache inconsistency. In [1], authors studied the cache coherency issues in the context of LDISs and proposed several cache invalidation schemes. For location dependent invalidation, authors propose bit vector with compression, grouped bit vector with compression, and implicit scope information methods. In this paper, authors separate location-dependent data corruption from traditional cache corruption. In [8], authors present an indexing and semantic cache method for location dependent queries based on the VDs. In this method, VDs are used to preprocess the data in order to answer location-dependent queries quickly, and a semantic cache to validate the answer. They also present three cache replacement schemes based mainly on the size of the area, and distance between two centers of the cached data.

C. Air Index Replication Scheme

In [9], authors present three energy-efficient index replication approaches, FL, NL, and SL, that are based on three criteria: Accessibility, energy efficiency, and adjacency. In the forward link (FL) approach, authors use forward address. The forward address means that it is the address in the next broadcast rather than that in the current broadcast. The nephew link (NL) approach adopts nephews as link pointers when replicating control index buckets. The nephew of a control index is the children of its next sibling. The sibling link (SL) approach uses the pointers to the sibling nodes instead of inaccessible ones. The proposed schemes reduce the waste of index space on wireless channel and gives *tuning time* reduction. The main contributions of our work can be summarized as follows.

- A new broadcast structure based on the data distribution for the *NN* query processing is proposed.
- The client can perform the *NN* query processing without an index segment. In this case, the best *access time* is obtained, since no index is broadcast along with the file [10].
- The proposed schemes help the clients to selectively tune only relevant data items, without an index segment.

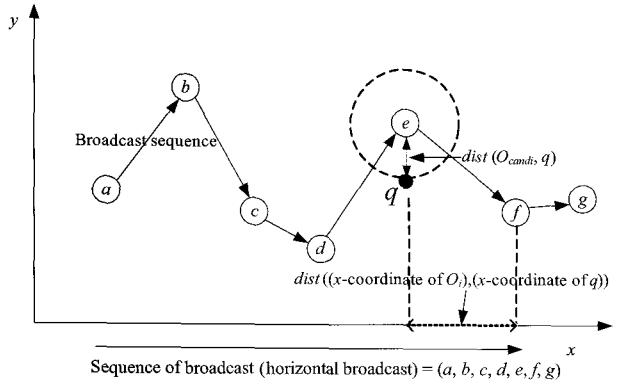


Fig. 3. An example of horizontal broadcast.

- The performance of the proposed schemes is investigated by an analytic and simulation study, representing different environments.

We believe this is the first groundbreaking work on supporting energy efficient *NN* query processing considering both the client's *tuning time* and sleep time.

III. VD-BASED INDEXING FOR NN SEARCH

In this section, we first introduce the broadcast-based LDIS scheme (BBS) [6]. Then, we describe the VD-based energy efficient selective tuning method. Finally, we present adjacency data prefetching and caching methods in a way with the aim of reducing the client's *access time* and energy consumption.

A. Broadcast Sequence for NN Search

In a recent paper [6], we have proposed the concept of data sorting for broadcasting called broadcast-based location dependent data delivery scheme (BBS). In the BBS method, the server periodically broadcasts the IDs and coordinates of the data objects, without an index segment, to the clients, and these broadcasted data objects are sorted sequentially, according to the location of the data objects, before being sent to the clients. In this method, since the data objects broadcasted by the server are sequentially ordered based on their location, it is not necessary for the client to wait for the index segment, if the desired data object is able to be identified before the associated index segment has arrived. In this method, the structure of the broadcast affects the distribution of the data object. The BBS provides the fastest *access time*, since no index is broadcasted along with the data and thus, the size of the entire broadcast cycle is minimized. A preliminary simulation-based results showed that BBS is significantly reduced the *AT*.

A simple sequential broadcast can be generated by linearizing the two dimensional coordinates in two different ways, i.e., horizontal broadcasting (HB) or vertical broadcasting (VB). In HB, the server broadcasts the location dependent data (LDD) in horizontal order, that is, from the leftmost coordinate to the rightmost coordinate. On the other hand, in VB, the server broadcasts the LDD in vertical order, that is, from the bottom coordinate to the top coordinate. In this paper, we assume that the server broadcasts the data objects using HB.

Definitions of symbols and parameters:

- S: Server data set.
- T_i: Boundary lines of the current broadcast data object.
- T: Set of T_i.
- O_i: Broadcast data object, where $O_i \in S$.
- O_c: Current broadcast data object (initially O_c regarded as *NN*), where $O_c \in S$.
- O_{FP}: Data object of FP.
- O_f: The client's first tuned data item in the broadcast channel.
- $Data\_first$: The server's first broadcast data item in the current broadcast cycle.
- TS: Safe nearest boundary line on the left-hand side of the q, where (x-coordinate of TS) $\leq$ (x-coordinate of nearest boundary line on the left-hand side of the q).

Lemma 1: While the data objects are sequentially broadcasted in horizontal order, from the leftmost coordinate to the rightmost coordinate, if $O_c = O_i$ and $dist$ ((x-coordinate of O_i), (x-coordinate of q)) $> dist$ (candidate for the nearest data object, q), then O_i and the rest of the broadcast data objects are located outside the *NN* range.

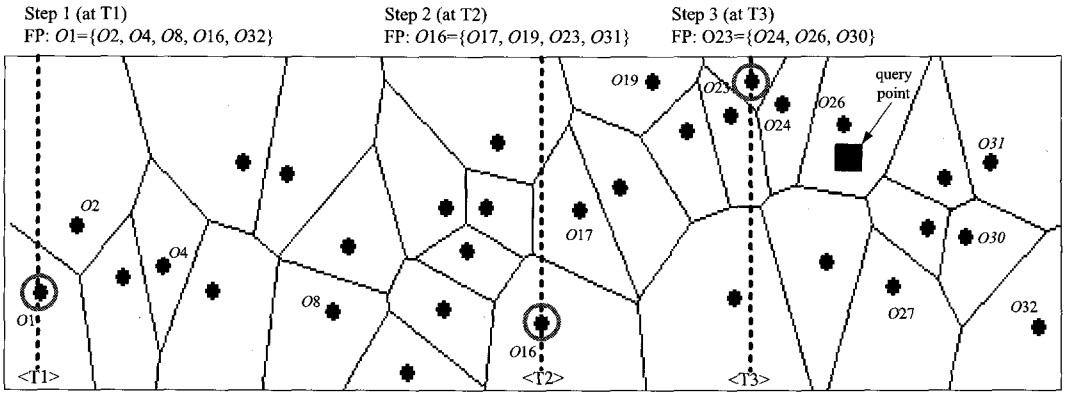
Proof: Consider the example in Fig. 3. Given a query point 'q', let the candidate for the nearest data object be the object 'e' and O_c be the object 'f'. If $dist$ ((x-coordinate of the object 'f'), (x-coordinate of 'q')) $> dist$ ('e','q'), then the objects 'f' and 'g' are located outside the *NN* range and thus the client stops tuning the broadcast channel and selects the object 'e' as the *NN*. □

