Efficiently Monitoring Nearest Neighbors to a Moving Object

Cheqing Jin and Weibin Guo

Dept. of Computer Science, East China University of Science and Technololy, China
130 Meilong RD, Shanghai, 200237, China
{cqjin,gweibin}@ecust.edu.cn

Abstract. Continuous monitoring k nearest neighbors in highly dynamic scenarios appears to be a hot topic in database research community. Most previous work focus on devising approaches with a goal to consume litter computation resource and memory resource. Only a few literatures aim at reducing communication overhead, however, still with an assumption that the query object is *static*. This paper constitutes an attempt on continuous monitoring k nearest neighbors to a *dynamic* query object with a goal to reduce communication overhead. In our RFA approach, a Range Filter is installed in each moving object to filter parts of data (e.g. location). Furthermore, RFA approach is capable of answering three kinds of queries, including precise kNN query, non-value-based approximate kNN query, and value-based approximate kNN query. Extensive experimental results show that our new approach achieves significant saving in communication overhead.

1 Introduction

Finding k nearest neighbors (kNN) to a query object is one of the most critical operations in the field of spatial databases. The primary focus of spatial database research till recently has been on static spatial data which are updated infrequently, such as buildings, roads, .etc[7,13,14]. Nowadays, there has been an increasing interest in processing objects in motion, which change location frequently, such as vehicles, mobile networks, .etc. In a typical scenario, each moving object continues to report its location to a special site frequently, where the answer is calculated and output in real-time. Most previous work focus on devising various solutions to generate qualified results with few memory resource and computation resource (e.g., [8,9,11]).

The network communication resource is a critical resource in many real-world scenarios, especially distributed environments. For example, if the amount of objects becomes larger and larger, the network resource appears to be the bottle-neck for the processing system. Consequently, it is necessary to devise communication-efficient solutions for such scenarios. Unfortunately, to our best knowledge, only a few literatures[1,2,10] take this factor into account and give out solutions. Babcock et al. propose a method to monitor k objects with largest numeric value over distributed environments, which can be viewed as a solution for finding kNN objects in 1-dimensional space with a special query object (∞)[1]. Recently, Cheng et al. present a solution to find k nearest neighbors with non-value tolerance over distributed environments[2]. Mouratidis et al. also propose one threshold-based approach to monitor k-nearest neighbors[10].

One common weakness of the above methods is that they mainly focus on handling *static* query object, i.e, the value of the query object being unaltered with time going on (e.g., the query object is ∞ in [1] and a constant in [2]). An example query can be described like: *what are k nearest taxies to a school?* However, there still exist some situations requiring *dynamic* query object, i.e, an object in motion. For example, in the query like: *what are k nearest clients to a free taxi?*, the *free taxi* is a moving object. Previous methods (e.g. [1,2]) cannot be easily adapted to solve such problem, it is necessary to seek new solutions. A simple way to handle *dynamic* query object is: some objects (more than k) are forced to report current location whenever the query object moves[10].

This paper focuses on devising novel approach on continuous monitoring k nearest neighbors to a *dynamic* query object with a goal to reduce the communication overhead. We assume an environment containing n moving objects sending their locations to a central site frequently. The central site continues to process the query without any knowledge about the velocities and the trajectories of objects.

The main contribution is that we have proposed a novel approach, the Range-Filter-based Approach (RFA), to cope with the problem. We sketch the approach as follows. Initially, each object is affiliated with a range filter whose purpose is to transmit new location to the central site if the location exceeds a specified range. The filter is initialized by the central site and maintained by the cooperation of the central site and the object itself. Simultaneously, the central site also reserves a copy of all filters. During the running time, when an object moves, it detects new location v, compares v with the range, and sends v to the central site once v is out of the range. The central site also updates filter settings of some objects if necessary. At any time, the central site can output the answer based on the a copy of filter settings in the central site.

The most significant characteristic of a range filter, the core structure in the RFA approach, is the self-adaptivity. In previous filter-based approaches, (e.g., [1,2,12]), when remote filters are outdated, the central site calculates the new setting, and sends them to remote objects. It may result in great network transmission overhead if such events frequently happen. But in RFA approach, when encountering such situation, the central site and the remote object are capable of calculating a same filter setting simultaneously. If the new setting satisfies the querying condition, no additional transmission occurs.

