A Study of Location Monitoring in Mobile System

Bhakte D.B*, Prof. Vharkate M.N**

* PG Student, College of Engineering, Osmanabad
** ME (CSE), College of Engineering, Osmanabad

Abstract- This paper also proposes a generic framework for monitoring continuous spatial queries over moving objects. The framework distinguishes itself from existing work by being the first to address the location update issue and to provide a common interface for monitoring mixed types of queries. We propose algorithms for query evaluation/reevaluation and for safe region computation in this framework. In this paper we address two issues first issue We show that existing approaches may fail to provide spatial anonymity for some distributions of user locations and describe a novel technique which solves this problem and second issue We propose Prive, a decentralized architecture for preserving the anonymity of users issuing spatial queries to LBS. Mobile users self organize into an overlay network with good fault tolerance and load balancing properties. The most important concept of PAM, it is used to the frequently encountered type of query in geographic information system is to find the k nearest neighbor objects to given point in space. In a mobile service scenario, users query a server for nearby points of interest but they may not want to disclose their locations to the service.

Index Terms- Spatial Relations, Database server, Query index, Object index, KNN query

I. INTRODUCTION

With the advent of mobile and ubiquitous computing, monitoring continuous spatial queries over moving objects has become a necessity for various daily applications, [4] such as fleet management, cargo tracking, child care, and location aware advertisement. The fundamental problem in a monitoring system is when and how a mobile client should send location updates to the server because it determines three principal performance measures of monitoring—accuracy, efficiency, and privacy. In the literature, very few studies on continuous query monitoring are focused on location updates. Two commonly used updating approaches are periodic update (every client reports its new location at a fixed interval) and deviation update (a client performs an update when its location or velocity changes significantly) .The application servers register spatial queries of interest at the database server, which then continuously updates the query results until the queries are The application servers register spatial queries of interest at the database server, which then continuously updates the query results until the queries are deregistered. There are two predominant costs that determine the system performance: the wireless communication cost for location updates and the query evaluation cost at the database server.

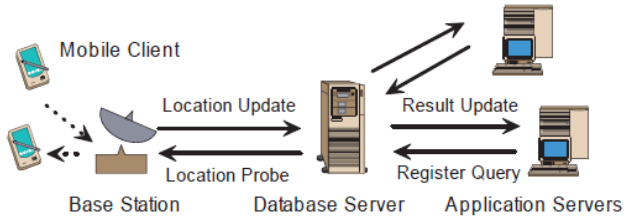

Fig1: The System Architecture

In this paper, we propose an innovative and generic monitoring framework to overcome these problems by taking a systematic approach. Figure2. Shows a simple example where there are two moving objects a and b and two registered queries Q1 (an NN query) and Q2 (a range query). The current results of these two queries are {a} and {a, b} respectively, which will not change if a and b are in their safe regions Sa and Sb (the shaded boxes). As an example in Figure 1.2, when a moves out of Sa to a new location a′, the result of Q1 becomes undecided as either of the two objects could be the nearest neighbor. To resolve the ambiguity, the server has to probe some objects (in this example, b) to request an immediate location update (this is called a server-initiated probe and update).

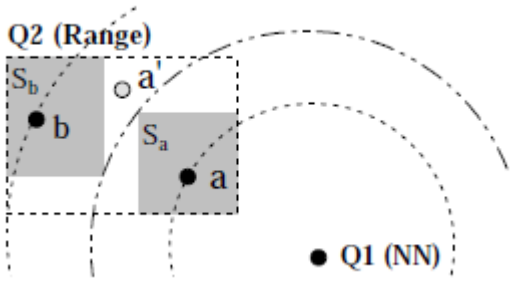

Fig 2: Example of Safe Regions

II. THE FRAMEWORK OVERVIEW

In this case, In this section, we first make some assumptions on the system model, and then introduce the framework by describing the structure and behaviors of the database server. The next two sections will show the detailed query evaluation/reevaluation and safe region computation algorithms at the database server. To simplify the system model, we make the following assumptions:

At the database server, all registered queries can be fit into main memory whereas not all the moving objects can. This is a common and fair assumption in monitoring continuous spatial

queries [2]. The database server handles location updates sequentially. In other words, no location updates take place during the processing of a new query or another location update. Although this is not a prerequisite for this framework,
it is a reasonable assumption to relieve us from considering the read/write consistency.