B. VD-Based Energy Efficient Selective Tuning

Previous index schemes use an index to conserve battery power consumption. By replicating for index m times, the waiting time for reaching the root node of a forthcoming index segment can be reduced. However, this method has the worst possible latency because clients have to wait until the beginning of the next broadcast cycle, even if the desired data is just in front of them [5]. In addition, if a user's location changes, the previous result may expired. Hence, the client has to tune the broadcast channel repeatedly while it moves. We conclude that existing indexing methods are unsuitable for LDISs. With the BBS scheme [6], the client can significantly reduce the average *access time*, since it eliminates the *probe wait* time for the clients. However, *tuning time* may increase, since the client has to tune the broadcast channel until the desire data item is arrived. In the previous index schemes [4], [5], each data item contains a pointer to the next occurrence of the index segment and additionally, every data item contains a pointer to the next data item that is the same value of the attribute [5], [11]. In our scheme, all data items contain pointers that contain IDs, locations and arrival times of the data items to be broadcast afterward. In this section, we present an energy efficient scheme under the BBS environment namely, exponential sequence scheme (ESS) with the VD method. The VCs can be used to find out *NN*. The present scheme provides the ability to selective tune the clients and therefore helps reduce the client's *tuning time*.

B.1 Exponential Sequence Scheme (ESS)

In this section, we present a selective tuning method for use in the BBS environment. In this method, the client uses expo-



Broadcast sequence = {$O1$, $O2$, $O3$, $O4$, $O5$, $O6$, $O7$, $O8$, . . . , $O32$}

Fig. 4. An example of exponential sequence scheme.

nential pointers from each data item for the purpose of reducing energy consumption. The server broadcasts data items and each data item contains the following information.

- It's ID and Voronoi edge (VE: Vertex pointers).
- Initial pointer (IP): Arrival time of the first broadcast data item for the next broadcast cycle.
- Forward pointer (FP): IDs, it's VE and arrival times of the data items that will be broadcasted at T_i. $T = \sum_{i=1}^{\log_e N} T_i$, from the each data item, where N be the number of data items and e be exponent value (e.g., if $e = 2$, the data item 1 has the following IDs and related information: The data items located in $O2$, $O4$, $O8$, $O16$, and $O32$ (see Fig. 4)).

The client obtains the IDs, VEs, location information and FP from the initial data object obtained at the broadcast channel. Then, it checks the VEs of the current data item and FP in order to find out the object that contains query point q within the VC. In other words, the client checks VC of O_f and O_{FP}, in order to find out *NN*. If there is no data object that contains query point q within the cell, then it switches to doze mode until the desired data item appears on the broadcast channel. The client repeatedly switches between the doze and wake up modes until it retrieves the *NN*.

Definition 1: A data object O_i is regarded as *NN*, if VC of O_i contains query point q.

Let us consider the example in Fig. 4. We assume that the server broadcasts the data objects using HB. In HB, the server broadcasts the data in horizontal order, that is, from the leftmost coordinate to the rightmost coordinate. The following summarizes the steps taken by the client to process the *NN* search by using the FP.

Step 1: The client tunes into the broadcast channel at T1 and obtains the following FP: ID=$\{2, 4, 8, 16, 32\}$, their arrival times and VEs from data item $O1$. Then the client checks the VEs of each date object $O1$, $O2$, $O4$, $O8$, $O16$, and $O32$ in order to find out the VC that contains query point q (see Definition 1).

Action: Since $O1$, $O2$, $O4$, $O8$, $O16$, and $O32$ does not contain the query point q, the client switches to doze mode and sleeps until the nearest data object on the left-hand side of the query point q up to the present time (i.e., $O16$) has appeared on the broadcast channel.

Step 2: The client wakes up and tunes into the broadcast channel at T2 and obtains the following FP: ID=$\{17, 19, 23, 31\}$, their arrival times and VEs from data item $O16$. Then the client checks the VEs of each date object $O16$, $O17$, $O19$, $O23$, and $O31$ in order to find out the VC that contains query point q.

Action: Since $O16$, $O17$, $O19$, $O23$, and $O31$ does not contain the query point q, the client switches to doze mode and sleeps until the nearest data object on the left-hand side of the query point q up to the present time (i.e., $O23$) has appeared on the broadcast channel.

Step 3: The client wakes up and tunes into the broadcast channel at T3 and obtains the following FP: ID=$\{24, 26, 30\}$, their arrival times and VEs from data item $O23$. Then the client checks the VEs of each date object $O23$, $O24$, $O26$, and $O30$ in order to find out the VC that contains query point q.

Action: Since VC of $O26$ contains the query point q, the client stops selective tuning and returns $O26$ as the *NN*.

Fig. 5 shows the example of the client's selective tuning.

B.2 Safety Boundary Line

The problem that we have to consider is the case of the missing *NN*. Let us consider the case where the server broadcasts data items using ESS and the client starts to tune to the broadcast channel at T1 (see Fig. 6). Now, the client obtains the following FP: ID=$\{2, 4, 8, 16, 32\}$ from data item $O1$ at T1. After the client obtains the FP from data item $O1$, it switches to doze mode and wakes up at T5, since T5 is the nearest boundary line on the left-hand side of the q up to the present time. However, in this case, the client misses the *NN* ($O15$), even though $O15$ is the nearest data object from the query point q. That is, while the server broadcasts data items in horizontal order, if $dist$ (x-coordinate of O_i, x-coordinate of q) $\leq$ $dist$ (data object of (nearest boundary line on the left-hand side of the q), q), where x-coordinate of O_i $<$ x-coordinate of data object of (nearest boundary line on the left-hand side of the q) (data object of nearest boundary line on the left-hand side of the q), then the client cannot guarantee that it does not miss the *NN* in the current broadcast cycle. In order to prevent the above problem from arising, the client must check for the following condition.

Lemma 2: To guarantee that the client does not miss the *NN*, the $dist$ (x-coordinate of TS, x-coordinate of q) must be longer

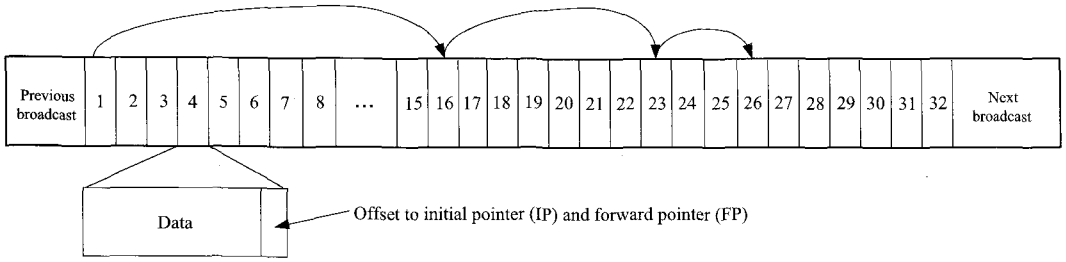


Fig. 5. An example of the client's selective tuning.

than $dist$ (data object of nearest boundary line on the left-hand side of the q, q).