Some scenarios prefer approximate answer because it can be calculated easily even though the rate of stream is rapid and the volume of data is huge. Besides providing precise answer, RFA approach is capable of processing two kinds of approximate queries, including value-based approximate kNN query and non-value-based approximate kNN query[2]. The value-based approximate kNN query introduces a numeric value to guarantee the answer, whereas the non-value-based approximate kNN query expresses the error tolerance in terms of a *rank*.

The rest of the paper is organized as follows. Section 2 describes the environment in brief and defines the query formally. Section 3 depicts Range Filter structure and RFA approach in detail. Section 4 evaluates the performance of the new approach by a series of experiments. Section 5 reviews related work of this paper. Finally, Section 6 concludes the paper with a summary and directions for future work.

2 Preliminaries

2.1 Environment

We consider an environment containing n moving objects and 1 central site. When an object (say, O_i) moves, it sends its identity and new location to the central site through wireless network. Let S denote a set containing all objects, $S = \{O_1, O_2, \cdots, O_n\}$; let $V_{i,t}$ denote the location of the object O_i at time t. Each object O_i generates a stream of trace: $\{V_{i,0}, V_{i,1}, \cdots,\}$. Although an object can send(/receive) data to(/from) the central site, no direct communication routine exists between any pair of objects. Based on the data received from objects, the central site is capable of answering a kNN query in real-time, as demonstrated in Figure 1.

The location of each object is in a d-dimensional metric space. The distance between two arbitrary locations can be described by a distance function *dist* with following properties, where V_1, V_2 and V_3 are locations of three objects.

1. $dist(V_1, V_2) = dist(V_2, V_1)$
2. $dist(V_1, V_2) > 0 (V_1 \neq V_2)$ and $dist(V_1, V_2) = 0 (V_1 = V_2)$
3. $dist(V_1, V_2) \leq dist(V_1, V_3) + dist(V_2, V_3)$

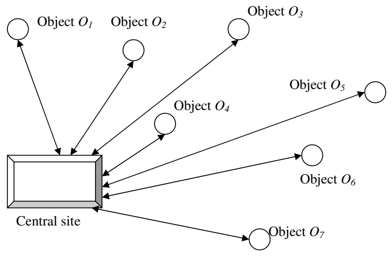


Fig. 1. Environment

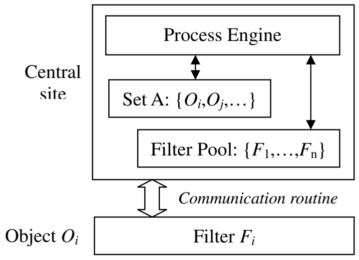


Fig. 2. Architecture

Table 1. An example of source data and query results

Time	O_1	O_2	O_3	O_4	O_5	$kNN(O_1,2)$	$nvakNN(O_1,2,1)$	$vakNN(O_1,2,1.5)$
0	1	3	7	10	12	$\{O_2,O_3\}$	$\{O_2,O_3,O_4\}$	$\{O_2,O_3\}$
1	4	5	10	14	9	$\{O_2,O_5\}$	$\{O_2,O_3,O_5\}$	$\{O_2,O_3,O_5\}$
2	3	4	15	14	9	$\{O_2,O_5\}$	$\{O_2,O_4,O_5\}$	$\{O_2,O_5\}$
3	1	3	11	12	8	$\{O_2,O_5\}$	$\{O_2,O_3,O_5\}$	$\{O_2,O_5\}$
4	2	8	15	16	7	$\{O_2,O_5\}$	$\{O_2,O_3,O_5\}$	$\{O_2,O_5\}$

2.2 Query Definition

This paper considers three kinds of kNN queries, including precise kNN query, non-value-based approximate kNN query and value-based approximate kNN query. The query object q can be either *static* (e.g, school, landmark), or *dynamic* (e.g, vehicle). Let q_t denote the location of q at time t.

