

Nearest Neighbor Queries with Data Sharing in Mobile Environments

Wei-Shinn Ku, Roger Zimmermann, and Chi-Ngai Wan
Computer Science Department
University of Southern California
Los Angeles, California 90089. USA
[wku, rzimmerm, cwan]@usc.edu

Abstract

Mobile clients feature increasingly sophisticated wireless networking support that enables real-time information exchange with remote databases. Location-dependent queries, such as determining the proximity of stationary objects (e.g., restaurants and gas stations) are an important class of inquiries. We present a novel approach to support nearest-neighbor queries from mobile hosts by leveraging the sharing capabilities of wireless ad-hoc networks. We illustrate how previous query results cached in the local storage of neighboring mobile peers can be leveraged to either fully or partially compute and verify spatial queries at a local host. The feasibility and appeal of our technique is illustrated through extensive simulation results that indicate a considerable reduction of the query load on the remote database. Furthermore, the scalability of our approach is excellent because a higher density of mobile hosts increases its effectiveness.

1 Introduction

Location-based queries are of interest in a growing number of applications and an important sub-class of such queries are nearest neighbor (NN) searches. Increasingly such queries are issued from mobile clients and there exist several algorithms that allow the efficient execution of NN queries on centralized databases. In this study we propose an approach that leverages short-range, ad-hoc networks to share information in a peer-to-peer (P2P) manner among mobile clients to answer location-based nearest neighbor queries. Such a P2P approach can be very valuable for applications where access to the server is not always guaranteed and may be spurious at times. For example, during a natural disaster such as an earthquake or a hurricane, the communication from rescue crews to stationary databases may be intermittent. In such a scenario, P2P data sharing can provide a robust alternative where fault-resilience is naturally built into the design.

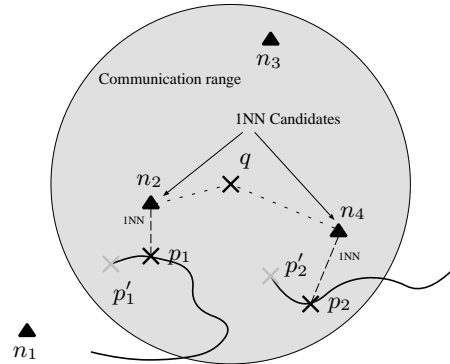


Figure 1. Nearest neighbor peer-to-peer result sharing.

The efficiency of our approach is derived from the observation that the results of spatial queries often exhibit spatial locality. For example, if two mobile hosts are close to each other, the result sets of their k NN queries for a specific object type may overlap significantly. Through mobile cooperative caching of the result sets, query results can be efficiently shared among mobile clients [3].

Figure 1 shows an example. At time T the mobile query point q can establish contact with two other mobile hosts within its communication range: p_1' and p_2' . Both of these clients in the past executed a 1NN query for the nearest fire station when they were located at p_1 and p_2 , respectively¹. The results that they obtained and cached were $\langle n_2, p_1 \rangle$ and $\langle n_4, p_2 \rangle$. These two tuples represent candidate solutions for q 's own 1NN query. Through a local verification process q can determine whether one of the solutions obtained from its neighbors is indeed its own nearest fire station. Note that the current location of the neighboring hosts, p_1' and p_2' , has no specific significance, as long as they are within q 's communication range.

The contributions of this study are as follows. We first identify a set of characteristics that enable the development

¹In our notation we use the object identifier to represent its position coordinates.

of effective sharing methods. We then introduce a set of algorithms that aid in the decision process within this distributed environment to verify whether the data items received from neighboring clients provide a complete, partial, or irrelevant answer to the posed query. Our initial method verifies results from a single neighbor, and we then extend it to work with multiple neighboring clients. Finally, through extensive simulation experiments we explore the benefits of our approach under different parameter sets (e.g., changes in the mobile host density and the wireless transmission range).

The rest of this paper is organized as follows. Section 2 surveys the related work for processing k NN queries. Our own approach is detailed in Section 3 and the experimental results are presented in Section 4. Finally, Section 5 concludes the paper and outlines future research directions.

2 Related Work

The existing work relevant to our approach can broadly be classified into *nearest neighbor query processing* and *cache management in mobile environments*.

2.1 Nearest Neighbor Query Processing

R-trees [8] and their derivatives have been a prevalent method to index spatial data and increase query performance. To find nearest neighbors, branch-and-bound algorithms have been designed that search an R-tree in either a depth-first [11] or best-first manner [9]. Both types of algorithms were designed for stationary objects and query points.

With the emergence of mobile devices attention has focused on the problem of continuously finding k nearest neighbors for moving query points (k -NNMP). A naïve approach might be to continuously issue k NN queries along the route of a moving object. This solution results in repeated server accesses and nearest neighbor computations and is therefore inefficient. One method to reduce the computational complexity is to sample the trajectory instead of treating it as a continuous curve [12].

2.2 Cache Management in Mobile Environments

Caching is a key technique to improve data retrieval performance in widely distributed environments. Leveraging the combined resources of several cooperating caches has been proposed to improve file system [5] and Web performance [13]. In conventional mobile environments, wireless connections are treated as extensions of the wired infrastructure. Hence, mobile clients retrieve information from database servers via intermediate base-stations. With the increasing deployment of new peer-to-peer wireless communication technologies (e.g., IEEE 802.11x and Bluetooth)

there exists a new information sharing alternative known as *peer-to-peer cooperative caching*. With this technique mobile hosts communicate with neighboring peers in an *ad-hoc* manner to share information rather than having to rely on the communication link to the remote information sources.