Proof: While the server is broadcasting the data objects sequentially, from the leftmost data object to the rightmost data object, if the client tunes to the data item before TS, where $dist$ (x-coordinate of TS, x-coordinate of q) (see Fig. 6: $dist$ A) > $dist$ (data object of nearest boundary line on the left-hand side of the q (e.g., $O16$), q) (see Fig. 6: $dist$ B), the client misses any *NN*. For example, let the client start to tune to the broadcast channel at T4 (see Fig. 6). Since $dist$ (x-coordinate of T4, x-coordinate of q) is longer than $dist$ (data object of nearest boundary line on the left-hand side of the q, q), any data object whose x-coordinate is T4 or less cannot be the *NN*. Thus, if the client starts tune the broadcast channel at TS or before TS, it misses any *NN* that meets this condition. □

Lemma 3: If $dist$ (x-coordinate of O_f , x-coordinate of q) < $dist$ (data object of nearest boundary line on the left-hand side of the q, q), then the client cannot guarantee that it does not miss the *NN* in the current broadcast cycle.

Proof: Let the client begin to tune at time T5 (see Fig. 6). At T5, $O_c = O16$ and query point q does not belong to VC of $O16$. Now, $O16$ is the currently nearest object from the query point q. Next, $O_c = O17$ and the query point q does not belong to VC of $O17$. Now, $O16$ is the currently nearest object from query point q, since $dist\ (O16, q) < dist\ (O17, q)$. Next, $O_c = O18$ and the query point q does not belong to VC of $O18$. Now, $O16$ is currently the nearest object from the query point q, since $dist\ (O16, q) < dist\ (O18, q)$. Next, $O_c = O19$ and the query point q does not belong to VC of $O19$. Now, $O16$ is the currently nearest object from the query point q, since $dist\ (O16, q) < dist\ (O19, q)$. Then, the client stops tuning to the broadcast channel, since $dist\ (O16, q) < dist$ (x-coordinate of $O19$, x-coordinate of q) (see Lemma 1). However, the client could not return $O16$ as the *NN*, since the query point q dose not belong to VC of $O16$. Thus, the client fails to retrieve the *NN*. □

Theorem 2: While the data objects are sequentially broadcasted in horizontal order, from the leftmost coordinate to the rightmost coordinate, if x-coordinate of O_f > x-coordinate of q, then the client cannot guarantee that it dose not miss *NN* in the current broadcast cycle.

The client uses following algorithm to find out *NN*

Algorithm 1. the client algorithm used to identify the nearest object

Input: locations of the clients and the data objects;
Output: *NN*;
Procedure:

```
1: do {
2: read O_i
3:   if (x-coordinate of O_f > x-coordinate of q and O_f ≠ Data_first)
4:       then switch to doze mode until Data_first comes
         O_i=Data_first
5:   else if (satisfy the Lemma 3)
6:       then switch to doze mode until Data_first comes
         O_i=Data_first
7:   else
8:       for T
9:           do check VE of O_i and O_FP to identify that VC of O_i or O_FP
             contains query point q
10:              if VC of O_i or O_FP contains query point q
11:                  then return O_i or O_FP as NN
12:              else
13:                  find (nearest boundary line on the left-hand side of the q)
                     on the left hand side of q (e.g., T5 in Fig. 6) and TS
                     (e.g., T4 in Fig. 6, see Lemma 2), then switch to doze
                     mode until the data object of TS appears on the channel
14:                  O_i= object of (nearest boundary line on the left-hand
                     side of the q)
15: }
16: while (find out NN)
```

C. Adjacency Data Caching (ADC)

As we mentioned before, if the user's location has changed, the previous result of the query might be invalid. Therefore, the client has to tune the broadcast channel repeatedly if it moves to the other VC. Data prefetching and caching have been proposed as a technique for hiding the access latency and reducing the wireless bandwidth requirement. Indeed, prefetching can be performed without adding any load on shared resources like wireless bandwidth [12]. The client prefetches pages in anticipation of future accesses. In this section, we present adjacency data prefetching and caching methods for use in LDIS. In this method, the client prefetches the data objects that are adjacently located in the map for future use.

Given a VD and the query point q, we can identify the VC containing q. Moreover, since the broadcasted data objects are sorted sequentially according to the location of the data objects, clients can prefetch adjacent located data objects. Let w_p denote the size of prefetched data objects. Prefetching can be categorized into pre-fetch and post-fetch, where pre-fetch represents fetching data objects before *NN* data item is broadcast while post-fetch represents fetching data objects after the *NN* data object is broadcasted, respectively. The value of pre-fetch and post-fetch represents the number of data items to be prefetched. The client adjusts the size of w_p according to the client's moving direction, speed or the cache size. It should be noted that prefetching does not necessarily require continuous active mon-

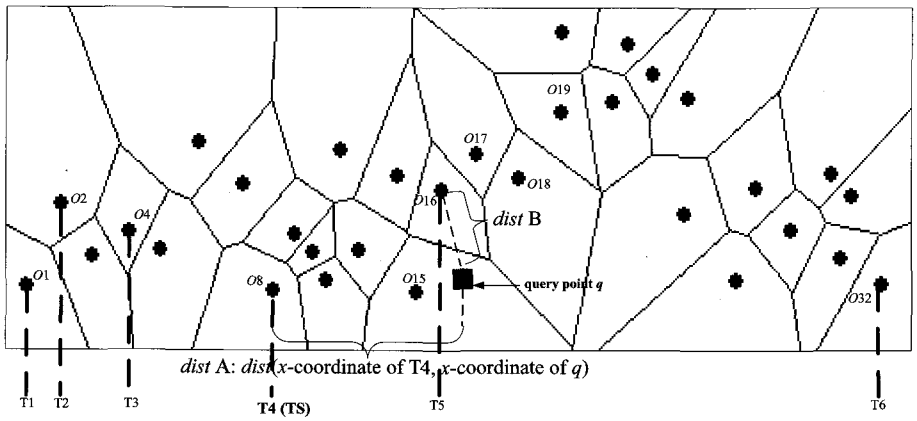


Fig. 6. An example of safety boundary line.

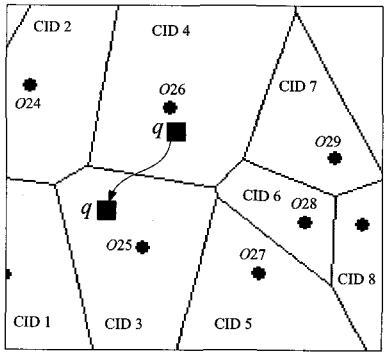


Fig. 7. Broadcast sequence= $\{O1, \cdots, O24, O25, O26, O27, O28, O29, \cdots, On\}$ and cache state=$\{O24, O25$ (*NN*, when the client located in CID3), $O26$ (*NN*, when the client located in CID4), $O27$, $O28$, $O29\}$.

itoring of the broadcast channel to determine what the next arriving data object is, if the client knows in advance the broadcast schedule. Cached data items contain the information of the object's ID and it's VE. Let us consider the case where the server broadcasts data items using the BBS and the client detects the nearest data object from the query point q as $O26$ (see Fig. 6). Let the value of the pre-fetch be 2 and post-fetch be 3 on the basis of the *NN*. In this case, the client wakes up at T4 and prefetch data objects as the following sequential order:=$\{O24$, $O25$, $O26$, $O27$, $O28$, $O29\}$, as shown in Fig. 7. The client cached the data object from $O24$ to $O29$ while it located at Voronoi cell ID (CID) 4. Then, the client moves from CID 4 to CID 3 (see Fig. 7). The client checks the prefetched data items in CID 3 instead of tuning the broadcast channel again and returns $O25$ as newly nearest data object, if it receives the same query.

