PAM: An Efficient and Privacy-Aware Monitoring Framework for Continuously Moving Objects

Haibo Hu, Jianliang Xu, *Senior Member*, *IEEE*, and Dik Lun Lee

Abstract—Efficiency and privacy are two fundamental issues in moving object monitoring. This paper proposes a privacy-aware monitoring (PAM) framework that addresses both issues. The framework distinguishes itself from the existing work by being the first to holistically address the issues of location updating in terms of monitoring accuracy, efficiency, and privacy, particularly, when and how mobile clients should send location updates to the server. Based on the notions of safe region and most probable result, PAM performs location updates only when they would likely alter the query results. Furthermore, by designing various client update strategies, the framework is flexible and able to optimize accuracy, privacy, or efficiency. We develop efficient query evaluation/reevaluation and safe region computation algorithms in the framework. The experimental results show that PAM substantially outperforms traditional schemes in terms of monitoring accuracy, CPU cost, and scalability while achieving close-to-optimal communication cost.

Index Terms—Spatial databases, location-dependent and sensitive, mobile applications.

1 Introduction

In mobile and spatiotemporal databases, monitoring continuous spatial queries over moving objects is needed in numerous applications such as public transportation, logistics, and location-based services. Fig. 1 shows a typical monitoring system, which consists of a base station, a database server, application servers, and a large number of moving objects (i.e., mobile clients). The database server manages the location information of the objects. The application servers gather monitoring requests and register spatial queries at the database server, which then continuously updates the query results until the queries are deregistered.

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. Accuracy means how often the monitored results are correct, and it heavily depends on the frequency and accuracy of location updates. As for efficiency, two dominant costs are: the wireless communication cost for location updates and the query evaluation cost at the database server, both of which depend on the frequency of location updates. As for privacy, the accuracy of location updates determines how much the client's privacy is exposed to the server.

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) [24], [32], [35], [47]. However, these approaches have several deficiencies. First, the monitoring accuracy is low: query results are correct only at the time instances of periodic updates, but not in between them or at any time of deviation updates. Second, location updates are performed regardless of the existence of queries—a high update frequency may improve the monitoring accuracy, but is at the cost of unnecessary updates and query reevaluation. Third, the server workload using periodic update is not balanced over time: it reaches the peak when updates arrive (they must arrive simultaneously for correct results) and trigger query reevaluation, but is idle for the rest of the time. Last, the privacy issue is simply ignored by assuming that the clients are always willing to provide their exact positions to the server.

Some recent work attempted to remedy the privacy issue. *Location cloaking* was proposed to blur the exact client positions into bounding boxes [18], [14], [31], [26]. By assuming a centralized and trustworthy third-party server that stores all exact client positions, various location cloaking algorithms were proposed to build the bounding boxes while achieving the privacy measure such as k-anonymity. However, the use of bounding boxes makes the query results no longer unique. As such, query evaluation in such uncertain space is more complicated. A common approach is to assume that the probability distribution of the exact client location in the bounding box is known and well formed. Therefore, the results are defined as the set of all possible results together with their probabilities [14], [31], [7]. However, all these approaches focused on one-time cloaking or query evaluation; they cannot be applied to monitoring applications where continuous location update is required and efficiency is a critical concern.

- *H. Hu and J. Xu are with the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Kowloon, Hong Kong SAR, China. E-mail: {haibo, xujl}@comp.hkbu.edu.hk.*
- *D.L. Lee is with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong SAR, China. E-mail: dlee@cse.ust.hk.*

Manuscript received 2 Apr. 2008; revised 30 July 2008; accepted 25 Mar. 2009; published online 15 Apr. 2009.
Recommended for acceptance by S. Wang.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2008-04-0175.
Digital Object Identifier no. 10.1109/TKDE.2009.86.

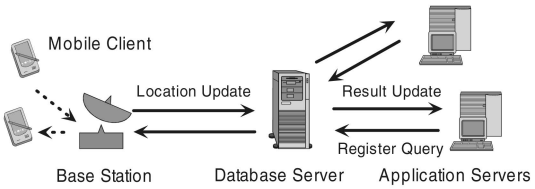


Fig. 1. The system architecture.

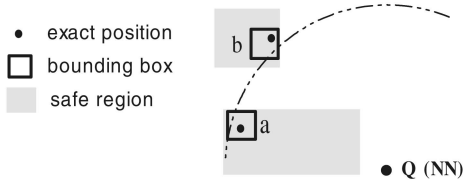


Fig. 2. Monitoring example.

In [21], we proposed a monitoring framework where the clients are aware of the spatial queries being monitored, so they send location updates only when the results for some queries might change. Our basic idea is to maintain a rectangular area, called *safe region*, for each object. The safe region is computed based on the queries in such a way that the current results of all queries remain valid as long as all objects reside inside their respective safe regions. A client updates its location on the server only when the client moves out of its safe region. This significantly improves the monitoring efficiency and accuracy compared to the periodic or deviation update methods. However, this framework fails to address the privacy issue, that is, it only addresses "when" but not "how" the location updates are sent.