Peer-to-peer cooperative caching can bring about several distinctive benefits to a mobile system: improving access latency, reducing server workload and alleviating point-to-point channel congestion. As a disadvantage, it may increase the communication overheads among mobile hosts. Recently, a number of techniques have been proposed to address caching in ad-hoc peer-to-peer networks. The COoperative CACHing (COCA) [3] scheme investigates the effects of client activity levels, data replication, and cache size. The benefits of clustering mobile clients into groups are investigated in [4].

While there has been significant prior work in the above two areas, none of the existing techniques encompasses all the features of our proposed approach.

3 System Design

The fundamental idea behind our methodology is to leverage the cached results from prior queries at reachable mobile hosts for answering nearest neighbor queries at the local host. To achieve scalability it is imperative that a mobile client can *locally* determine whether the result set from its neighbors provides a full, partial or no answer.

As a novel component in our methodology we present a verification algorithm that can validate whether a given result object is part of the solution set. We term such an object *verified*. If the object is not guaranteed to be part of the result set, we call it *unverified*. The first variant of our verification procedure validates object certainty from a single peer. We then extend the process to multiple peers. If no full set of verified objects can be retrieved from neighboring peers, the query is forwarded to a spatial database server including the acquired partial result. The database search efficiency can be improved by utilizing the partial query results from peers.

In this study we detail how k -nearest neighbor (k NN) queries can be processed by cooperating mobile hosts. In Section 3.1 we introduce the infrastructure that we assume for our work. Next, Section 3.2 presents a three-phase algorithm for verifying query results from neighboring peers and explains how to use partial peer results to decrease the server load for processing k NN queries.

3.1 Assumed Infrastructure

We are considering mobile clients, such as cars, that are instrumented with a global positioning system (GPS) for continuous position information. Furthermore, we assume that two-tiers of wireless connections are available on future

Symbol	Meaning
P	A set of all the peers which respond the query issued by q
$p.N$	The most recent query result set of a mobile host p where $p \in P$
$p.R$	The verified region of a mobile host p
q	A query mobile host
$\ a, b\ $	The Euclidean distance between objects a and b
n_i	A nearest neighbor element in $p.N$
H	A heap for storing SBNN query results. Its verified and unverified elements are defined as H_{verified} and $H_{\text{unverified}}$, respectively.
$ A $	The number of elements in set A

Table 1. Symbolic notations.

automobiles. Traditional, cellular-based networks (such as utilized by the OnStar service) allow medium range connections to base-stations that interface with the wired Internet infrastructure. They usually charge a subscription or metered usage fee. A second type of short-range networks allow ad-hoc connections with neighboring mobile clients. Technologies that enable short range communication include, for example, IEEE 802.11x using unlicensed, free spectrum. Benefiting from the power capacities of vehicles, we assume that each mobile host has a significant transmission range and virtually unlimited lifetime. The architecture can also support hand-held mobile devices. However, then power consumption becomes an additional parameter which we are not currently considering.

3.2 Sharing Based Nearest Neighbor Queries

With the system infrastructure described in Section 3.1, a mobile host q can collect NN data from peers to harvest these existing results for completing its own Euclidean distance k NN search. We term this approach a *Sharing Based Nearest Neighbor* (SBNN) query.

We propose two approaches to process NN information obtained from peers. The single peer NN verification process, also called k NN_{single}, attempts to verify the validity of k objects by sequentially verifying results obtained from each single peer. If the number of verified objects is less than k , then the multiple peer NN verification process, k NN_{multiple}, attempts to complete the verification process with several peers simultaneously. Table 1 summarizes the symbolic notations used throughout this section.

3.2.1 Step 1: Single Peer NN Verification

The objective of the k NN_{single} method is to verify whether a point of interest (POI) n_i obtained from a peer is a valid (i.e., top k) nearest neighbor of a mobile host q . To this end we utilize the spatial relationship between mobile hosts and their POIs as follows.

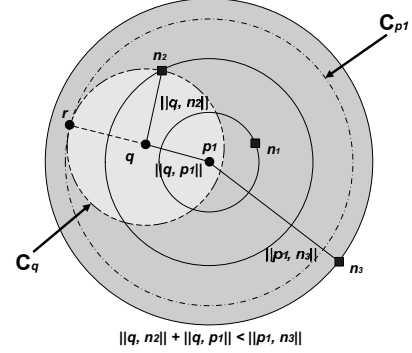


Figure 2. POI n_2 is verified as a valid NN of mobile host q .

Lemma 3.1 Let q and p_1 be two mobile hosts, and let p_1 have k nearest neighbors, n_1, n_2, \dots, n_k , which are sorted in ascending order according to their distance to p_1 . For any nearest neighbor n_i of p_1 , if $\|q, n_i\| + \|q, p_1\| \leq \|p_1, n_k\|$ then n_i is one of the top k -nearest neighbors of q .

$\|q, n_i\|$, $\|q, p_1\|$, and $\|p_1, n_k\|$ are the Euclidean distances between q and n_i , q and p_1 , and p_1 and its cached farthest nearest neighbor n_k , respectively.