Definition 2: While the server broadcasts data items using horizontal broadcasting (HB), x-coordinate of pre-fetch $\leq$ x-coordinate of *NN* $\leq$ x-coordinate of post-fetch.

Although the previous index schemes, such as $(1, m)$ index [5] can also be adapted in ADC, in this case, *AT* and *TT* may significantly increase, since the client has to repeatedly switch between the doze and wake up modes until it obtains the desired data items.

The formal description of the algorithm used for ADC at the client side is as follows.

Algorithm 2. Client algorithm for data prefetching

Input: sorted broadcast data objects;
Output: set of data items within w_p;
Procedure:

1: **do** {
2: active mode (selectively)
3: identify its current location and the *NN* (using Algorithm 1)
4: **then** doze mode until the desire data item arrives on the broadcast channel
5: wake up and prefetch the data items from pre-fetch to post-fetch on the basis of the *NN*
6:}
7: **while** (prefetches the desire data items)
8: doze mode

Let $V(O_x^c)$ be the validity region of data item in the cache and S^c be the set of data items in the cache. The following summarizes the steps taken by the client to process the *NN* search:

Step 1: When a query is issued by a mobile client, it first checks the $V(O_x^c)$, where $V(O_x^c) \in S^c$. If $V(O_x^c)$ contains the query point q, go to step 2; otherwise, go to step 3.

Step 2: If $V(O_x^c)$ contains the query point q, the data item is retrieved locally. Go to step 5.

Step 3: If $V(O_x^c)$ does not contain the query point q, the client starts tune the broadcast channel in order to process the *NN* query.

Step 4: The client switches to doze and wake up mode until *NN* has appeared on the broadcast channel. Obtain the *NN* through the broadcast channel.

Step 5: A result is returned to the client.

IV. PERFORMANCE EVALUATIONS

In this section, we evaluate the performance with various kinds of parameters settings such as the client's speed, the size of the service area, and the distributions of the data items. In Section IV-A, we analyze the proposed approaches. Then, we present performance results of simulation experiments in Section IV-B.

A. Analytic Evaluation

A.1 Access Time

In this section, we compare the *access time* of the BBS with $(1, m)$ index. The following shows comparison of the *probe wait* and the *Bcast wait* between BBS and previous index method [4], [5]. Let AAT be the average *access time*, m be the number of times broadcast indices, N be the number of data items and C be the average number of buckets containing records with the same attribute value.

probe wait:

- Previous index method: $\frac{1}{2}(index + \frac{N}{m})$
- BBS method: None

Bcast wait:

- Previous index method: $\frac{1}{2}((m \cdot index) + N) + C$
- BBS method: $\frac{1}{2}N + C$

Since the AT is the sum of the *probe wait* and the *Bcast wait*, average AT for **previous index method** is

$$\begin{aligned} AAT_{\mathrm{PRE}} &= \frac{1}{2}(index + \frac{N}{m}) + \frac{1}{2}((m \cdot index) + N) + C \\ &= \frac{1}{2}((m+1)index + (\frac{1}{m} + 1)N) + C. \end{aligned} \quad (2)$$

BBS is

$$AAT_{\mathrm{BBS}} = \frac{1}{2}N + C. \quad (3)$$

A.2 Tuning Time

In this section, we evaluate the *tuning time* for the proposed schemes with $(1, m)$ index. The probability distribution of the initial probe of clients is assumed to be uniform within a broadcast and data items of the same size.

Let ATT be the average *tuning time* and l be the number of levels in the multileveled index tree. The ATT for $(1, m)$ index is

$$ATT_{\mathrm{PRE}} = 1 + l + C. \quad (4)$$

The ATT of BBS is as follows.

Let m denote the number of times broadcast indices, k' denote the number of levels in the index tree for BBS and P denote the probability.

$P\{$containing the desired data object among the index$\}$ is $\frac{1}{m}$, and then, $P\{$obtaining the desired data object$\}$ is $\frac{1}{2m}$.

Thus, ATT of BBS is $P\{$obtaining data object without an index$\}\times$ cost of reading data objects $+P\{$failure obtaining the desired data object after read the index$\}\times$ cost of obtain the desire data object after read the index, and thus

$$\begin{aligned} f(m) &= \frac{1}{2m}\left(\frac{Data}{m}\frac{1}{2}\right) + \frac{2m-1}{2m}\left(\frac{Data}{m}\frac{1}{2} + k' + 1\right) \\ &= \frac{Data - k' - 1}{2}m^{-1} + k' + 1 \end{aligned}$$

thus,

$$ATT_{\mathrm{BBS}} = \frac{Data - k' - 1}{2}m^{-1} + k' + 1. \quad (5)$$

Finally, we evaluate the ATT for ESS.

Let N be the number of data items, e be the exponent value and ATT_{ESS} be the average *tuning time* for ESS. The minimum number of steps is 1 and the maximum number of steps is $k-1$, where $k = \lceil \log_e N \rceil$. For example, if $N = 1024$ and $e = 2$, then in the best case, the client obtains the desired data item within a single step while, in the worst case, the client obtains the desired data item within 9 steps. The frequency of the worst case for N is 1, while the frequency of the best case for N is k.

The ATT for

$$\begin{aligned} ATT_{\mathrm{ESS}} &\approx \frac{k2^{k-1}\sum_{i=0}^{e-1} i}{N} \\ &\approx \frac{ke(e-1)}{4}. \end{aligned} \quad (6)$$

Assuming that the number of broadcast data items is 1024. By using the above cost model, we can obtain the ATT_{ESS} as the value of e is increased from 1 to 4. If $e = 1$, $ATT_{\mathrm{ESS}} = \frac{k1(1-1)}{4} = 0$, if $e = 2$, $ATT_{\mathrm{ESS}} = \frac{k2(2-1)}{4} = 5$, if $e = 3$, $ATT_{\mathrm{ESS}} = \frac{k3(3-1)}{4} = 15$ and if $e = 4$, $ATT_{\mathrm{ESS}} = \frac{k1(4-1)}{4} = 30$.