In this paper, we take a more comprehensive approach—instead of dealing with "when" and "how" separately like most existing work, we propose a privacy-aware monitoring (PAM) framework that incorporates the accuracy, efficiency, and privacy issues **altogether**. We adapt for the monitoring environment the privacy model that has been employed by location cloaking and other privacy-aware approaches. More specifically, a client encapsulates its exact position in a bounding box, and the timing and mechanism with which the box is updated to the server are decided by a client-side location updater as part of PAM.

However, the integration of privacy into the monitoring framework poses challenges to the design of PAM. First, with the introduction of bounding boxes, the result of a query is no longer unique. Among all possible results, we argue that the most probable result, i.e., the one with the highest probability, is most promising for approximating the genuine result (the result derived based on the exact positions). The probability is computed by assuming a uniform distribution of the exact client position in the bounding box. Fig. 2 shows two clients a, b together with their bounding boxes. Both the genuine and most probable result for the 1NN query Q are $\{a\}$. However, even monitoring only the most probable result adds great complexity to query evaluation. As such, one of the main contributions of this paper is to devise efficient query processing algorithms for common spatial query types. Second, the most probable result also adds complexity to the definition of safe region. New algorithms must be designed to compute maximum safe regions in order to reduce the number of location updates, and thus, improve efficiency. Third, as the location updater decides when and how a bounding box is updated, its strategy determines the accuracy, privacy, and efficiency of the framework. The *standard* strategy is to update *when the centroid of the bounding box moves out of the safe region*, which guarantees accuracy—no miss of any change of the most probable result. To optimize privacy or efficiency, however, alternative strategies must be devised. Compared to the previous work, the PAM framework has the following advantages:

- To our knowledge, this is the first comprehensive framework that addresses the issue of location updating holistically with monitoring accuracy, efficiency, and privacy *altogether*. This framework extends from our previous work [21] by introducing a common privacy model, and therefore, suits realistic scenarios.
- As for efficiency, the framework significantly reduces location updates to only when an object is moving out of the safe region, and thus, is very likely to alter the query results.
- As for accuracy, the framework offers *correct* monitoring results *at any time*, as opposed to only at the time instances of updates in systems that are based on periodic or deviation location update.
- The framework is generic in the sense that it is not designed for a specific query type. Rather, it provides a common interface for monitoring various types of spatial queries such as range queries and kNN queries. Moreover, the framework does not presume any mobility pattern on moving objects.
- The framework is flexible in that by designing appropriate location update strategies, accuracy, privacy, or efficiency can be optimized.

In the rest of this paper, we will explore the PAM framework, especially on the aspects of query evaluation and safe region computation. The remainder of this paper is organized as follows: Section 2 reviews the related work. Section 3 overviews the framework components, followed by Sections 4 and 5 where query evaluation and safe region computation are presented, with an emphasis on range and kNN queries. Dynamic client update strategies are given in Section 6 to optimize privacy and efficiency. Experimental results of PAM are shown in Section 7.

2 Related Work

There is a large body of research work on spatial temporal query processing. Early work assumed a static data set and focused on efficient access methods (e.g., R-tree [19]) and query evaluation algorithms (e.g., [20], [37]). Recently, a lot of attention has been paid to moving-object databases, where data objects or queries or both of them move.

Assuming that object movement trajectories are known a priori, Saltenis et al. [38] proposed the Time-Parameterized R-tree (TPR-tree) for indexing moving objects, where the location of a moving object is represented by a linear function of time. Benetis et al. [3] developed query

evaluation algorithms for NN and reverse NN search based on the TPR-tree. Tao et al. [41] optimized the performance of the TPR-tree and extended it to the TPR^*-tree. Chon et al. [10] studied range and kNN queries based on a grid model. Patel et al. [34] proposed a novel index structure called STRIPES using a dual transformation technique.

The work on monitoring continuous spatial queries can be classified into two categories. The first category assumes that the movement trajectories are known. Continuous kNN monitoring has been investigated for moving queries over stationary objects [40] and linearly moving objects [22], [36]. Iwerks et al. [22] extended to monitor distance semijoins for two linearly moving data sets [23]. However, as pointed out in [39], the known-trajectory assumption does not hold for many application scenarios (e.g., the velocity of a car changes frequently on road).

The second category does not make any assumption on object movement patterns. Xu et al. [44] and Zhang et al. [48] suggested returning to the client both the query result and its validity scope where the result remains the same. As such, the query is reevaluated only when the query exits the validity scope. However, their solutions work for stationary objects only. For continuous monitoring of moving objects, the prevailing approach is periodic reevaluation of queries [24], [32], [35], [47]. Prabhakar et al. [35] proposed the Q-index, which indexes queries using an R-tree-like structure. At each evaluation step, only those objects that have moved since the previous evaluation step are evaluated on the Q-index. While this study is limited to range queries, Mokbel et al. [32] proposed a scalable incremental hash-based algorithm (SINA) for range and kNN queries. SINA indexes both queries and objects, and achieves scalability by employing shared execution and incremental evaluation of continuous queries [32], [43]. Kalashnikov et al. and Yu et al. suggested grid-based in-memory structures for object and query indexes to speed up reevaluation process of range queries [25] and kNN queries [47]. Access methods to support frequent location updates of moving objects have also been investigated [24], [29]. Our study falls into this category but distinguishes itself from existing studies with a comprehensive framework focusing on location update.