Proof: Assume $n_i \notin k$ NN of q and $n_i \in k$ NN of p_1 . Then, there exist $m_1, m_2, \dots, m_k \in k$ NN of q such that $\forall m_i \in \{m_1, m_2, \dots, m_k\}, \|q, m_i\| < \|q, n_i\|$. We can identify a point r located at the intersection of the extension of the line from p_1 to q and the circumference of the circle with center q and radius $\|q, n_i\|$ (identified as circle C_q in Fig. 2). Then, $\forall m_i \in k$ NN of q :

$$\|p_1, m_i\| < \|p_1, r\| \quad (1)$$

Recall that we assume the following inequality holds:

$$\|q, n_i\| + \|q, p_1\| \leq \|p_1, n_k\| \quad (2)$$

Because the circle C_q is fully covered by the circle C_{p_1} (with center p_1 and radius $\|p_1, r\|$), it follows that

$$\|q, n_i\| + \|q, p_1\| = \|q, r\| + \|q, p_1\| = \|p_1, r\| \quad (3)$$

By Equations 1, 2 and 3, $\forall m_i \in k$ NN of q , $\|p_1, m_i\| < \|p_1, n_k\|$. Thus $n_k \notin k$ NN of p_1 . However, this contradicts the assumption that $n_k \in k$ NN of p_1 . Therefore, n_i must be one of the top k -nearest neighbors of q . ■

Verified/Unverified	V	V	UV	UV
Points of interest	n_{2-p1}	n_{1-p1}	n_{3-p1}	n_{3-p2}
Distance to q (examples)	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{5}$	$\sqrt{8}$

Table 2. The data structure of the heap H .

An illustration of Lemma 3.1 is shown in Figure 2. The nearest neighbor n_2 of mobile host p_1 , which is a peer of mobile host q , can be verified as the nearest neighbor of q and is termed a *verified nearest neighbor*. This holds because the Euclidean distance between n_2 and q plus the Euclidean distance between q and p_1 is no greater than the Euclidean distance between p_1 and its presently cached farthest nearest neighbor n_3 . The exact ranking of nearest neighbors can also be obtained (see detailed proof in [10]).

The $k\text{NN}_{single}$ method maintains a heap H with the entries of verified and unverified points of interest discovered so far (illustrated in Table 2). The size of H is determined by the total number of queried interest objects q_k . Initially H is empty and the $k\text{NN}_{single}$ method sequentially processes the result set P of nearest neighbor objects from mobile hosts in the vicinity of q . For each POI the heap H is updated with the distance from the location of q to the POI object and its validity. If there exist unverified nearest neighbor objects in H , a newly discovered verified NN object will replace an unverified object and H maintains the verified objects in an ascending order of their Euclidean distance to q . Unverified objects exist in H only if the number of verified objects is less than q_k . These unverified objects are also stored in ascending distance order.

Consider the following example to illustrate the operation of $k\text{NN}_{single}$. Figure 3 illustrates the location of q and its two closest mobile hosts, p_1 and p_2 . The single peer NN verification rule follows Lemma 3.1. Assuming that q searches for four nearest neighbors, then after processing p_1 and p_2 the content of the heap H is as shown in Table 2. Based on the set P which contains peers p_1 and p_2 , q can retrieve two verified NNs, n_{2-p_1} and n_{1-p_1} , and two unverified NNs, n_{3-p_1} and n_{3-p_2} .

$k\text{NN}_{single}$ is executed iteratively for each peer in the nearest neighbor result set P . If k elements in H are verified, the $k\text{NN}$ query is fulfilled and H will remember the top k NN in sequence. Otherwise, we must expand the search space to include more candidate interest objects. This is accomplished by the $k\text{NN}_{multiple}$ algorithm, which is described next. The single peer NN verification procedure is summarized in Algorithm 1.

3.2.2 Step 2: Multiple Peer NN Verification

Under some conditions the $k\text{NN}_{single}$ method may not be able to verify all k nearest neighbors. Therefore, we extend the verification process to include results from multiple peers simultaneously. Figure 4 demonstrates an example in which a point of interest (n_{2-p_3}) cannot be verified by $k\text{NN}_{single}$: neither with peer p_3 nor with peer p_4 . The $k\text{NN}_{multiple}$ method combines the *verified regions* of all the peers, each bounded by the outermost NN circle, into a *merged verified region* R_v (the shaded area in Figure 4c). It is computationally expensive to compute an exact solution

Algorithm 1 $k\text{NN}_{single}(q, p, k)$

```

1:  $H \leftarrow \emptyset$ 
2:  $n_{far} \leftarrow$  the farthest node in  $p.N$ 
3: for  $\forall n_i \in p.N$  and  $|H.\text{verified}| < k$  do
4:   if  $\|(q, n_i)\| + \|(q, p)\| \leq \|(p, n_{far})\|$  then
5:      $H.\text{verified} \cup = n_i$ 
6:   else
7:     if  $|H| < k$  then
8:        $H.\text{unverified} \cup = n_i$ 
9:     else if  $\exists n_j$  and  $\|(q, n_j)\| > \|(q, n_i)\|$  where  $n_j \in H.\text{unverified}$  then
10:      replace  $n_j$  in  $H.\text{unverified}$  with  $n_i$ 
11:     end if
12:   end if
13: end for
14: return  $H$ 

```

for R_v . Therefore, we adopt a polygonization technique that transforms the verified region of each peer ($p.R$) into polygons as a close approximation. After this transformation the polygons can be merged together into a merged verified region R_v by performing the *MapOverlay* algorithm [6] (line 3 of Algorithm 2). The $k\text{NN}_{multiple}$ verification technique is executed with R_v providing an area bound similarly to the $k\text{NN}_{single}$ case. Lemma 3.2 provides the rules for verifying nearest neighbors with multiple peers and the multiple peer NN verification procedure is formalized in Algorithm 2.