Precise kNN query ($kNN(q,k)$): q is a query object, $q \in S$; k is the number of neighbors, $k \in N^+$. At any time point t, return a set A satisfying following conditions: (1) $q \notin A$; (2) $\forall O_i \in A, O_j \in S - A - \{q\}$, we have: $dist(V_{i,t}, q_t) \leq dist(V_{j,t}, q_t)$; (3) $|A| = k$.

Non-value-based approximate kNN query ($nvakNN(q,k,r)$): q is a query object; k is the number of neighbors, $k \in N^+$; r is an error parameter, $r \in N^+$. At any time point t, return a set A satisfying following conditions: (1) $A \subseteq kNN(q, k+r)$; (2) $|A| = k$.

Value-based approximate kNN query ($vakNN(q,k,e)$): q is a query object; k is the number of neighbors, $k \in N^+$; e is an error parameter, $e \in R^+$. At any time point t, return a set A satisfying following conditions: (1) $q \notin A$; (2) $\forall O_i \in A$, $dist(V_{i,t}, q_t) \leq e + \max_{O_j \in kNN(q,k)}(dist(V_{j,t}, q_t))$; (3) $|A| = k$.

Example 1. *Table 1 demonstrates a small example on processing above queries. The locations (in 1-dimensional space) of 5 objects in first 5 time points are illustrated in the left 6 columns. The seventh column shows the answer for* $kNN(O_1,2)$*. The right 2 columns show the maximum result set for* $nvakNN(O_1,2,1)$ *and* $vakNN(O_1,2,1.5)$ *respectively, which implies that any subset containing two objects is a legal answer. For instance, at time point 4, any subset of* $\{O_2,O_3,O_5\}$ *(i.e.,* $\{O_2,O_3\}$, $\{O_2,O_5\}$, $\{O_3,O_5\}$*) is legal for the query* $nvakNN(O_1,2,1)$*.*

3 Range Filter-Based Approach (RFA)

This section describes our novel Range-Filter-based Approach (RFA) in detail. Figure 2 illustrates the architecture. A *range filter* (say, F_i) is installed in one object (say, O_i) to reduce communication overhead by filtering parts of new locations within a range. The central site consists of three components, such as a *range filter*, an answer set A and *process engine*. The *filter pool* reserves a copy

Algorithm 1. *newRange(F, v)* /* $F = (c, l, u, b)$ */

```
1: $\Delta = dist(c, v) - b$;
2: if $(\Delta > 0)$ then
3:     $l = b + \frac{\Delta}{2}$;
4:     $u = b + \frac{3\Delta}{2}$;
5: else
6:     $l = \max(0, b + \frac{3\Delta}{2})$;
7:     $u = b + \frac{\Delta}{2}$;
```

of all filters of moving objects. At any time point, the *process engine* is capable of calculating an *answer set A* based on the data in *filter pool*. Another task of the *process engine* is to reset filters in remote objects if current settings cannot satisfy the query requirement.

Section 3.1 describes the structure of *Range Filter*. Section 3.2 describes a way to calculate answers from filters. Finally, Section 3.3 introduces the overall algorithm.

3.1 Range Filter

A Range Filter F is defined as (c, l, u, b). Field c is a location value in d-dimensional space, representing the central point of the filter's range. Fields l and u are the *lower diameter* and *upper diameter* respectively, representing the minimum and maximum distance from the location c, $0 \leq l \leq u$. The range of a filter F is the collection of locations whose distances to c are within $[l, u]$. In other words, a new location v is claimed in the range of filter F only if $dist(v, c) \in [l, u]$. In RFA approach, an object sends current location to the central site only when (1) its location exceeds range, (2) receives a request from the central site.

One important characteristic of a range filter is that it can generate a new range when current location v exceeds the range, as shown in Algorithm 1. With the help of field b (*backup diameter*). Algorithm `newRange` (Algorithm 1) alters the lower diameter l and upper diameter u, but retains the central location c unchanged. Clearly, after processing, we still have: $l \leq dist(c, v) \leq u$. Note that the value of b is only determined by the central site (see Algorithm 3).

Figure 3 demonstrates how to reset the range by invoking Algorithm `newRange`. When the new value v (the shadowed circle) jumps out of the range (covered by the dotted curve in the left part of Figure 3), Algorithm `newRange` calculates new lower and upper diameter (l', u'), as shown in the right part of Figure 3. Figure 3(a) shows situations when $dist(c, v) < b$, while Figure 3(b) shows situations when $dist(c, v) > b$.