A.3 Energy Consumption

In order to evaluate the energy consumption, both doze time and active time for the client must be estimated. As the result of the following analytic evaluation, proposed methods significantly reduce the energy consumption compare to $(1, m)$ index method, since it minimize not only average *access time* but also average *tuning time*. Let $\widehat{ec}_{\mathrm{PRE}}$ be the average energy consumption of previous index method and $\widehat{ec}_{\mathrm{ESS}}$ be the average energy consumption of ESS.

$$\begin{aligned} \widehat{ec}_{\mathrm{PRE}} = (1 + l + C)\mathcal{E}_{ACTIVE} + \Big(&(\frac{1}{2}((m+1)index \\ &+(\frac{1}{m} + 1)N)) - (1 + l + C)\Big)\mathcal{E}_{DOZE} \end{aligned} \quad (7)$$

$$\widehat{ec}_{\mathrm{ESS}} \approx \frac{ke(e-1)}{4}\mathcal{E}_{ACTIVE} + (\frac{N}{2} - \frac{ke(e-1)}{4})\mathcal{E}_{DOZE}. \quad (8)$$

B. Experimental Evaluation

This section presents the numerical results obtained through analysis and simulation. For fairness, we use a single dedicated machine. The machine is a PC with a Pentium III 800 MHz, 512 MB, and running on Windows 2000. We implemented the system in Java and run it under the JVM version 1.3.0. We evaluated two different parameters: Energy consumption and *access time*. We assume that the broadcast data objects are static, such as restaurants, hospitals, and hotels. We use a system model similar to that described in [13] and [14]. The whole geometric service area is divided into groups of mobile supporting stations (MSSs). In this paper, two datasets are used in the experiments. The first data set $\mathcal{D}1$ contains data objects randomly distributed in a square Euclidian space, while the second data set $\mathcal{D}2$ contains the data objects of hospitals in the Southern California area, and is extracted from the data set at [15]. The number of switching time of the previous algorithm may smaller

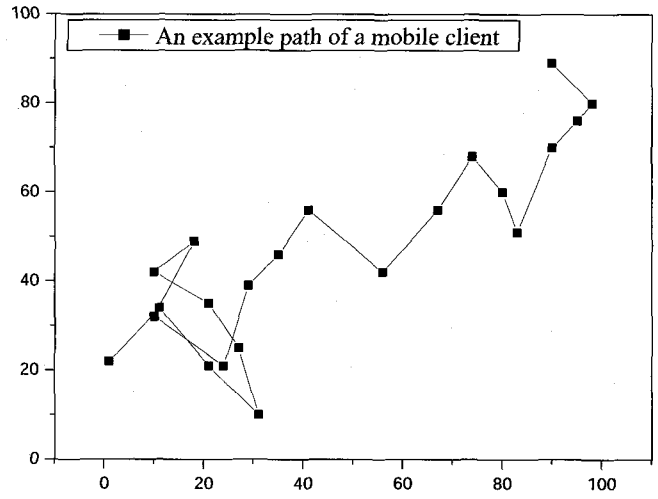


Fig. 8. An example path of a mobile client.

then the proposed algorithm, However, the maximum number of switching time for the proposed scheme is only $k-1$, where $k = \lceil \log_e N \rceil$ as we have described before. Thus, we ignore the energy consumed on switching between doze and active mode, since it barely affect the total result.

Let the clients be equipped with the hobbit chip (AT&T). Energy coefficient of the active-to-sleep ratio is fixed to 48.61. We compare the *tuning time* of proposed schemes with optimal *tuning time*. Then, convert the result of the *tuning time* into energy consumption, since the energy consumption can be measured by the amount of unit energy in a given time. Then, we evaluate the *access time* for various parameter settings, such as the client's speed, the size of the service area and the number of clients.

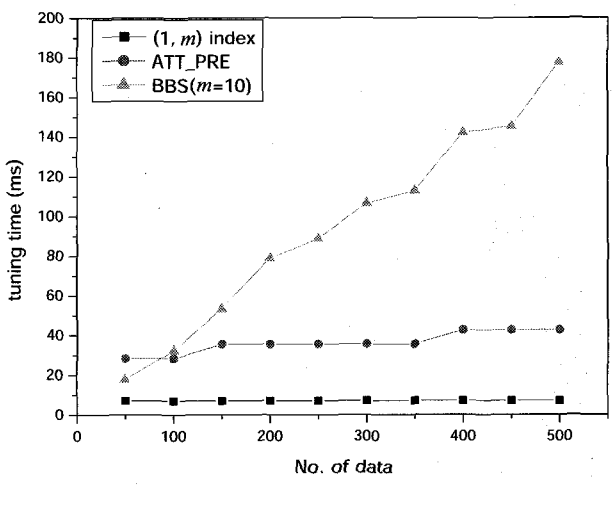
B.1 Mobility Model

In this paper, we assume that the client's mobility pattern follows random waypoint mobility model [16]. The random waypoint mobility model is also a widely used mobility model. The random waypoint mobility model includes pause times between changes in direction and/or speed. A mobile client begins by staying in one location for a certain period of time. Once this time expires, the mobile client chooses a random destination in the simulation area and a speed that is uniformly distributed between [minspeed, maxspeed]. The mobile then travels toward the newly chosen destination at the selected speed. Upon arrival, the mobile client pauses for a specified time period before starting the process again [16]. Fig. 8 shows an example traveling pattern of a mobile client using the random waypoint mobility model starting at a randomly chosen point or position $(1, 22)$; the speed of the mobile client in the figure is uniformly chosen between 0 and 90 km/h.

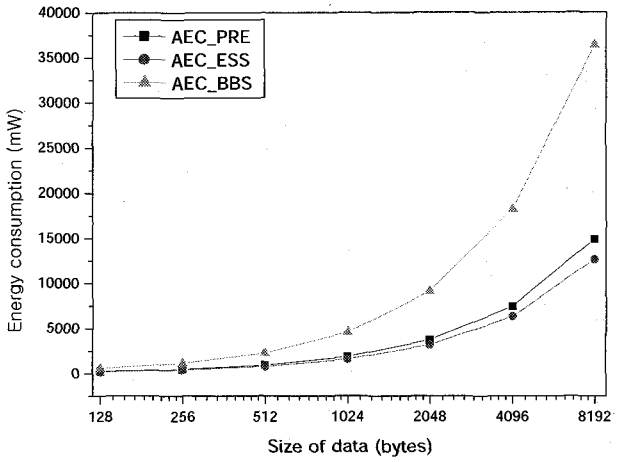
B.2 Energy Consumption Model

In this paper, there are two energy states such as *doze* mode, *active mode*. We now describe the ratio of energy consumption for these states. $\mathcal{E}_s$ describes the amount of energy consumption in an energy state s per unit time.

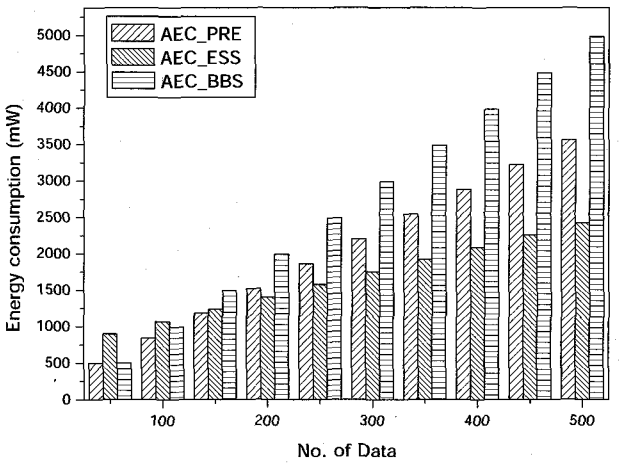
$$\mathcal{E}_{DOZE} : \mathcal{E}_{ACTIVE} = 1 : ec \tag{9}$$



(a)



(b)



(c)

Fig. 9. Tuning time and energy consumption in $\mathcal{D}1$.