Lemma 3.2 *If the nearest neighbor data set P is composed of data from j peers, the merged verified region R_v can be represented as:*

$$R_v = p_1.R \cup p_2.R \cup \dots \cup p_j.R.$$

For any point of interest n_i in R_v , the distance between q and n_i is used as a radius to create a circle C_{n_i} . If C_{n_i} is fully covered by R_v , then n_i is a verified NN of q .

There will be cases when neither $k\text{NN}_{single}$ nor $k\text{NN}_{multiple}$ can fulfill a $k\text{NN}$ query. Hence a set which contains unverified elements is returned. If the response time is critical, a user may agree to accept a $k\text{NN}$ data set with unverified elements, where the objects are not guaranteed to be the top k nearest neighbors. Otherwise the $k\text{NN}$ query must be forwarded to a spatial database server (Step 3). The partial results in H can be used to bound and hence speed up the server search process.

3.2.3 Step 3: Server $k\text{NN}$ Query with Pruning Bounds

We assume that the spatial database server executes an efficient k -nearest neighbor search algorithm based on R-tree indexing [9] for solving $k\text{NN}$ queries. The NN search is supported with a priority queue containing the nodes visited so far. Initially the priority queue contains the entries of

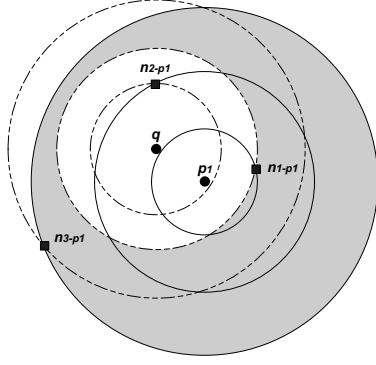


Fig. 3a. Mobile host q retrieves two verified nearest neighbors from the NN set of peer p_1 .

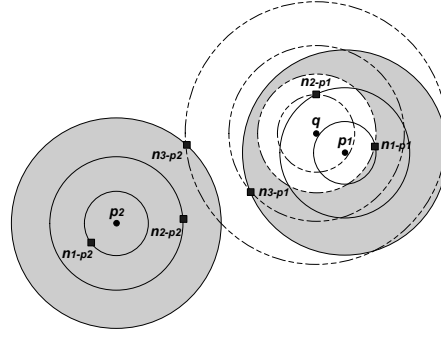


Fig. 3b. Mobile host q can only retrieve unverified nearest neighbors from peer p_2 .

Figure 3. The mobile host q and its two closest peers, p_1 and p_2 .

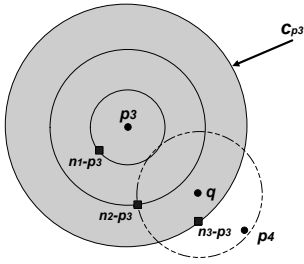


Fig. 4a. Mobile host q cannot verify the point of interest n_{2-p3} as a verified NN with peer p_3 .

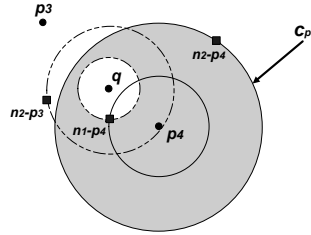


Fig. 4b. Mobile host q cannot verify the point of interest n_{2-p3} as a verified NN with peer p_4 , either.

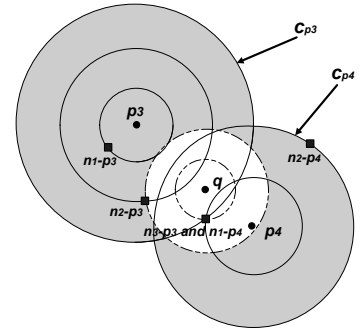


Fig. 4c. After merging the verified regions of peer p_3 and p_4 , the point of interest n_{2-p3} can be verified as a valid object.

Figure 4. An example of the multiple peers NN verification process. The point of interest n_{2-p3} can only be verified as a NN of q based on the merged verified region of both peer p_3 and peer p_4 .

Algorithm 2 $kNN_{multiple}(q, H, P, k)$

```

1:  $R_v \leftarrow \emptyset$ 
2: for  $\forall p \in P$  do
3:    $R_v \cup = p.R$ 
4: end for
5: for  $\forall p \in P$  and  $|H.verified| < k$  do
6:   for  $\forall n_i \in p.N$  and  $n_i \notin H.verified$  do
7:      $C_{ni} \leftarrow$  create a circle region with  $\|q, n_i\|$  as the radius and  $q$  as the center point
8:     if  $c_{ni} \subset R_v$  and  $|H.verified| < k$  then
9:        $H.verified \cup = n_i$ 
10:    end if
11:   end for
12: end for
13: return  $H$ 

```

the R-tree root sorted according to their minimum distance (MINDIST) to the query point q . In general most of the moving objects have executed either one or both kNN_{single}

and $kNN_{multiple}$ processes before forwarding kNN search queries to the server. Hence, it is worthwhile to calculate branch expanding upper and lower bounds from the entries in heap H to speed up the NN search process at the server. The heap H is in one of six different states after a mobile host has executed both the kNN_{single} and $kNN_{multiple}$ mechanisms without retrieving k verified objects:

- State 1: H is full and contains both verified and unverified entries.
- State 2: H is full and contains only unverified entries.
- State 3: H is not full and contains both verified and unverified entries.
- State 4: H is not full and contains only verified entries.
- State 5: H is not full and contains only unverified entries.
- State 6: H contains no entry.