3.2 Finding Nearest Neighbors from *Filter Pool*

One task of the *process engine* is to seek the nearest neighbors based on a copy of objects' filters reserved in *filter pool*. The first step is to determine the minimum

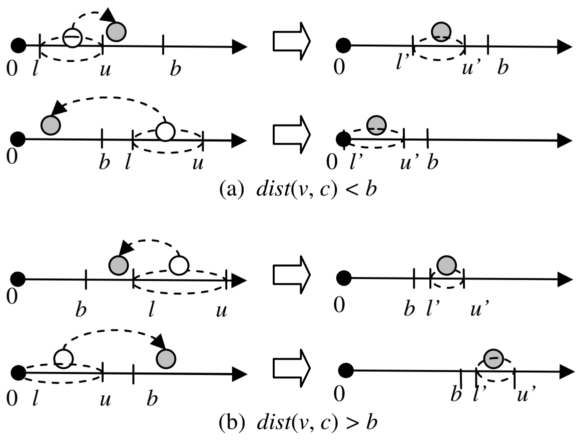


Fig. 3. Resetting the filter by invoking Algorithm `newRange`

and maximum distance between $V_{i,t}$ and q_t (q_t is the current value of query object q). Remember that $dist(V_{i,t}, c_i) \in [l_i, u_i]$. Consequently, $dist(V_{i,t}, q_t)$ is influenced by two factors, including the filter setting and the value of q_t. According to the definition of subroutine *dist*, the minimum and the maximum possible values of $dist(V_{i,t}, q_t)$, denoted as L_i and U_i, are calculated as follows.

$$L_i = \begin{cases} l_i - dist(q_t, c_i) & : \quad dist(q_t, c_i) < l_i \\ 0 & : \quad (dist(q_t, c_i) < u_i) \wedge (dist(q_t, c_i) > l_i) \\ dist(q_t, c_i) - u_i & : \quad dist(q_t, c_i) > u_i \end{cases} \tag{1}$$

$$U_i = dist(q_t, c_i) + u_i \tag{2}$$

Let R_i denote the maximum possible number of objects nearer to q_t than O_i. The value of R_i is calculated when assuming O_i is at the furthest possible point to q_t, and other objects are at the nearest possible points to q_t, as shown in Equ. (3).

$$R_i = |X|, \quad where \quad X = \{O_j | O_j \in (S - \{O_i, q\}), L_j \leq U_i\} \tag{3}$$

Lemmas (1)-(3) show how to answer three kinds of queries from the *filter pool*.

Lemma 1. *Let $A = \{O_i | R_i < k\}$. If $|A| = k$, then A is the answer for Precise kNN query $kNN(q, k)$.*

The correctness of Lemma 1 comes from the definition of R_i (Equ. (3)).

Lemma 2. *Let $A = \{O_i | R_i < k + r\}$. If $|A| \geq k$, then any subset of A containing k objects is an answer for non-value-based approximate kNN query $nvakNN(q, k, r)$.*

The correctness of Lemma 2 also comes from the definition of R_i (Equ. (3)).

Algorithm 2. `RFA_client(i)`

```
loop
  if Sees a new location $V_{i,t}$ then
    if Object $O_i$ itself is the query object $q$ then
      Sends $V_{i,t}$ to central site
    else if $dist(V_{i,t}, c_i) \notin [l_i, u_i]$ then
      newRange($F_i, V_{i,t}$);
      Sends $V_{i,t}$ to central site;
  else if Receives $(c'_i, l'_i, u'_i, b'_i)$ from central site then
    $(c_i, l_i, u_i, b_i) = (c'_i, l'_i, u'_i, b'_i)$;
  else if Receives a request SEND from central site then
    Sends $V_{i,t}$ to central site;
```

Lemma 3. *Let $A = \{O_i | R_i < k\} \cup \{O_i | U_i \leq \hat{L} + e, R_i \geq k\}$, where $\hat{L}$ is the k^{th} smallest value of L_i. If $|A| \geq k$, any subset of A containing k objects is an answer for a value-based approximate kNN query $vakNN(q, k, e)$.*

We sketch the proof here. The set A consists of two parts. The first part is $\{O_i | R_i < k\}$. According to the definition of R_i, all objects belong to k nearest neighbors. The second part is $\{O_i | U_i \leq \hat{L} + e, R_i \geq k\}$. Because $\hat{L}$ is the minimum possible distance between the k^{th} nearest neighbor and the query object, and U_i is the maximum possible distance between the object O_i and q_t, any object in the second part also meets the requirement according to the definition of value-based approximate kNN query.