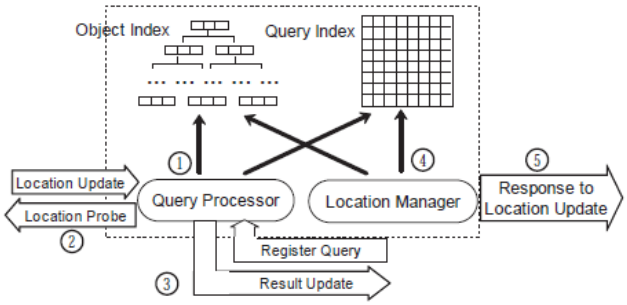

Fig 3: The Database Server Structure

In fact, in reality communication incurs a measurable update propagation delay and thus the exact client location and the location maintained at the database server are not always synchronized. The violation of this assumption affects the monitoring accuracy, which will be measured in the experiments.

The communication cost between every client and the database server is constant. Throughout this paper, C_l denotes the cost for one source-initiated location update and C_p denotes the cost for one server-initiated location probe and update.

Mobile clients are able to detect their locations, through positioning technologies such as GPS.

2.1 The Database Server

As depicted in Figure 3, the database server has four components: the on-disk index for the moving objects, the in memory index for the queries, the location manager, and the query processor. Upon receiving a location update, the query processor first reevaluates those queries affected by this update based on the indexes. During the reevaluation, the query processor might need to probe some objects for server-initiated location updates to determine the query results. The updated query results are then reported to the application servers which register these queries. Afterwards, the location manager computes the new safe regions for this object and the probed objects, also based on the two indexes. Finally, these new safe regions are sent back to the corresponding clients as the responses for their location updates. The server's behaviors upon a new query being registered are similar: it goes through, except that in, the query is evaluated from scratch instead of being reevaluated incrementally.

2.2 The Object Index

The object index stores the current safe regions of all the objects. While many spatial index structures can serve this purpose, this paper employs the well-known R-tree based index [2]. Since the safe region changes each time the object updates its location (either client-initiated or server initiated), the index should be optimized to handle frequent updates. The existing study on this issue can be adopted in this framework.

2.3 The Query Index

For each query, the database server stores: (1) the parameters of the query (e.g., the rectangle of a range query, the query point and the k value of a kNN query), (2) the current query results and, (3) the quarantine area of the query. The quarantine area is such areas that as long as all result objects stay inside it and all non-result objects stay outside it, the results of this query do not change. This area is used to identify the affected queries upon a source-initiated location update. For a range query, the quarantine area is simply the query rectangle; for a kNN query, the area can be any circle centered at the query point and completely covering the k-th NN_{OK} but not the k + 1-th $NN_{O_{k+1}}$. In other words, the radius of the circle should be equal to or greater than $\Delta(q, o_k)$. The maximum distance between the query point q and the safe region of o_k, but less than $\delta(q, o_{k+1})$, the minimum distance between q and the safe region of o_{k+1}. In this paper, we set the radius as the midpoint of these two distances.

III. QUERY EVALUATION AND REEVALUATION

Algorithm 1 Overview of Database Behavior

```
1: while receiving a request do
2: if the request is to register query q then
3: evaluate q (probe objects' locations if necessary);
4: compute q's quarantine area and insert it into the
   query index;
5: return the results to the application server;
6: else if the request is to deregister query q then
7: remove q from the query index;
8: else if the request is a location update from object p
   then
9: determine the set of affected queries;
10: for each affected query q′ do
11: reevaluate q′ (probe objects' locations if neces-
    sary);
12: update the results to the application server;
13: recompute q′'s quarantine area and update the
    query index;
14: update the safe region of p;
15: update the safe region of any probed object;
```

3.1 Evaluating New Range Query

Processing a new range query on safe regions is very similar to that on exact object locations. We start from the index root and recursively traverse down the index entries that overlap the query rectangle until reaching the leaf entries where the safe regions are stored. If the safe region of an object is fully covered by the query rectangle, the object is a result. Otherwise, they overlap and the object must be probed to resolve the ambiguity.

3.2 Evaluating New kNN Query

Evaluating an order-sensitive kNN query on safe regions is more complicated than that on exact object locations. We adopt the best-first search (BFS) as the paradigm for this algorithm. Similar to the original BFS, we maintain a priority queue which stores intermediate or leaf index entries (i.e., object locations). The object location is in the form of either a safe region or the exact point (after the server probes it). The key to sort the elements in the queue is the (minimum) distance to the query point q. The searching algorithm is the same as the original BFS except when a leaf entry, object p, is popped from the queue. If p is already represented by a point, it is returned as a result immediately. Otherwise p, represented by a safe region, is held until the next object u is popped. Then, the maximum distance between p and q ($\Delta(q,p)$) is compared with the minimum distance between u and q ($\delta(q,u)$). If the former is shorter, p is guaranteed closer to q than any other objects in the queue. As such, p is returned as a result. However, if the former is longer, the query result is undecided based on the safe regions. Therefore, p is probed and p together with its exact location is inserted back to the priority queue. And u is also inserted back to the queue. The algorithm continues until k objects are returned as results. In addition, in order to find the radius r of the quarantine area for this query, the algorithm pops one more element from the queue and the value of r is the midpoint between the keys of the k-th NN and the last popped element. It is worthwhile to note that this algorithm guarantees that the object is not probed until it is about to be returned.