In State 1 there may exist some POIs which are closer to q compared with the last element in H . Hence, we can consider the last entry of H as the final candidate nearest neighbor in the NN search and forward its distance attribute to the server as the branch expanding upper bound. In addition, the distance attribute d_v of the last verified entry can be another bound, the branch expanding lower bound. Because we are certain about the POIs within the circle region C_r with radius d_v and center point q , the NN search algorithm executed in the server does not need to expand any minimum bounding rectangle which is completely covered by C_r . Conversely, when H is full and contains only unverified entries, we can infer only the upper bound (State 2). In States 3 and 4 there have been less than k interest objects found. Therefore, we can only infer the lower bound from the distance attribute of the last verified element in H . In the last two states, H is not full and contains only unverified entries or no entry at all. Consequently we cannot infer any bounds in these cases.

To take advantage of the two new bounds for k NN queries, we slightly modified the k NN best-first search algorithm such that it calculates one more metric for pruning R-tree branches: the maximum distance (MAXDIST). MAXDIST indicates which MBRs are totally covered in region C_r and hence no expansion is needed. Furthermore, we added two new MBR pruning strategies for the k NN search as follows:

1. Any MBR m with $\text{MAXDIST}(q, m)$ smaller than the branch expanding lower bound is pruned.
2. Any MBR m with $\text{MINDIST}(q, m)$ greater than the Euclidean distance from q to the k^{th} element in H is discarded.

The complete procedure of performing the SBNN sharing based k NN search is described in Algorithm 3.

4 Experimental Validation

To ascertain the performance of our approach we have implemented the sharing based spatial query algorithms within a simulation environment. In addition to enabling robust and decentralized applications, the objective of our peer-to-peer design is to increase scalability in two dimensions. First, the server access workload can be reduced as queries are answered directly by peers. Second, for the remaining queries that must be sent to the server, our technique diminishes the search overhead by providing pruning-bounds for the k NN search algorithm. Consequently, the focus of our simulations is on quantifying the server load variations as a function of two main parameters, the *spatial query request rate* (SQRR) and the *page access rate* (PAR). SQRR quantifies what percentage of the client spatial query requests are processed by the central database

Parameter	Description
POI_{Number}	The number of point of interest in the system
MH_{Number}	The number of mobile hosts in the simulation area
C_{Size}	The cache capacity of each mobile host
$M_{Percentage}$	The mobile host movement percentage
$M_{Velocity}$	The mobile host movement velocity (mph)
λ_{Query}	The mean number of queries per minute
Tx_{Range}	The transmission range of queries
λ_{kNN}	The mean number of queried nearest neighbors
$T_{execution}$	The length of a simulation run

Table 3. Parameters for the simulation environment.

server, and PAR denotes the server side memory (primary and secondary) access rate for a sequence of spatial queries. We have performed our experiments with both synthetic and real-world parameter sets.

4.1 Simulator Implementation

Our simulator consists of two main modules, the *mobile host module* and the *server module*. The former generates and controls the movements and query launch patterns of all mobile hosts. Each mobile host (MH) is an independent object which decides its movement autonomously. The server module processes spatial searches and is responsible for estimating the I/O load of the spatial database server. Spatial data indexing is provided with the well known R*-tree algorithm [1]. We implemented our SBNN query algorithm in the mobile host module.

Each MH is implemented as an independent object that encapsulates all its related parameters such as the movement velocity $M_{Velocity}$, the cache capacity C_{Size} , the wireless

Algorithm 3 SBNN(q, k)

- 1: $H \leftarrow \emptyset; P \leftarrow$ peer nodes responding the query request issued from q .
 - 2: **for** $\forall p \in P$ and $|H_{verified}| < k$ **do**
 - 3: $H \cup = k\text{NN}_{single}(q, p, k)$ (**Step 1**)
 - 4: **end for**
 - 5: **if** $|H_{verified}| < k$ **then**
 - 6: $H \cup = k\text{NN}_{multiple}(q, H, P, k)$ (**Step 2**)
 - 7: **end if**
 - {if k verified NN have been retrieved, or the heap is full and q accepts unverified results.}
 - 8: **if** $(|H_{verified}| = k)$ or $(|H| = k$ and $accept = \text{true})$ **then**
 - 9: **return** H
 - 10: **end if**
 - {if H is not full or q denies any unverified results, forward the query with the branch expanding upper and lower bounds to the database server.}
 - 11: $H \leftarrow k\text{NN}$ query results returned from the server. (**Step 3**)
 - 12: **return** H
-

transmission range $TxRange$, etc. All MHs move inside a geographical area, measuring 30 miles by 30 miles. Additionally, user adjustable parameters are provided for the simulation such as execution length, the number of POIs, the number of mobile hosts and their query frequency, and more. Table 3 lists all of the simulation parameters.

The simulation is initialized by randomly choosing a starting location for each mobile host within the simulation area. Mobile hosts follow an underlying road network and their travel speed s is determined by the speed limit on the corresponding road segment. We employ the *random waypoint* model [2] to determine mobility. Each MH selects a random destination point inside the simulation area and progresses towards it. When reaching that location, it pauses for a random interval and decides on a new destination for the next travel period. This process repeats for all MHs until the end of the simulation.