3.3 Algorithm Description

This section introduces RFA approach in detail. RFA approach consists of two parts: (1) Algorithm `RFA_client` (Algorithm 2), running in moving objects; and (2) Algorithm `RFA_server` (Algorithm 3), running in the central site.

Running in each moving object, the goal of Algorithm `RFA_client` is to handle new location value and to communicate with the central site. When object O_i '*sees*' a new location $V_{i,t}$, and the object O_i is just the query object q, O_i sends $V_{i,t}$ to the central site immediately. Otherwise, O_i begins to check whether $V_{i,t}$ stays in the range or not. Only when O_i exceeds the range, subroutine `newRange` is invoked to update filter F_i and send $V_{i,t}$ to the central site (lines 2-7). Each remote object may receive two kinds of messages from the central site. The first kind is new filter setting (c'_i, l'_i, u'_i, b'_i), and the second kind is a SEND request. When receiving a first kind message, object O_i updates the local filter F_i accordingly. When receiving a second kind message, O_i sends $V_{i,t}$ to the central site immediately(lines 8-11).

Algorithm `RFA_server` contains two phases, *initialization* phase and *maintaining* phase. In *initialization* phase (lines 1-7), the central site receives all locations of objects, based on which it creates set A containing k objects nearest to q_0. And then, it calculates filter settings to update all objects. The field c_i of all filters is set to q_0. The field b_i is set to the average value of the maximum distance

Algorithm 3. `RFA_server()`

```
1: Receives all locations $\{V_{1,0}, V_{2,0}, \cdots V_{n,0}\}$ from moving objects;
2: Creates set $A$, containing $k$ objects with minimum $dist(V_{i,0}, q_0)$;
3: $V' = \min_{O_i \in S-A-\{q\}}(dist(V_{i,0}, q_0))$;   $V'' = \max_{O_i \in A}(dist(V_{i,0}, q_0))$;
4: $B = \frac{V'+V''}{2}$;
5: foreach filter $F_i$
6:    $c_i = q_0$;  $b_i = B$;  newRange($F_i, V_{i,0}$);
7:    Sends $F_i$ to object $O_i$;
8: loop
9:    if Receives $V_{i,t}$ from object $O_i$ then
10:       newRange($F_i, V_{i,t}$);
11:       if Can't find answer according to Lemma (1)-(3) then
12:          adjust();
13:       Output result;
```

Algorithm 4. `adjust()`

```
1: Calculates a set $A$ satisfying $A = \{O_i | R_i < k\}$;
2: while ($|A| < k$)
3:    Finds $O_i$ with min $L_i$ in $S - A - \{q\}$;
4:    if $O_i$ has not send $V_{i,t}$ to the central site then
5:       receives $V_{i,t}$ from $O_i$ by sending SEND signal to $O_i$;
6:       $L_i = dist(V_{i,t}, q_t)$;   $U_i = dist(V_{i,t}, q_t)$;
7:    else
8:       $A = A + \{O_i\}$;
9: $B = \frac{L'+U'}{2}$, where $L' = \max(U_i | O_i \in A), U' = \min(L_i | O_i \in S - A - \{q\})$;
10: forall objects with $L_i = U_i$
11:    $c_i = q_t$;   $b_i = B$;   newRange($F_i, V_{i,t}$);
12:    Sends $(c_i, l_i, u_i, b_i)$ to object $O_i$;
```

in A and the minimum distance in $S - A - \{q\}$, so that Lemmas (1)-(3) can be satisfied after invoking Algorithm `newRange`. In *maintaining* phase (lines 8-13), the central site begins to process query when a new location $V_{i,t}$ arrives. First, it invokes Algorithm `newRange` to update the filter setting in *filter pool* (lines 9-10). Second, it continues to check whether Lemmas (1)-(3) are satisfied. Once these lemmas cannot be satisfied, it would invoke Algorithm `adjust`(Algorithm 4) to update filter settings, and output new data.