In this paper, the amount of energy consumed in doze mode for unit time is denoted as *unit energy* which is 33.16 mW in our experiment. In many processors, the doze mode has extremely low power consumption. In the Hobbit chip from AT&T, for

example, the ratio of power consumption in the active mode to the doze mode is 5,000 [5]. In brief, the "ec" stands for energy coefficient which means the active-to-doze ratio, $\frac{\mathcal{E}_{ACTIVE}}{\mathcal{E}_{DOZE}}$.

In this paper, the average energy consumption can be measured by the amount of unit energy in a given time. In order to choose reasonable coefficients, we should have some reference values. Then, the energy coefficients ($\widehat{ec}$) can estimate.

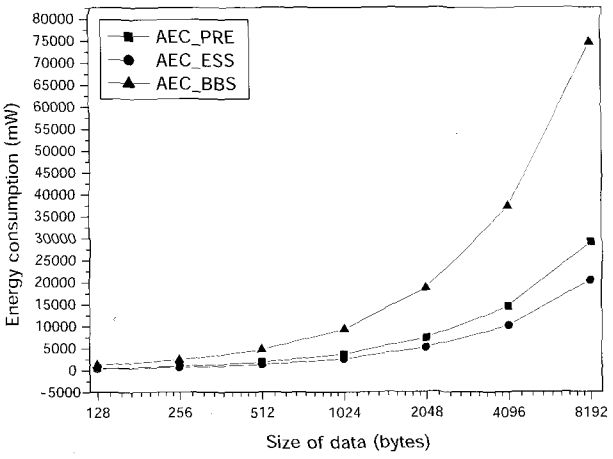
$$\widehat{ec} = \frac{\mathcal{E}_{ACTIVE}}{\mathcal{E}_{DOZE}} = \frac{(400 + 750 + 462)\,\text{mW}}{(0.16 + 0 + 33)\,\text{mW}} = 48.61. \tag{10}$$

In our experiment, parameters ec is fixed with 48.61. As developing mobile devices is trying to minimize the energy consumption in hardware design, energy coefficients will continuously increases.

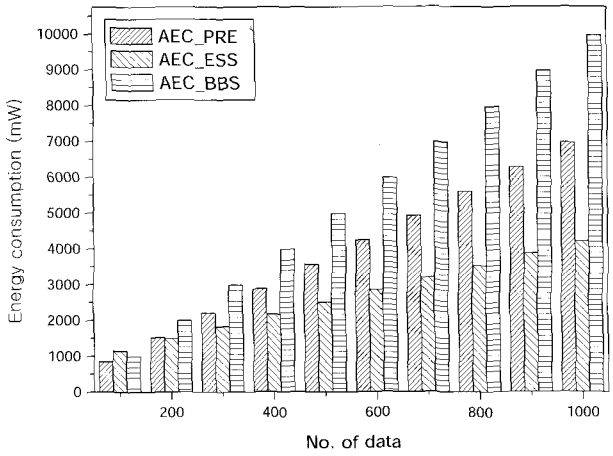
B.3 Tuning Time and Energy Consumption

Fig. 9(a) shows the result of *tuning time* as the number of data increases. As shown in the figure, AAT_PRE outperforms comparisons to others, since the average *tuning time* of tune_opt is $1 + C + k$ [5]. Fig. 9(b) shows the result of energy consumption as the size of the data is increased from 128 bytes to 8192 bytes in $\mathcal{D}1$. As shown in the figure, in this case, the proposed scheme ESS outperforms when compared to AEC_PRE, since AEC_PRE minimizes the *tuning time* but does not minimize *access time*, while ESS reduces the *tuning time* and *access time* simultaneously. In other words, when we estimate the energy consumption, it has to consider not only active time, but also sleep time even if sleep time is smaller than active time (i.e., we assume $1/48.61$). Fig. 9(c) shows the results of the energy consumption as the data increases in $\mathcal{D}1$.

Figs. 10(a) and 10(b) show the results of the energy consumptions as the size and frequency of data increases in $\mathcal{D}2$.



(a)



(b)

Fig. 10. Energy consumption in $\mathcal{D}2$.

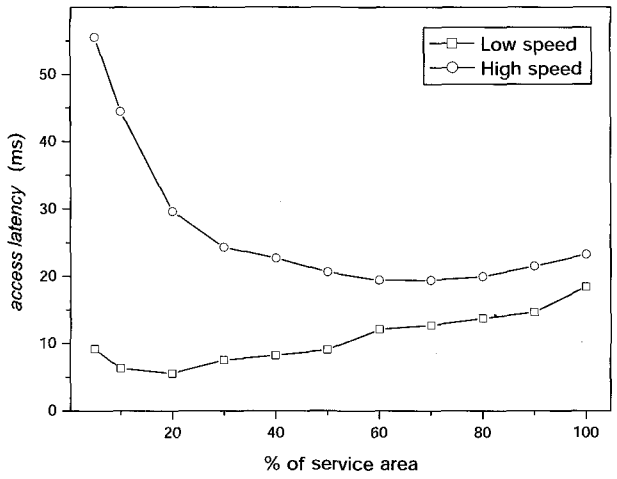
B.4 Access Time

In this section, we evaluate the AT for various parameter settings, such as the client's speed, the size of the service area and the number of clients. First, we study the effect of the size of the service area according to the client's speed. We vary the service coverage area from 5% to 100% of the whole geographic area. As the size of the entire broadcast is reduced, the query arrival time decreases and the size of the service area decreases. However, the query arrival time is significantly increased when the client's speed increases and the client goes outside of the service coverage area, as shown in Fig. 11(a). Then, we study the effect of the client speed. We vary the client's speed from 5 to 50 in $\mathcal{D}1$. When the client's speed is the lowest, broadcast size of 10% (of the coverage area) is the best. However, as the client's speed increases, it's performance is inadequate when compared with that of the other clients, since most of the clients are outside the service coverage area, as shown in Fig. 11(b). Figs. 12(a), 12(b), and 12(c) show the results of the access latency as the number. of clients, size of data and the number of data increases in $\mathcal{D}\infty$, respectively.

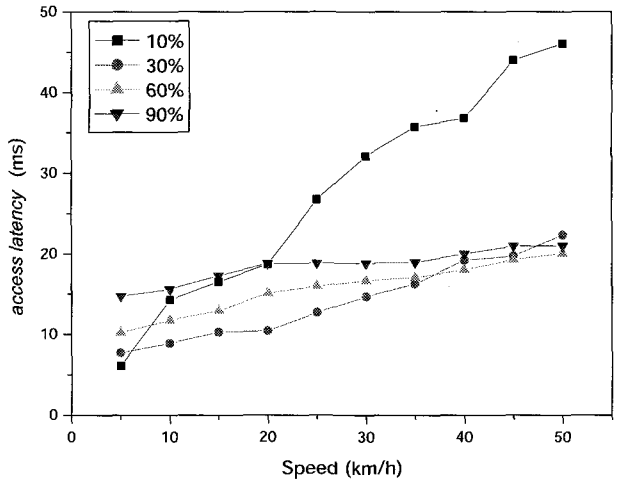
Finally, Figs. 13(a) and 13(b) show the result of the access latency as the number of clients and the size of data increases in $\mathcal{D}2$, respectively.