Every simulation has numerous intervals (whose lengths are Poisson distributed) and during each interval the simulator selects a random subset of the mobile hosts to launch spatial queries (the query intervals are also based on a Poisson distribution). The subset size is controlled via the λ_{Query} parameter (e.g., 1,000 queries per minute). These hosts then execute the SBNN algorithm by interacting with their peers. A mobile host will first attempt to answer each spatial query via the kNN_{single} and $kNN_{multiple}$ approaches. If this is unsuccessful, the query will be forwarded to the remote database server. Each mobile host manages its local NN query result cache with a combination of the following two policies:

1. A MH only stores the query location (the coordinates where it launched the query) and all the verified nearest neighbors of the most recent query.
2. If a kNN query must be sent to the server, the MH will query for as many NNs as its cache capacity allows (e.g., if the cache capacity is 20, the query will be for 20-NNs).

The single peer nearest neighbor verification process is implemented according to the algorithm detailed in Section 3.2.1. A mobile host sequentially verifies the candidate points of interest starting with the results obtained from its closest peer query location (using Euclidean distance). In the multiple peer verification algorithm of Section 3.2.2, multiple, potentially overlapping circles must be combined to provide the verification area. We utilize a polygonization technique that transforms all the peer verified region circles into polygons and then sequentially combines them into a merged verified region R_v by performing the *MapOverlay* algorithm. Afterwards, a MH can verify POIs with the $kNN_{multiple}$ verification technique based on the R_v area.

Parameter	Los Angeles County	Riverside County	Synthetic Suburbia	Units
POI_{Number}	4050	2160	3105	
MH_{Number}	121500	11700	66600	
C_{Size}	20	20	20	
$M_{Percentage}$	80	80	80	%
$M_{Velocity}$	30	30	30	mph
λ_{Query}	8100	780	4440	min^{-1}
$TxRange$	200	200	200	m
λ_{kNN}	5	5	5	
$T_{execution}$	5	5	5	hr

Table 4. The simulation parameter sets for Los Angeles County, Riverside County, and Synthetic Suburbia.

4.1.1 Simulation Parameter Sets

To obtain results that closely correspond to real world conditions we obtained our simulation parameters from public data sets, for example, car and gas station densities in urban areas. We term the two parameter sets based on these real-world statistics the *Los Angeles County* and the *Riverside County* parameter sets, respectively. The details of the parameters were as follows.

- **Points of Interest:** We obtained information about the density of interest objects (e.g., gas stations, restaurants, hospitals, etc.) in the Greater Los Angeles area from two online sites: GasPriceWatch.com² and CNN/Money. Because gas stations are commonly the target of kNN queries, we use them as the sample POI type for our simulations. The server load variations of other POI types are expected to be similar.
- **Mobile Hosts:** We collected vehicle statistics of the Greater Los Angeles area from the Federal Statistics web site. The data provide the number of registered vehicles in the Los Angeles and Riverside Counties (5,498,554 and 944,645, respectively). In our simulations we assume that about 10% of these vehicles are on the road during non-peak hours according to the traffic information from Caltrans³. We further obtained the land area of each county to compute the average vehicle density per square mile.

The Los Angeles and the Riverside County parameter sets represent a very dense, urban area and a low-density, more rural area. Hence, for comparison purposes we blended the two real parameter sets to generate a third, synthetic set. The synthetic data set demonstrates vehicle and interest object densities in-between Los Angeles County and Riverside County, representing a suburban area. Table 4 lists the three parameter sets.

²<http://www.gaspricewatch.com>

³<http://www.dot.ca.gov/hq/traffops/saferesr/trafdata/>

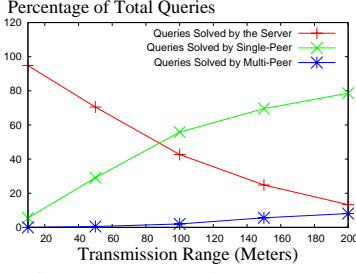


Fig. 5a. Los Angeles County.

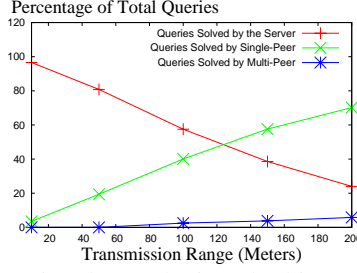


Fig. 5b. Synthetic Suburbia.

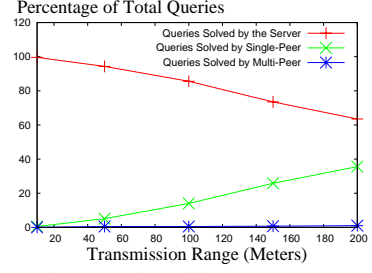


Fig. 5c. Riverside County.

Figure 5. The percentage of queries that are resolved by one peer, multiple peers and the server as a function of the wireless transmission range.

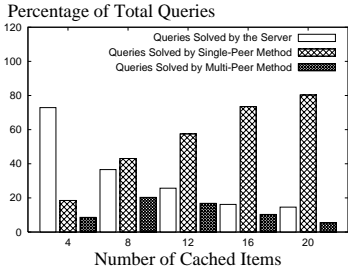


Fig. 6a. Los Angeles County.

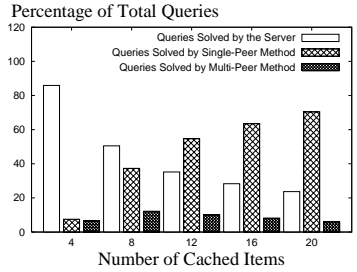


Fig. 6b. Synthetic Suburbia.