The goal of Algorithm `adjust` (Algorithm 4) is to find a set of k objects nearest to the query object q and update filters accordingly. In fact, the initialization phase of Algorithm 3 has implied a simple method to cope with it. However, it requires all objects sending their locations to the central site, which results in heavy network transmission burden. Algorithm 4 illustrates a more efficient way. First, it calculates a set $A = \{O_i | R_i < k\}$. Clearly, if $|A| = k$, set A contains all k nearest neighbors. Otherwise, we should continue to add a neighbor object into A to make $|A| = k$ by iterations. For every an iteration, we check an object with smallest L_i (because this object is a candidate) (lines 1-8). Second, it continues to reset filter settings for parts of objects just sending new locations to the central site. Similar to the initialization phase in Algorithm 3, the local variable B is calculated as the average value of the maximum possible value in A and minimum possible value in $S - A - \{q\}$. Finally, it updates filter settings for objects with $L_i = U_i$ by invoking Algorithm `newRange`, and sends new settings to corresponding objects (lines 10-12).

Analysis: RFA approach is capable of answering a kNN query at any time point. Initially, Algorithm `RFA_server` creates filters for all objects. During the maintaining phase, if no new location is transmitted from remote objects, we can always answer the query because all current locations are within the range. Otherwise, if the central site receives a new data from any object, it invokes `newRange` to create new setting, and checks the validation by Lemmas (1)-(3). Algorithm `adjust` is then invoked to generate new settings satisfying Lemma 1 on condition that the above exam fails.

4 Experiments

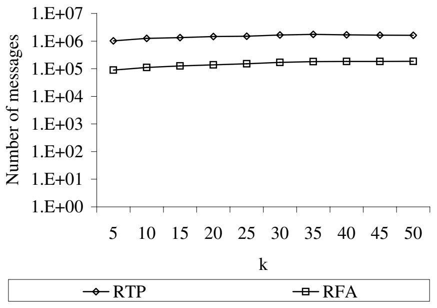
This section begins to evaluate the performance of RFA approach through a series of experiments. Section 4.1 compares the performance between RFA approach and RTP approach[2]. Section 4.2 continues to analyze the network communication overhead in RFA approach. Finally Section 4.3 reports the performance of RFA approach upon different error tolerances.

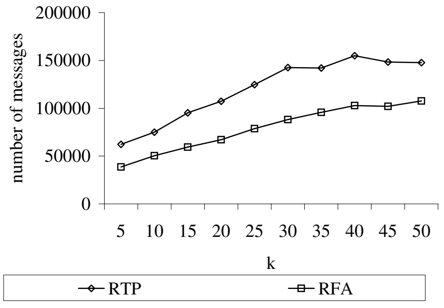
All experiments are based on a dataset containing the location traces (in 2-dimensional space) of hundreds of vehicles. Each vehicle is initialized with (1) a location (x_0, y_0) random selected from [-1000, 1000], (2) a velocity s, $s \in [5, 15]$ and (3) a moving direction α, $\alpha \in [-\pi, \pi]$. At any time point t, the new location (x_t, y_t) is calculated as:$(x_t, y_t) = (x_{t-1} + s \cdot \cos(\alpha), y_{t-1} + s \cdot \sin(\alpha))$. The velocity and the direction are changed randomly for every a minute. Totally, 100 vehicles will generate 100*(60*60*24)=8,640,000 locations.

4.1 Handling a Static Query Object

The RTP approach is a filter-based solution to answer kNN queries over distributed environments[2]. However, this method only focuses on handling static query object, such as school, landmark, and so on.