Uncertainty and privacy issues have been recently studied in moving object monitoring. To protect location privacy, various cloaking or anonymizing techniques have been proposed to hide the client's actual location. Among them are the spatiotemporal cloaking [18], the Clique-Cloak [14], [15], the Casper anonymizer [31], $hilbASR$ [17], [26], and peer-to-peer cloaking [11], [16]. In spatiotemporal cloaking, for each location update, the server divides the space recursively in a quad-tree-like format till a suitable subspace is found to cloak the updated location. The CliqueCloak algorithm constructs a clique graph to combine some clients who can share the same cloaked spatial area. The Casper anonymizer is associated with a query processor to ensure that the anonymized area returns the same query result as the actual location. In $hilbASR$, all user locations are sorted by Hilbert space-filling curve ordering, and then, every k users are grouped together in this order. Besides, location cloaking, *pseudonym*, *dummy*, and *transformation* were also proposed for privacy preservation. Pseudonym decouples the mapping between the user identity and the location so that an untrusted server only receives the location without the user identity [33], [4]. Dummy generates fake user locations (called dummies) and mixes them together with the genuine user location into the request [28], [46], [45]. Transformation utilizes certain one-way spatial transformations (e.g., a space filling curve) to map the query space to another space and resolves query blindly in the transformed space [27].

As for location uncertainty, a common model for characterizing the uncertainty of an object is a closed region with a predefined probability distribution of this object in the region. Based on this probabilistic model, query processing and indexing algorithms have been proposed to evaluate probabilistic range queries [12], [31] and kNN queries [9]. While in these studies, the objects are uncertain, the queries themselves are still certain. Chen and Cheng extended the probabilistic processing to more general cases where the queries are also uncertain [7]. Our study, on the other hand, addresses the continuous monitoring issue. By adopting the notion of "safe region," the frequency of query reevaluation on uncertain location information is reduced, and hence, the system efficiency and scalability are improved.

Distributed approaches have been investigated to monitor continuous range queries [6], [13] and continuous kNN queries [42]. The main idea is to shift some load from the server to the mobile clients. Monitoring queries have also been studied for distributed Internet databases [8], data streams [1], and sensor databases [30]. However, these studies are not applicable to monitoring of moving objects, where a two-dimensional space is assumed.

3 Fundamentals of PAM Framework

3.1 Privacy-Aware Location Model

In this paper, we assume that the clients are privacy conscious. That is, the clients do not want to expose their *genuine point locations* to the database server to avoid *spatiotemporal correlation inference* attack [14], by which an adversary may infer users' private information such as political affiliations, alternative lifestyles, or medical problems. For example, knowing that a user is inside a heart specialty clinic during business hours, the adversary can infer that the user might have a heart problem. This has been cited as a major privacy threat in location-based services and mobile computing. To protect against it, most existing work suggests replacing accurate point locations by bounding boxes to reduce location resolutions [18], [14], [31], [26], [7], [17]. With a large enough location box covering the sensitive place (e.g., the clinic) as well as a good number of other insensitive places, the success rate or confidence of such spatiotemporal correlation inference can be reduced significantly. In our monitoring framework, we take the same privacy-aware approach. Specifically, each time a client detects his/her genuine point location, it is encapsulated into a bounding box. Then, the client-side *location updater* decides whether or not to update that box to the server.[1] Without any other knowledge about the client locations or moving patterns, upon receiving such a box, the server can only presume that the genuine point location is distributed uniformly in this box. To simplify the presentation in this

1. The computation of a proper bounding box to satisfy a certain privacy metric (such as k-anonymity) has been extensively studied in the literature [14], [26], [17] and is beyond the scope of this paper. Nonetheless, the larger the box is, the less successful and confident the adversary's inference becomes.

paper, we further restrict the shape of such a bounding box to a δ-by-δ square (or in short δ-square), where δ is customizable for each object. Our problem is therefore to monitor result changes of spatial queries as objects move, and monitor them as accurately as possible and at the lowest cost of location updates.

The key idea to solving the problem is "safe region," which was defined in [21] as a rectangle *within* which the change of object location does not change the result of any registered spatial query. Now that locations are δ-squares instead of points, to clarify the definition of "within," we use the *centroid* point of the square as a representative, so the safe region is essentially a safe region for the centroid of the δ-square. However, the consequence of introducing δ-square is more than that—the result of a spatial query is no longer unique. For example, if the δ-square of an object partially overlaps with a range query, this object could be either a result object or a nonresult object of this query. As such, a unique definition of query result under δ-squares is a prerequisite of safe region.

Since the genuine point location of an object is distributed uniformly in its δ-square, we can define the (unique) query result as the one with the highest probability among all possible results. As in the previous range query example, if the majority of the δ-square falls inside the range query, that object is most probably a result object of this query; otherwise, that object is most probably a nonresult object. With the notion of most probable result, we thereby define the safe region as a rectangle within which the change of the centroid of the object's δ-square does not change the most probable result of any registered spatial query. The *standard update strategy* of the client is therefore "to update when the centroid of the δ-square is out of the safe region."