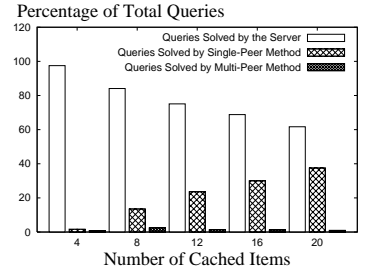


Fig. 6c. Riverside County.

Figure 6. The percentage of queries that are resolved by one peer, multiple peers and the server as a function of the mobile host cache capacity.

4.2 Experimental Results

We used all three input parameter sets – Los Angeles County, Riverside County, and Synthetic Suburbia – to simulate our peer sharing techniques in conjunction with the simulator’s *road network mode*. We varied the following parameters to observe their effects on the system performance: the wireless transmission range, the cache capacity, the movement velocity, and the nearest neighbor number k .

The performance metric in the mobile host module was SQRR. The primary difference between the three different parameter sets is the vehicle and the POI densities. Hence, we utilized the simulation to verify the applicability of our design to different geographical and urban areas. All simulation results were recorded after the system reached a steady state.

4.2.1 Effect of the Transmission Range

In our first experiment we varied the mobile host wireless transmission range from 10 to 200 meters, with all other parameters unchanged. We chose 200 meters as a practical upper limit on the transmission range of the IEEE 802.11 technology. Although the reliable coverage range for IEEE 802.11 in open space with good antennas can be more than 300 meters [7], obstacles such as buildings could diminish the range to 200 meters or less in urban areas. Figure 5 illus-

trates the percentage of the queries that can be resolved by one peer, multiple peers or the server with the Los Angeles County, the Synthetic Suburbia, and the Riverside County parameter sets, respectively. As the transmission range extends, an increasing number of queries can be answered by surrounding peers. As expected, the effect is most pronounced in Los Angeles County, because of its high vehicle density. With a transmission range of 200 meters less than 20% of the queries must be sent to the server.

4.2.2 Effect of the MH Cache Capacity

Next we varied the mobile host cache capacity, which denotes how many nearest neighbor objects a mobile host can store. Figure 6 illustrates cache capacities from 4 to 20 with the three parameter sets. In Figure 6a, even though the number of interest objects is much larger than the maximum capacity of the cached NN query results, we observe a remarkable server workload decrease with a higher MH cache capacity.

4.2.3 Effect of the MH Movement Velocity

We studied the effect of the host movement velocity by changing the MH speed from 10 miles per hour (mph) to 50 mph and the results are shown in Figure 7. We observe that the movement velocity has a stronger effect on the server

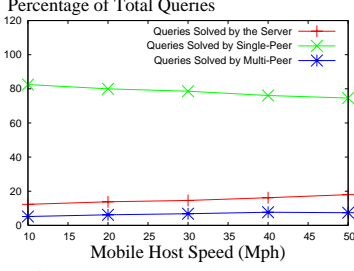


Fig. 7a. Los Angeles County.

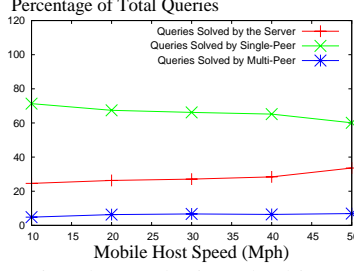


Fig. 7b. Synthetic Suburbia.

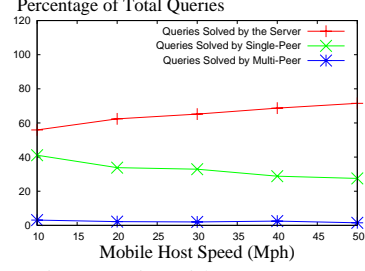


Fig. 7c. Riverside County.

Figure 7. The percentage of queries that are resolved by one peer, multiple peers and the server as a function of the mobile host movement velocity.

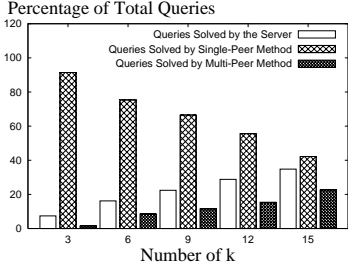


Fig. 8a. Los Angeles County.

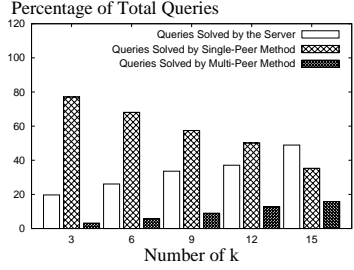


Fig. 8b. Synthetic Suburbia.

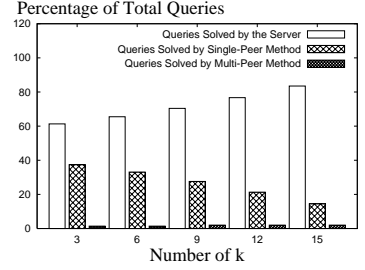


Fig. 8c. Riverside County.

Figure 8. The percentage of queries that are resolved by one peer, multiple peers and the server as a function of k .

workload in areas with a lower vehicle and interest object density. However, the effect is quite gradual in all cases.

4.2.4 Effect of k

We were also interested in the effect that varying the number of requested nearest neighbors, i.e., k , would have on the system performance. In our simulation we chose k randomly for each host and each query in the range from 3 to 15. Figure 8 illustrates the results. The server workload of the Los Angeles County parameter set increases 29% when we raise k from 3 to 15. The server workload of the Riverside County parameter set increases by only 19%, because its starting level is much higher. Not surprisingly result sharing is much more effective for small values of k .