B.5 Cache Hits

This section evaluates the cache hit ratio for various parameters settings such as the size of the cache, the client's speed and the number of clients. First, we vary the size of the cache from 3072 to 11264 bytes. In this experiment, we assume that all data items are of the same size, for example 1024 bytes. Fig. 14(a) shows the result of the number of cache hits as the size of the cache is increased. As shown in the figure, adjacency data caching (ADC) outperforms $(1, m)$ index method, since the ADC caches the sequentially ordered data items according to their locations, whereas the $(1, m)$ index method caches the irregularly ordered data items. Evidently, in the $(1, m)$ index method, the average *access time* may significantly increase if the client is attempting to cache the data items with the sequential order, since the broadcast data items are not ordered based on their locations. Second, we vary the client's speed from 10 to 100 in $\mathcal{D}1$. As shown in Fig. 14(b), the number of cache hits decreases as the client's speed increases. Third, we vary the number of clients from 10 to 50 in $\mathcal{D}1$. As shown in Figs. 14(b) and 14(c), ADC outperforms the $(1, m)$ index method for the same reason shown above.



(a)

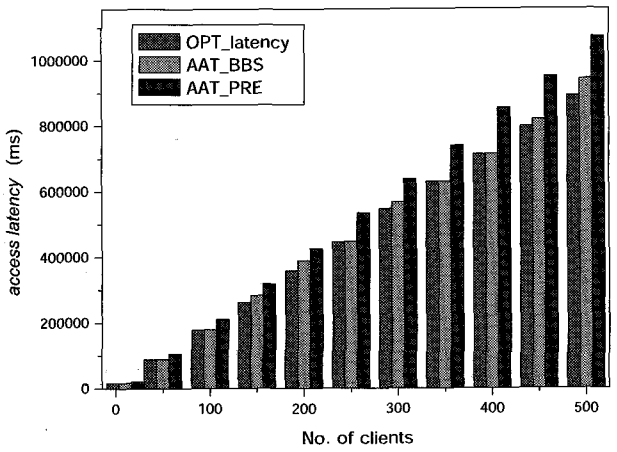


(b)

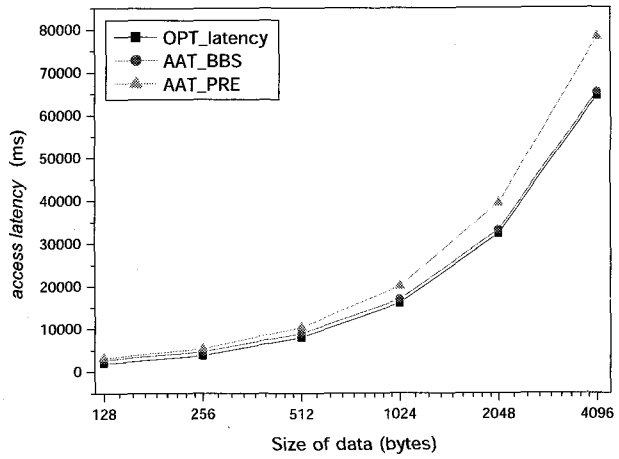
Fig. 11. *access latency*.

V. CONCLUSION

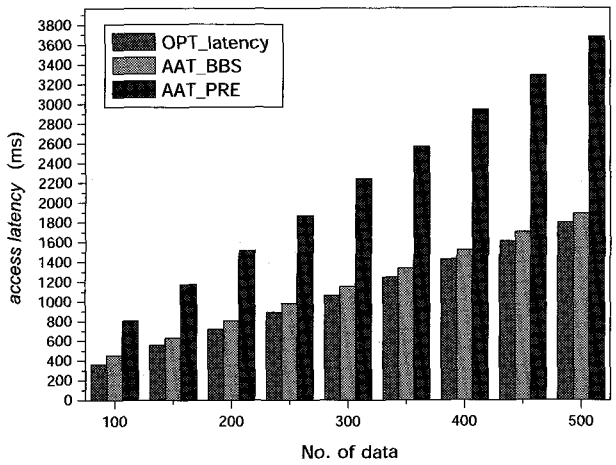
We have explored the different broadcasting and tuning methods, which can be used for *NN* query processing. For the purpose of broadcasting in LDISs, we present the BBS and for the purpose of selective tuning with the BBS, we present the ESS method. The BBS method attempts to reduce the *AT* and the ESS methods attempt to conserve battery power consumption. With the proposed schemes, the client can perform *NN* query processing without having to tune an index segment. We also present the proposed schemes as investigated in relation to various environmental variables, such as the distributions of the data objects, and the average speed of the client and the size of the service area. The experimental results show that the proposed schemes significantly reduce not only access latency, but also energy consumption, since the client does not always have to wait for the index segment. The resulting latency and *tuning time* is close to the optimum as our analysis and simulation results indicate. In a future study, we plan to investigate the cache replacement scheme. We also have plan to consider the client's speed and direction and geographical constraint in order to enhance the performance of the prefetching and the caching method.



(a)

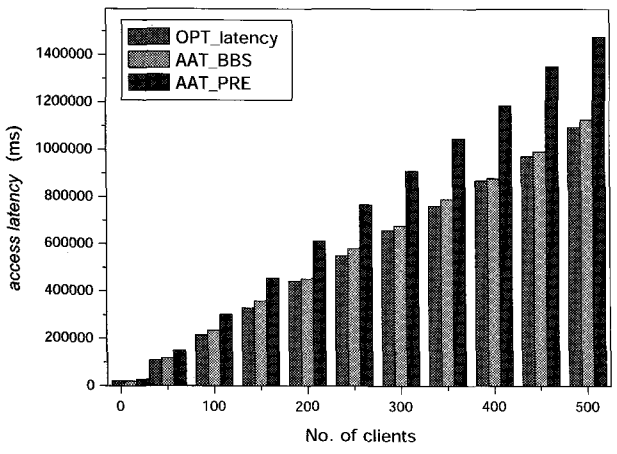


(b)

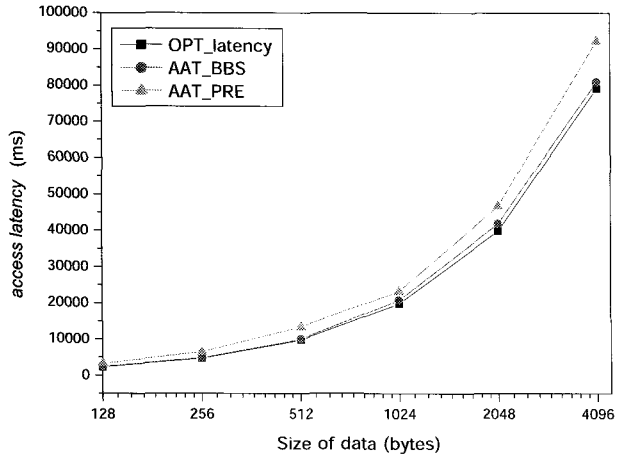


(c)

Fig. 12. *access latency* in $\mathcal{D}1$.