The reason why we exclude all other less probable results in this definition is threefold: 1) monitoring continuous queries usually trades accuracy for efficiency—although the most probable result does not always align with the *genuine result* (the result derived based on genuine point locations of all objects), we will show in Section 4 that it is efficient to compute, and therefore, prevents the server from being computationally overloaded; 2) if the query result were defined as the set of all possible results, the safe region would have to be extremely small to report location updates if any of the possible results changes, which makes the update cost overwhelmingly high; and 3) we do not want the choice of δ-square—which is made by the client—to affect query results heavily, and obviously the most probable results are less vulnerable than other result definitions.

3.2 Framework Overview

As shown in Fig. 3, the PAM framework consists of components located at both the database server and the moving objects. At the database server side, we have the moving object index, the query index, the query processor, and the location manager. At moving objects' side, we have location updaters. Without loss of generality, we make the following assumptions for simplicity:

- The number of objects is some orders of magnitude larger than that of queries. As such, the query index can accommodate all registered queries in main memory, while the object index can only accommodate all moving objects in secondary memory. This assumption has been widely adopted in many existing proposals [25], [47], [21].
- The database server handles location updates sequentially; in other words, updates are queued and handled on a first-come-first-serve basis. This is a reasonable assumption to relieve us from the issues of read/write consistency.
- The moving objects maintain good connection with the database server. Furthermore, the communication cost for any location update is a constant. With the latter assumption, minimizing the cost of location updates is equivalent to minimizing the total number of updates.

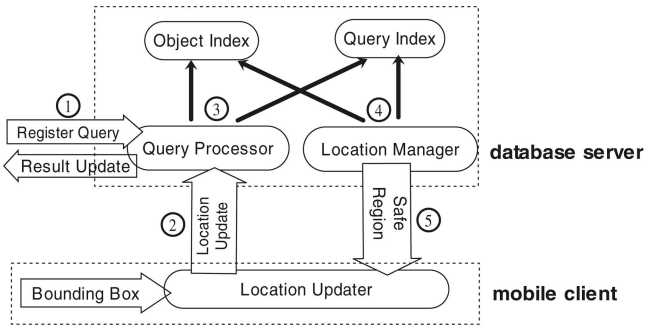


Fig. 3. PAM framework overview.

PAM framework works as follows (see Fig. 3): At any time, application servers can register spatial queries to the database server (step ①). When an object sends a location update (step ②), the query processor identifies those queries that are affected by this update using the query index, and then, reevaluates them using the object index (step ③). The updated query results are then reported to the application servers who register these queries. Afterward, the location manager computes the new safe region for the updating object (step ④), also based on the indexes, and then, sends it back as a response to the object (step ⑤). The procedure for processing a new query is similar, except that in step ②, the new query is evaluated from scratch instead of being reevaluated incrementally, and that the objects whose safe regions are changed due to this new query must be notified. Algorithm 1 summarizes the procedure at the database server to handle a query registration/deregistration or a location update.