4.3 Impact of Pruning Bounds on Performance of Spatial Database Server

In order to evaluate the nearest neighbor query pruning bounds of Section 3.2.3, we extended the R-tree incremental nearest neighbor (INN) algorithm [9] with one more metric, MAXDIST. For each incoming NN query from mobile hosts, the server module executes both the original INN algorithm and our extended INN algorithm with pruning bounds (denoted EINN) to compare the performance improvement with respect to page accesses. We examined the behavior of the two algorithms as the number of k increased. We utilized the R*-tree for indexing the POI data set (gas

station locations) in the server module. The R*-tree has an advantage in query response time over the conventional R-tree algorithm by utilizing more sophisticated insertion and node-splitting methods, which attempt to minimize a combination of overlap between bounding rectangles and the total area. The branching factor of both the index and leaf nodes was set to 30. Because NN queries are generated by randomly selected mobile hosts, query points are uniformly distributed over the simulation area. The experiments were executed sufficiently often to obtain consistent results.

At the ends of the performance spectrum there are two extreme I/O behaviors of the spatial database server: all requested memory pages are found in main memory or every I/O leads to disk activity. In the former case, because of fast main memory access, we cannot discern a significant performance difference between INN and EINN. However, any reasonably large data set will not fit into main memory and more disk I/Os will be performed. Hence, the database I/O behavior is closer to the other end of the spectrum. Since the EINN usually requests fewer R*-tree nodes and objects than INN, we believe that the k NN search algorithm with query pruning bounds (EINN) will have good scalability with large data sets. During the simulation process, the server module counted the number of R*-tree node (index nodes and data nodes) accesses which corresponded to both main memory and disk I/Os. According to our observation, the number of node accesses provides a good predication of the actual NN query I/O cost.

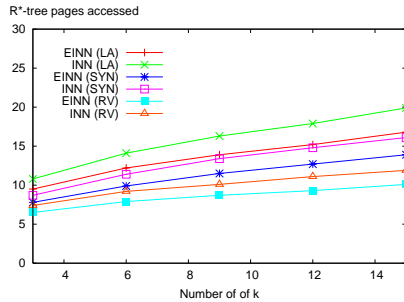


Figure 9. The page access comparison of EINN and INN as a function of k .

Next, we varied the number of k with all the three parameter sets with both the EINN and INN algorithms. The server module recorded the relevant R*-tree page access information (Section 4.2.4). As shown in Figure 9, the EINN algorithm performs consistently better than INN, while the rate of growth is similar for both. We conclude that the pruning bounds can always decrease the number of page accesses. We varied the number of k from 3 to 15 with the three parameter sets and the EINN algorithm accesses 10% to 21% fewer pages than INN.

We conclude from all the performed experiments that the mobile host density has a considerable impact on the *spatial query request rate*. As a result, if more mobile hosts travel in a specific area, each MH has a higher opportunity to fulfill its k NN queries by peers. Furthermore, the nearest neighbor query pruning bounds also have a significant positive effect on the *page access rate* and successfully decrease the server load.

5 Conclusions and Future Work

We have presented a novel approach for answering spatial nearest neighbor search queries by leveraging results from neighboring peers within a mobile environment. Significantly, our method allows a mobile peer to locally verify whether candidate objects received from neighbors are indeed part of its own nearest neighbor data set. Our simulation results indicate that the technique can reduce the access traffic to remote servers by a considerable amount, for example up to 80% in a dense urban area. This is achieved with minimal caching at the peers. By virtue of its peer-to-peer architecture, the method exhibits great scalability: the higher the mobile peer density, the more queries can be answered by peers. Therefore, the load on the remote databases increases sub-linearly with the number of clients. We plan to extend our work to investigate other types of spatial queries, such as range and spatial join searches.

References

- [1] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.
- [2] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the 4th ACM/IEEE MobiCom*, pages 85–97, 1998.
- [3] Chi-Yin Chow, Hong Va Leong, and Alvin Chan. Peer-to-Peer Cooperative Caching in Mobile Environment. In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops*, pages 528–533, 2004.
- [4] Chi-Yin Chow, Hong Va Leong, and Alvin T. S. Chan. Group-based Cooperative Cache Management for Mobile Clients in a Mobile Environment. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, Montreal, Quebec, Canada, August 15-18, 2004.
- [5] M. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the 1st USENIX OSDI*, pages 267–280, November 1994.
- [6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry Algorithms and Applications (2nd Edition)*. Springer, 2000.
- [7] Gregor Gaertner and Vinny Cahill. Understanding link quality in 802.11 mobile ad hoc networks. *IEEE Internet Computing*, 8(1):55–60, 2004.
- [8] Antomn Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, Massachusetts, June 18-21, 1984.
- [9] Gísli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24(2):265–318, 1999.
- [10] Wei-Shinn Ku, Roger Zimmermann, and Chi-Ngai Wan. Location-based Spatial Queries with Data Sharing in Mobile Environment. Technical Report USC-CS-TR05-843, University of Southern California, 2005.
- [11] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest Neighbor Queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 71–79, San Jose, CA, May 22-25, 1995.
- [12] Zhexiong Song and Nick Roussopoulos. k -Nearest Neighbor Search for Moving Query Point. In *Proceedings of Advances in Spatial and Temporal Databases, 7th International Symposium (SSTD)*, pages 79–96, Redondo Beach, CA, July 12-15, 2001.
- [13] D. Wessels and K. Claffy. ICP and the Squid Web Cache. *IEEE Journal on Selected Areas in Communications (JSAC)*, pages 345–357, March 1998.