Figure 4 compares the performance between the RTP approach and RFA approach. The query point is fixed at $(0, 0)$. Figures 4(a) and (b) report the number of messages transfered via network when running $kNN((0, 0), k)$ and $nvakNN((0, 0), k, 10)$ respectively. The x-axis represents the number of neighbors k, and the y-axis represents the number of messages raised. In all situations, RFA approach outperforms RTP approach significantly. The main reason is that all filters share same width in RTP approach, so that nearly all objects are forced to send their new locations to the central site once $|A| > k$. But in RFA





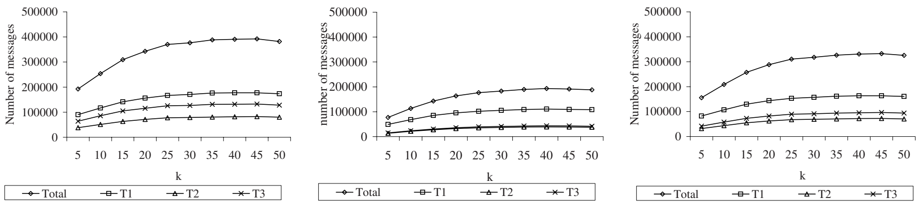
(a) $kNN((0,0),k)$ (b) $nvakNN((0,0),k,10)$

Fig. 4. Comparison between RTP approach and RFA approach

approach, each filter owns a different range, so that when $|A| < k$ or $|A| > k$ occurs, only a small fraction of objects send new locations to the central site.

4.2 Analysis on Communication Cost

RFA approach involves three kinds of network transmission costs, including (1) T_1: moving objects transmit new location to the central site when it moves out of the range; (2) T_2: the central site sends a SEND message to a moving object for new location; (3) T_3: the central site updates filter settings of moving objects. Figure 5 demonstrates the number of messages transmitted via network when running three different queries, such as $kNN(q,k)$, $nvakNN(q,k,10)$ and $vakNN(q,k,100)$. The number of neighbors changes from 5 to 50. When k increases, all kinds of costs increase. In all situations, T_2 and T_3 are smaller than T_1. We can also observe that the amount of messages transfered is still only a small fraction of total messages $(8,640,000)$.

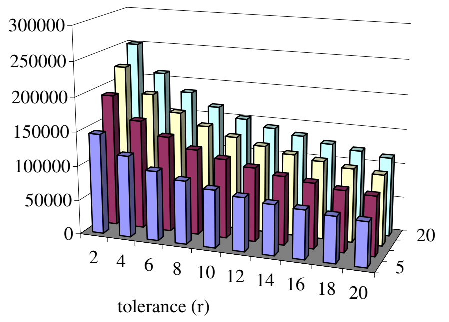


(a) $kNN(q,k)$ (b) $nvakNN(q,k,10)$ (c) $vakNN(q,k,100)$

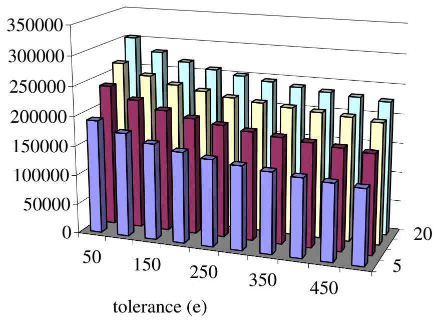
Fig. 5. All kinds of communication costs

4.3 The Impact of Tolerance

RFA approach can support two kinds of approximate kNN queries, including non-value-based approximate kNN query and value-based approximate kNN query. Here, we implement experiments to evaluate the communication cost under different tolerances. Figure 6(a)and (b) examine queries $nvakNN(q, k, r)$ and $vakNN(q, k, e)$ respectively. The x-axis represents the error tolerance (e.g. r in Figure 6(a) and e in Figure 6(b)); the y-axis represents the number of neighbors; the z-axis represents the total number of messages transfered. In all situations, the communication cost is reduced when given larger tolerance.



(a) $nvakNN(q, k, r)$



(b) $vakNN(q, k, e)$

Fig. 6. The performance under different error tolerance

5 Related Work

Finding k nearest neighbors to a query object has been widely studied for a long period. Traditional approaches mainly focus on creating and maintaining various indexes over static objects for optimization with a premise that all data have resided in the disks, and can be accessed multiple times[7,13,14]. Nowadays, there has been an increasing interest in continuous monitoring objects in motion[8,9,10,11,15]. Iwerks et al. invented CW approach to monitor PKO objects (point kinematic object)[8]. Koudas et al. proposed a system for approximate kNN queries over streams of multi-dimensional points[9]. Yu et al. gave out grid-base algorithms to index objects and queries to reduce processing cost[15]. Mouratidis et al. pioneered the work on continuous monitoring objects in road networks[11]. The common goal of such work is to provide qualified results with small memory resource and computation resource.