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;
4:     compute its quarantine area and insert it into the
         query index;
5:     return the results to the application server;
6:     update the changed safe regions of objects;
7:   else if the request is to deregister query q then
8:     remove q from the query index;
9:   else if the request is a location update from object p
       then
10:    determine the set of affected queries;
11:    for each affected query q' do
```

12: reevaluate q';
13: update the results to the application server;
14: recompute its quarantine area and update the query index;
15: update the safe region of p;

It is noteworthy that although in this paper, the most probable result is used, this framework can also adapt to other query result definitions such as over a probability confidence (e.g., "returns objects that have 90 percent probability inside the query range"). The only changes needed to reflect the new result definition are the query evaluation algorithms in the query processor and safe region computation in the location manager. In the rest of this paper, we stick to the definition of the most probable result and leave the modification details for other definitions to interested readers.

The following sections explain the components at the database server in detail, and Section 6 describes the update strategy of the client-side location updater.

3.3 The Object Index

The object index is the server-side view on all objects. More specifically, to evaluate queries, the server must store the spatial range, in the form of a bounding box, within which each object can possibly locate. Note that this bounding box is different from a δ-square because its shape also depends on the client-side location updater. That is, it must be a function (denoted by $\odot$) of the last updated δ-square and the safe region. As such, this box is called a *bbox* as a mark of distinction. In particular, for the standard update strategy, the $bbox$ is the safe region enlarged by $\delta/2$ on each side, or formally, the "Minkowski sum"[2] of the safe region and a $\delta/2$-square.

With the same rationale for which we assume the genuine point location of an updating object to distribute uniformly in the δ-square, we assume that the genuine point locations are distributed uniformly in their respective $bbox$es when queries are evaluated or reevaluated. The object index is built on the $bbox$es to speed up the evaluation. While many spatial index structures can serve this purpose, this paper employs the R*-tree index [2], [19], which is most widely adopted in the literature. Since the $bbox$ changes each time the object updates, the index is optimized to handle frequent updates [29].

3.4 The Query Index

For each registered query, the database server stores: 1) the query parameters (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 used to identify the queries whose results might be affected by an incoming location update. It originates from the *quarantine line*, which is a line that splits the entire space into two regions: the inner region and the outer region. An object becomes a result object if it enters the inner region; likewise, it becomes a nonresult object once it enters the outer region.

2. The Minkowski sum of two shapes A and B in euclidean space is the result of adding every point in B to every point in A, i.e., the set $\{a+b|a \in A, b \in B\}$.

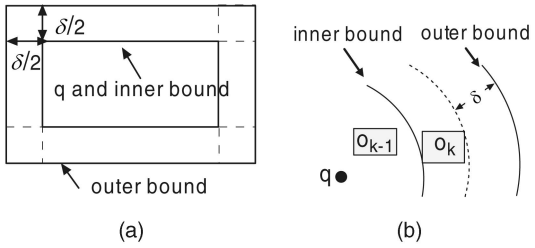


Fig. 4. Quarantine area. (a) Range query. (b) kNN query.

However, the ideal quarantine line is difficult to compute, especially in the context of the most probable result. In addition, as object locations have extensions rather than points, the quarantine line is not unique for a query. As such, we allow fuzziness by relaxing the line to an area called "quarantine area." That is, the entire space is split into three regions: the inner region, the quarantine area, and the outer region. The former two are separated by the *inner* bound of the quarantine area, whereas the latter two are separated by the*outer* bound of the quarantine area. To ease the computation of these two bounds, an object becomes a result object if its δ-square moves totally inside the inner bound; on the other hand, an object becomes a nonresult object once its δ-square crosses or is outside the outer bound. Therefore, a query Q is not affected only if "of the updated δ-square p and its last updated δ-square p_{lst}, both of them are totally inside the inner bound or both of them cross or are outside the outer bound of the quarantine area."[3]

For a range query q, the query window can serve as an inner bound of the quarantine area, because any object whose δ-square is fully inside q is a trivial result of q. On the other hand, an outer bound can be the Minkowski sum of q and a $\delta/2$-square, i.e., enlarging q by $\delta/2$ on each side. The correctness of this bound can be verified by the observation that for any δ-square that crosses this bound, the majority of this square must be outside q, thus making the object a nonresult object. In case, there are different δs for different objects, the largest δ is used. Fig. 4a shows the inner and outer bounds of q's quarantine area.

For a kNN query, since only the distance to the query point q matters, we set both the inner and the outer bounds as circles centered at q. Furthermore, since the kth NN o_k determines whether other object is or is not a result object, we set the radii of the two circles based on o_k. More specifically, the inner bound circle is set to be the minimum distance between q and the $bbox$ of o_k so that if a δ-square is totally inside this circle, it is guaranteed to be closer to q than o_k. On the other hand, the outer bound circle is set to be the maximum distance between q and the $bbox$ of o_k, plus δ. If $d(s,t)$ denotes the distance between two points s and t, $d(S,T)$ ($D(S,T)$) denotes the minimum (maximum) distance between a pair of points in areas S and T, then the radii of the inner and outer circle are $d(q,o_k)$ and $D(q,o_k)+\delta$, respectively.

To quickly find all affected queries, an in-memory grid-based index is built on the quarantine areas of all queries.

3. For kNN queries, if the order of the result objects is sensitive, Q is not affected only if both of them cross or are outside the outer region.

The index partitions the entire space into $M \times M$ uniform grid cells, and the bucket for each cell points to those queries whose quarantine areas overlap with or fully enclose this cell. If we define the cell(s) that overlap with the δ-square of the updating object p as the *home* cell(s) of p, then only queries pointed at by the home cell(s) of p or p_{lst} are affected, and thus, need reevaluation.

3.5 Query Processor and Location Manager

In the PAM framework, based on the object index, the query processor evaluates the most probable result when a new query is registered, or reevaluates the most probable result when a query is affected by location updates. Obviously, the reevaluation is more efficient as it can be based on previous results. The detailed algorithms of query evaluation and reevaluation will be presented in Section 4.

The location manager computes the safe region of an object p (denoted as $p.sr$). Recall that a safe region is a rectangle within which the change of the centroid of p's δ-square does not change the most probable result of any registered query. As queries are independent of each other, we can further define the safe region for a single query Q (denoted as $p.sr_Q$) as a rectangle in which the change of the centroid of p's δ-square does not change the most probable result of Q. By this definition, $p.sr_Q$ is a rectangular approximation, or more accurately an inscribed rectangle, of Q's inner (if p is a result object) or outer (if p is a nonresult object) regions, which are separated by the quarantine line. The reason why the safe region is based on the quarantine line rather than the quarantine area is that the latter is much coarser. Furthermore, the quarantine area is used only to filter out the queries that are not affected by a location update, so we trade accuracy for efficiency. The safe region, on the other hand, directly dictates the frequency, and hence, the cost of location updates, so we compute it based on the more accurate quarantine line.

After each individual $p.sr_Q$ is computed, $p.sr$ is simply the intersection of these $p.sr_Q$ from all registered queries. To eliminate those queries whose safe regions do not contribute to $p.sr$, the location manager further requires every $p.sr_Q$ (and thus, the $p.sr$) to be fully contained in the home cell(s). Recall that the home cell(s) are the grid cell(s) of the query index where the δ-square of p is contained or overlaps. By this means, the location manager only needs to compute the safe regions for those queries (subsequently called *relevant queries*) whose quarantine areas are contained or overlap with the home cell(s). These relevant queries are exactly those indexed by the home cell(s) of the query index.