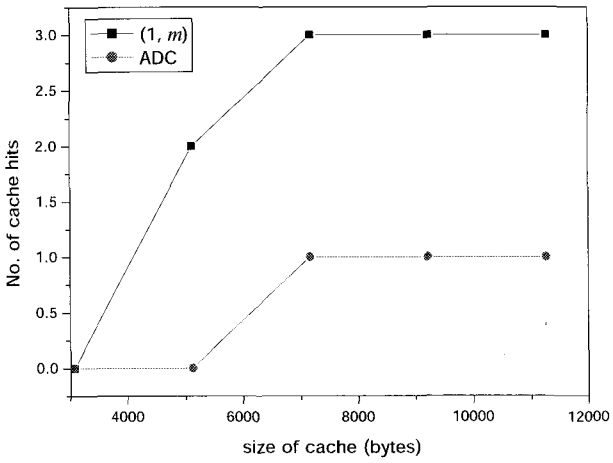


(a)

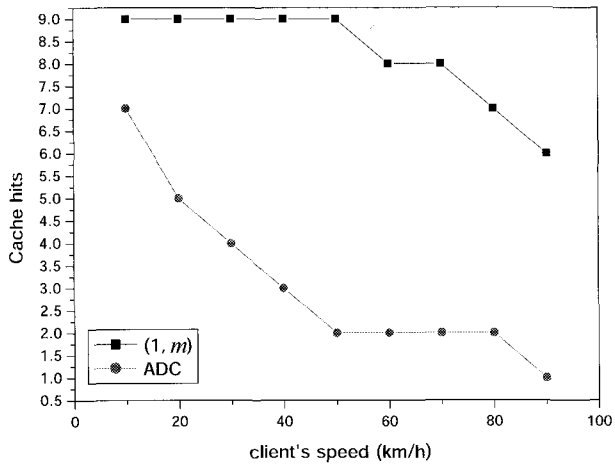


(b)

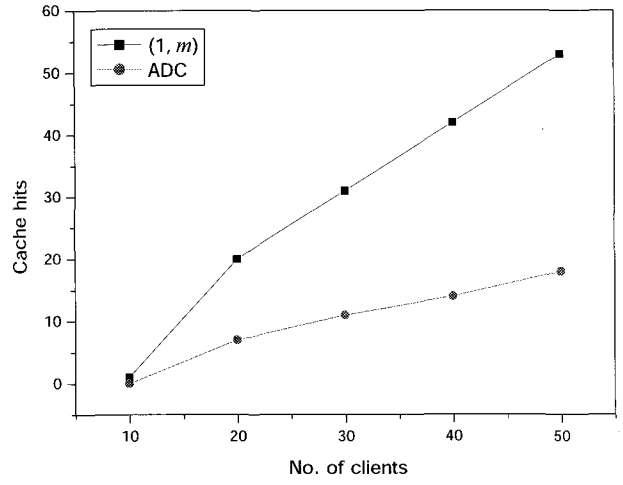
Fig. 13. *access latency* in $\mathcal{D}2$.



(a)



(b)



(c)

Fig. 14. Number of cache hits.

REFERENCES

[1] J. Xu, X. Tang, D. L. Lee, and Q. Hu. "Cache coherency in location-dependent information services for mobile environment," in *Proc. Int. Conf. Mobile Data Access'99*, 1999.

[2] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee, "Location-based spatial queries, " in *Proc. SIGMOD 2003*, 2003.

[3] X. Yang and A. Bouguettaya, "Broadcast-based data access in wireless environments," in *Proc. EBDT 2002*, 2002.

[4] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Energy efficient indexing on air," in *Proc. SIGMOD'94*, 1994.

[5] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Data on air: Organization and access," *IEEE Trans. Knowledge Data Eng.*, 1997.

[6] K. Park, M. Song, and C.-S. Hwang, "Broadcasting and prefetching schemes for location dependent information services," in *Proc. W2GIS 2004*, 2004.

[7] B. Zheng, W.-C. Lee, and D. L. Lee, "On semantic caching and query scheduling for mobile nearest-neighbor search," *Wireless Network*, 2004.

[8] B. Zheng and D. L. Lee, "Semantic caching in location-dependent query processing, " in *Proc. Int. Symposium on Spatial and Temporal Databases*, 2001.

[9] Y. D. Chung and M. H. Kim, "An index replication scheme for wireless data broadcasting," *J. Syst. Software*, 2000.

[10] D. L. Lee, J. Xu, and B. Zheng, "Data management in location-dependent information services," *IEEE Pervasive Computing*, vol. 1, no. 3, 2002.

[11] B. Zheng, W.-C. Lee, and D. L. Lee, "Spatial queries in wireless broadcast systems," *Wireless Network*, 2004.

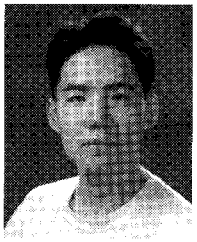
[12] V. Grassi, "Prefetching policies for energy saving and latency reduction in a wireless broadcast data delivery system," in *Proc. MSWiM 2004*, 2004.

[13] B. Zheng, J. Xu, and D. L. Lee, "Cache invalidation and replacement strategies for location-dependent data in mobile environments," *IEEE Trans. Comput.*, vol. 51, no. 10, 2002.

[14] D. Barbara and T. Imielinski, "Sleepers and workaholics: Cashing strategies in mobile environments," in *Proc. SIGMOD'94*, 1994.

[15] Spatial Datasets, available at http://dias.cti.gr/ytheod/research/datasets/spa

tial.html.
[16] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," in *Proc. IEEE WCMC 2002*, 2002.
[17] B. Zheng, W.-C. Lee, and D. L. Lee, "Search continuous nearest neighbor on air," in *Proc. Int. Conf. Mobile and Ubiquitous Systems: Networking and Services*, Boston, Massachusetts, Aug. 2004.
[18] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. SIGMOD'84*, 1984.
[19] N. Roussopoulos, S. Kelley, and F. Vincent. "Nearest neighbor queries," in *Proc. SIGMOD'95*, 1995.
[20] 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. ICDE 2003*, 2003.
[21] J. Xu, B. Zheng, W.-C. Lee, and D. L. Lee, "D-tree: An index structure for planar point queries in location-based wireless services," *IEEE Trans. Knowledge Data Eng.*, 2004.



Kwangjin Park received his B.S. and M.S. degrees in Computer Science from Korea University, Korea in 2000 and 2002, respectively. He is currently a Ph.D. candidate in Computer Science and Engineering and a researcher in the Research Institute of Computer Information and Communication, Korea University, Korea. His research interests include location-dependent information systems, mobile databases, and mobile computing systems.

Chong-Sun Hwang received his M.S. degree in Mathematics from Korea University, Seoul, Korea in 1970, and his Ph.D. degree in Statistics and Computer Science from the University of Georgia in 1978. From 1978 to 1980, he was an assistant professor at South Carolina Lander State University. He is currently a full professor in the Department of Computer Science and Engineering at Korea University, Seoul, Korea. Since 1995, he has been a dean in the Graduate School of Computer Science and Technology at Korea University. His recent research interests include distributed systems, distributed algorithms, and mobile computing systems.