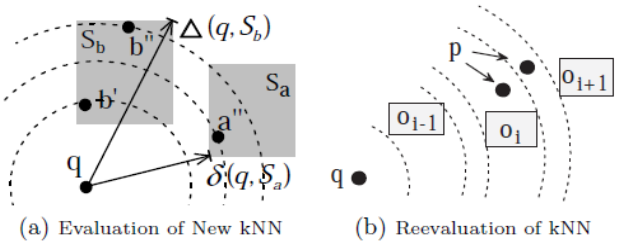

Fig 4: Processing a KNN Query

Algorithm 2 Evaluating a new kNN Query

Input: root: root node of object index
q: the query point
Output: C: the set of kNNs
r: the radius of the quarantine area
Procedure:
1: initialize the priority queue;
2: enqueue hroot, _(q, root)i into the queue;
3: while |C| < k and queue is not empty do
4: dequeue the top element to u;
5: if u is an object location then
6: if there is an object p held then
7: if _ (q, p) ≤ _(q, u) then
8: insert p into C;
9: else
10: enqueue hu, _(q, u)i back to the queue;
11: probe p;
12: enqueue hp, d(q, p)i back to the queue;
13: continue;
14: if u is represented by a safe region then
15: hold u;
16: else
17: insert u into C;
18: else if u is an index entry then
19: for each child entry v of u do
20: enqueue hv, _(v, q)i into the queue;
21: dequeue one more element to u;
22: r = (_(q,Ck) + _(q, u))/2;
23: return C and r;

IV. SAFE REGION COMPUTATION

The safe region of a moving object p (denoted as p.sr) designates how far p can reach without affecting the results of any registered query. As queries are independent of each other, we define the safe region for a query Q as the rectangular region in which p does not affect Q's result. p.srQ is essentially a rectangular approximation of Q's quarantine area or its complement. Obviously,p.sr is the intersection of individual p.srQ for all registered queries. To efficiently eliminate those queries whose p.srQ do not contribute to p.sr, we require p.sr (and p.srQ) to be fully contained in the grid cell in which p currently resides. By this means, we only need to compute p.srQ for those queries whose quarantine areas overlap this cell as the p.srQ for any rest query is the cell itself. These overlapping queries are called relevant queries and are exactly pointed by the bucket of this cell in the query index

V. CONCLUSION

We are currently deepening our research on this aspect, introducing other measures of similarity. We are also This paper proposes a generic framework for monitoring continuous spatial queries over moving objects. The frame-work distinguishes itself from existing work by being the first to address the location update issue and to provide a common interface for monitoring mixed types of queries. Based on the notion of safe region, the location updates are query aware and thus the wireless communication and query reevaluation costs are significantly reduced. We provide detailed algorithms for query evaluation/reevaluation and safe region computation in this framework. Enhancements are also proposed to take advantage of two practical mobility assumptions: maximum speed and steady movement. To evaluate the performance, we thoroughly conduct a series of experiments and compare the proposed framework with the optimal monitoring and the traditional periodic monitoring schemes.

REFERENCES

[1] Hayes, P. and Weinstein, S. CONSTRUE/TIS: a system for content-based indexing of a database of news stories.In IAAI-90,1990.

[2] K. Daphne and M. Sahami, "Toward Optimal Feature Selection," Proc 13th Int'l Conf. Machine Learning, pp. 284-292, 1996.

[3] Y. Yang and J.O. Pedersen, "A Comparative Study on Feature Select on in Text Categorization," Proc. 14th Int'l Conf. Machine Learning, pp. 412-420, 1997

[4] D.D. Lewis, "Feature Selection and Feature Extraction for Text Categorization," Proc. Workshop Speech and Natural Language, pp. 212-217, 1992.

[5] Fabrizio Sebastiani: Machine Learning in Automated Text Categorization. ACM Computing Surveys, Vol.34, No 1,PP. 1-47, March 2002.

AUTHORS

First Author – Bhakte D.B., PG Student, College of Engineering, Osmanabad (deep.bhakte@gmail.com)
Second Author – Prof.Vharkate M.N., ME (CSE), College of Engineering, Osmanabad (minakshi_ghodake1@yahoo.com)