The location manager recomputes the safe region of an object p in two cases: 1) after a new query Q is evaluated and 2) after p sends a location update. In the former case, since no existing queries change their quarantine lines, the new safe region $p.sr'$ is simply the intersection of the current safe region $p.sr$ and $p.sr_Q$, the safe region for this new query Q. If $p.sr'$ is different from $p.sr$, the new safe region should be updated to p. In the latter case, the quarantine areas of some existing queries might change; therefore, $p.sr'$ needs to be completely recomputed by computing the $p.sr_Q$ for each relevant query and then getting the intersection.

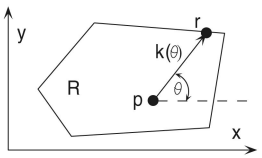


Fig. 5. Random movement.

As the objective of the PAM framework is to minimize the number of location updates, the following theorem shows that the safe region should be the inscribed rectangle of the inner or outer region with the maximum perimeter:

Theorem 3.1. *Assume that the object p moves in a randomly chosen direction with a constant speed ϕ (see Fig. 5), and that δ-square is small enough to be ignored. Given a convex safe region R and the updated location p, the amortized location update cost for p over time $Cost_p$ is*

$$Cost_p = C_l \cdot \left(\int_0^{2\pi} \frac{k(\theta)d\theta}{2\pi\phi} \right)^{-1} \doteq \frac{C_l \cdot 2\pi\phi}{Perimeter(R)},$$

where C_l is the cost for one location update, θ is the angle between the moving direction and the positive x-axis, $k(\theta)$ is the length of segment $\overline{pr}$, r is the intersection point of this direction and the boundary of R, in other words, r is the location at which the next location update occurs.

Proof. First of all, r must be unique for every θ. Otherwise, if there were another r', the points in segment rr' do not belong to R, which contradicts the convex assumption. As such, given θ, the elapsed time before the next location update is $\frac{k(\theta)}{\phi}$. The average elapsed time over all θ is

$$\frac{\int_0^{2\pi} \frac{k(\theta)}{\phi} d\theta}{\int_0^{2\pi} d\theta} = \int_0^{2\pi} \frac{k(\theta)d\theta}{2\pi\phi}.$$

Therefore, we have

$$Cost_p = C_l \cdot \left(\int_0^{2\pi} \frac{k(\theta)d\theta}{2\pi\phi} \right)^{-1} \doteq \frac{C_l \cdot 2\pi\phi}{Perimeter(R)},$$

because $\int_0^{2\pi} k(\theta)d\theta \doteq Perimeter(R)$. □

Therefore, the optimal safe region $p.sr_Q$ is the inscribed rectangle with the longest perimeter, or shortly $Ir - lp$, of Q's inner or outer region. Section 5 will present the detailed algorithms to compute the optimal $p.sr_Q$ for each type of query.

4 Query Processing

In this section, we present the detailed algorithms to evaluate or reevaluate a spatial query Q in terms of the most probable result. Aside from the definition of the query result, we know that Q also differs from a conventional spatial query in that the object locations are in the form of δ-square (for updating objects) or $bbox$ (for other objects), both of which are rectangular. In this section, instead of regarding Q as a special query type, we take an alternative approach by regarding the space where the object locations are defined as a special euclidean space. In this space, spatial relations such as *overlapping*, *containment*, or even

distance are implemented differently from a conventional euclidean space. By using the new implementations of spatial relations, existing spatial query processing algorithms can be applied directly to the new space.

In the following sections, we implement two relations that are required for spatial queries, namely, containment and closer.

4.1 Spatial Relations

In this new space, an object p is *contained* in a rectangle R if in the euclidean space, the majority of p is in R. The rectangle divides the enlarged safe region of any object p into two regions: the region inside rectangle q (where p is a result object of q) and the region outside q (where p is not a result). The region with the larger area decides the most probable result.

In this new space, an object p_1 is *closer* to a point q than object p_2 if and only if in the euclidean space, for two randomly picked points a, b from p_1 and p_2, respectively, a is equally or more probably closer to q than b. The *closer* relation has a nice property that it is a total order relation, which is proved by the following preposition:

Proposition 4.1. *The closer relation is a total order relation, that is, it satisfies*

1. *reflexivity,*
2. *antisymmetry,*
3. *transitivity, and*
4. *comparability.*

Proof sketch. Cases 1 and 2 are trivial.

3. **Transitivity**: if p_1 is closer than p_2 and p_2 is closer than p_3, then a (from p_1) is more probably closer to q than b (from p_2), which is, in turn, more probably closer to q than c (from p_3). As such, p_1 is closer than p_2.

4. **Comparability**: $\forall p_1, p_2$, either p_1 is closer than p_2, or p_2 is closer than p_1. □

Therefore, the most probable result of kNN query q is defined as the top-k objects of all objects in the closer order of their enlarged safe regions.

To implement the "closer" relation, we present an efficient algorithm that is based on finding out which object has more portion of area closer to point q. Instead of computing the exact shape of such area, which is forbiddingly costly, the algorithm is based on the divide-and-conquer paradigm. It maintains a priority queue $\mathcal{Q}$ whose elements are pairs of subrectangles of p_1 and p_2 that have not yet been compared. Initially, the pair $<p_1, p_2>$ is inserted into $\mathcal{Q}$ and the portion of area where p_1 (or p_2) is closer is 0. Each time an element $<p_1', p_2'>$ pops up from $\mathcal{Q}$ (where p_1' is a subrectangle of p_1 and p_2' is a subrectangle of p_2), the algorithm checks if any point in p_1' (or p_2') is always closer than any point in p_2' (or p_1'). If this is the case (case 1), the multiple of the area p_1' (or p_2') is added to the portion of area where p_1 (or p_2) is closer. If this is not the case (case 2), p_1' or p_2', whichever is larger, is split into four equal subrectangles, and thus, four new pairs are inserted to $\mathcal{Q}$. The reason to split the larger rectangle is that the resulted pairs are more probable to become pairs of case 1. The algorithm continues until either the portion of area where p_1 (or p_2) is closer exceeds 0.5, or the queue $\mathcal{Q}$ becomes empty. It is noteworthy that the portion of area where p_1 (or p_2) is closer is essentially the probability that p_1 (or p_2) is closer. As such, this algorithm always returns the correct result. On the other hand, the algorithm is efficient because it terminates as soon as one portion of area exceeds 0.5. In order to let the portion of area converge to the actual probability more quickly, we use the multiple portions of area as the key to sort the pairs in $\mathcal{Q}$.

4.2 Query Evaluation and Reevaluation on Object Index

In conventional euclidean space, a new range query is evaluated as follows: We start from the index root and recursively traverse down the index entries that overlap with the query window until the leaf entries storing the objects are reached. Then, we test each object using the containment relation in the new space.

Reevaluation of an existing range query q is even simpler—only the δ-square of the updating object needs to be tested on the containment relation.

The best-known algorithm to evaluate a kNN query q in conventional euclidean space is the best-first search (BFS) [20]. It uses a priority queue H to store the to-be-explored index entries which may contain kNNs. The entries in H are sorted by their minimum distances to the query point q. BFS works by always popping up the top entry from H, pushing its child entries into H, and then, repeating the process all over. When a leaf entry, i.e., an entry of a leaf node, is popped, the corresponding object is returned as a nearest neighbor. The algorithm terminates if k objects have been returned.

In the new space, the query is evaluated similarly, which is shown in Algorithm 2. However, the algorithm maintains an additional priority queue $\mathcal{H}$ besides H. It is a priority queue of objects sorted by the "closer" relation. The reason to introduce $\mathcal{H}$ is that when an object p is popped from H, it is not guaranteed a kNN in the new space. Therefore, $\mathcal{H}$ is used to hold p until it can be guaranteed a kNN. This occurs when another object p' is popped from H, and its minimum distance to q ($d(q, p')$) is larger than the maximum distance of p to q ($D(q, p)$). In general, when an object u is popped from H, we need to do the following. If $d(q, u)$ is larger than $D(q, v)$, where v is the top object in $\mathcal{H}$, then v is guaranteed a kNN and removed from $\mathcal{H}$. Then, $d(q, u)$ is compared with the next $D(q, v)$ until it is no longer the larger one. Then, u itself is inserted to $\mathcal{H}$ and the algorithm continues to pop up the next entry from H. The algorithm continues until k objects are returned.

Algorithm 2. Evaluating a new kNN Query
Input: $root$: root node of object index
q: the query point
Output: C: the set of kNNs
Procedure:
1: initialize queue H and $\mathcal{H}$;
2: enqueue $\langle root, d(q, root)\rangle$ into H;
3: **while** $|C| < k$ and H is not empty **do**
4: $u = H$.pop();
5: **if** u is a leaf entry **then**
6: **while** $d(q, u) > D(q, v)$ **do**