One of the critical goals for applications over distributed environments is to minimize the usage of network communication cost. Example algorithms consist of cardinality of set-expressions monitoring[6], quantile monitoring[4], general-purpose approximate query monitoring[3], distinct count estimate and distinct sample estimate[5], .etc. However, only a few literatures on monitoring k nearest neighbors consider network communication factors[1,2,10]. Babcock et al. give a solution for top-k monitoring, which can be treated as a special case of kNN monitoring. Cheng et al. propose solution for kNN monitoring with non-value tolerance[2]. These work only consider a static query object, while the work in [10] gives a simple method to handle dynamic query object, which is the target of this paper.

6 Conclusions

This paper constitutes an attempt in finding k nearest neighbors to a *dynamic* query object with a goal to reduce the communication overhead, which is critical for distributed monitoring applications. The main contribution is a novel approach, the Range-Filter-based Approach(RFA). Each moving object is installed with a range filter, so that only a small part of data is transmitted via network. At any time point, the central site can find k nearest neighbors to the query object by merely checking the information in its filter pool. Our approach supports three kinds of queries: precise kNN query, non-value-based approximate kNN query, and value-based approximate kNN query. Experimental results show that our approach only consumes small amount of network transmission. One challenging research direction is the kNN monitoring in road network, which is more common in real-life.

References

1. Babcock, B., Olston, C.: Distributed top-k monitoring. In: Proc. of SIGMOD (2003)
2. Cheng, R., Kao, B., Prabhakar, S., Kwan, A., Tu, Y.: Adaptive stream filters for entity-based queries with non-value tolerance. In: Proc. of VLDB (2005)
3. Cormode, G., Garofalakis, M.: Sketching streams through the net: Distributed approximate query tracking. In: Proc. of VLDB (2005)
4. Cormode, G., Garofalakis, M., Muthukrishnan, S., Rastogi, R.: Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In: Proc. of ACM SIGMOD (2005)
5. Cormode, G., Muthukrishnan, S., Zhang, W.: What's different: Distributed, continuous monitoring of duplicate-resilient aggregates on data streams. In: Proc. of ICDE (2006)
6. Das, A., Ganguly, S., Garofalakis, M., Rastogi, R.: Distributed set-expression cardinality estimation. In: Proc. of VLDB (2004)
7. Hjaltason, G., Samet, H.: Ranking in spatial databases. In: Egenhofer, M.J., Herring, J.R. (eds.) SSD 1995. LNCS, vol. 951, Springer, Heidelberg (1995)
8. Iwerks, G., Samet, H., Smith, K.: Continuous k-nearest neighbor queries for continuously moving points with updates. In: Proc. of VLDB (2003)
9. Koudas, N., Ooi, B., Tan, K., Zhang, R.: Approximate nn queries on streams with guaranteed error/performance bounds. In: Proc. of VLDB (2004)
10. Mouratidis, K., Papadias, D., Bakiras, S., Tao, Y.: A threshold-based algorithm for continuous monitoring of k nearest neighbors. IEEE Transactions on Knowledge and Data Engineering, 17(11) (November 2005)
11. Mouratidis, K., Yiu, M.L., Papadias, D., Mamoulis, N.: Continuous nearest neighbor monitoring in road networks. In: Proc. of VLDB (2006)

12. Olston, C., Jiang, J., Widom, J.: Adaptive filters for continuous queries over distributed data streams. In: Proc. of SIGMOD (2003)
13. Weber, R., Schek, H.-J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proc. of VLDB (1998)
14. Yu, C., Ooi, B., Tan, K.-L., Jagadish, H.V.: Indexing the distance: An efficient method to knn processing. In: Proc. of VLDB (2001)
15. Yu, X., Pu, K.Q., Koudas, N.: Monitoring k-nearest neighbor queries over moving objects. In: Proc. of ICDE (2005)