```
        v = H.pop();
        insert v to C;
      enqueue u into H;
  else if u is an index entry then
      for each child entry v of u do
        enqueue ⟨v, d(v, q)⟩ into H;
```

To reevaluate an existing kNN query that is affected by the updating object p, the first step is to decide whether p is a result object by comparing p with the kth NN using the "closer" relation: if p is closer, then it is a result object; otherwise, it is a nonresult object. This then leads to three cases: 1) case 1: p was a result object but is no longer so; 2) case 2: p was not a result object but becomes one; and 3) case 3: p is and was a result object.[4] For case 1, there are fewer than k result objects, so there should be an additional step of evaluating a 1NN query at the same query point to find a new result object u. The evaluation of such a query is almost the same as Algorithm 2, except that all existing kNN result objects are not considered. The final step of reevaluation is to locate the order of new result object p in the kNN set. This is done by comparing it with other existing objects in the kNN set using the "closer" relation. For cases 1 and 2, since this object is a new result object, the comparison should start from the kth NN, then k-1th NN, and so on. However, for case 3, since p was in the set, the comparison can start from where p was. Algorithm 3 shows the pseudocode of kNN query reevaluation, where p^* denotes the starting position of the comparison.

Algorithm 3. Reevaluating a kNN Query
Input: C: existing set of kNNs
p: the updating object
Output: C: the new set of kNNs
Procedure:

```
1: if p is closer to the k-th NN then
2:     if p ∈ C then
3:         p* = the rank of p in C;
4:     else
5:         p* = k;
6:         enqueue p into C;
7: else
8:     if p ∈ C then
9:         evaluate 1NN query to find u;
10:        p* = k;
11:        remove p and enqueue u into C;
12: relocate p or u in C, starting from p*;
```

5 Safe Region Computation

As mentioned in Section 3, the location manager computes the optimal safe region for an individual query Q, which is the inscribed rectangle with the longest perimeter ($Ir - lp$) of Q's inner or outer region, separated by the quarantine line. Therefore, the safe region is obtained in two steps: finding the quarantine line, and then, finding the $Ir - lp$. It is noteworthy that the safe region must contain the updating object p (i.e., its centroid), because otherwise, this object has to send an immediate location update after it receives this safe region. In this section, we present the detailed algorithms to compute the quarantine line, and hence, the safe region for various types of queries.

4. There is a fourth case where p was not and is not a result object. In this case, the reevaluation is completed by doing nothing.

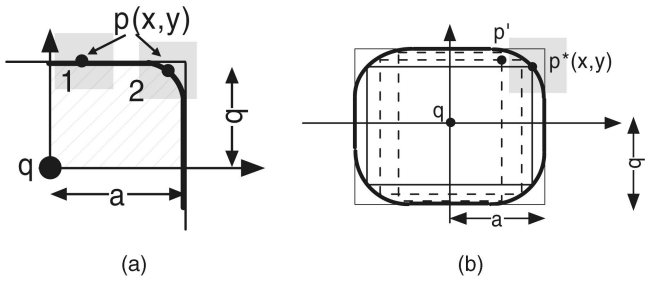


Fig. 6. Safe region for range query. (a) Quarantine line. (b) The optimal safe region.

5.1 Safe Region for Range Query

We first consider the case when object p is a result object. Fig. 6b shows an example of range query where q is the centroid of the query. The gray box shows the δ-square of p. Without loss of generality, let us consider the first quadrant, and let the same p ((x, y)) denote the centroid of the δ-square. Fig. 6a is the close-up image of Fig. 6b. According to the definition of the most probable result, more than half of the δ-square must reside in the query window. To obtain the quarantine line, we only need to consider the special case when exactly half of the square resides in the query window, which can be further divided into two subcases. In the first subcase, p is "on" the window border as box "1" shows, we have either "$y = b$ and $x + \delta/2 \leq b$" or "$x = a$ and $y + \delta/2 \leq a$." In the second subcase, p is not on the border as box "2" shows, we have $(\delta/2 + a - x)(\delta/2 + b - y) \geq \delta^2/2$. The two subcases give us the quarantine line in the first quadrant (the bold curve in Fig. 6a), which is defined by the following formulae:

$$\begin{cases} x = a, & if\ y \leq b - \delta/2, or \\ y = b, & if\ x \leq a - \delta/2, or \\ (\delta/2 + a - x)(\delta/2 + b - y) = \delta^2/2, & otherwise. \end{cases} \quad (1)$$

And the inner region in the first quadrant is therefore the shaded shape. Summing up all the four quadrants, the total inner region of this query is the bold shape in Fig. 6b.

The second step is to find the $Ir - lp$ of the inner region. For any inscribed rectangle whose corner point in the first quadrant is (s, t), the perimeter is $2s + 2t$. On the other hand, since (s, t) must also be on the quarantine line, $x = s, y = t$ must be a solution to (1). This equation shows that the perimeter $2s + 2t$ is maximized at p^* when $\frac{\delta}{2} + a - x = \frac{\delta}{2} + b - y = \frac{\delta}{\sqrt{2}}$. Thus, the optimal safe region is the solid rectangle whose corner point is p^* (see Fig. 6b). However, this safe region may not contain the centroid of the updating object p. For example, in Fig. 6b, if the centroid is at p', then all inscribed rectangles that contain p lie between the two dotted rectangles whose horizontal sides and vertical sides pass p'. In this case, the optimal safe region is one of the two dotted rectangles with longer

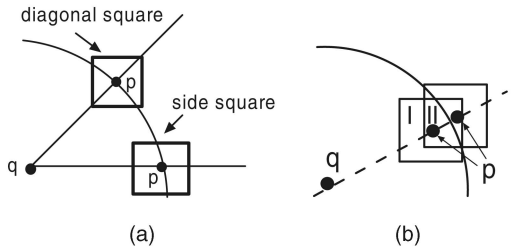


Fig. 7. Lemmas on squares. (a) Diagonal and side squares. (b) q and inside part.

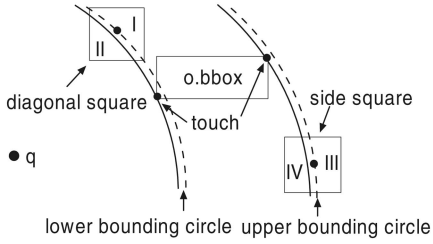


Fig. 8. Upper and lower bounding circles.

perimeter. Therefore, we reach the following proposition on the safe region for a result object:

Proposition 5.1. *For a result object p of a range query (size 2a-by-2b), the corner of the safe region is:*

$$(x,y) = \begin{cases} \left(a - \frac{\sqrt{2}-1}{2}\delta, b - \frac{\sqrt{2}-1}{2}\delta\right), & if\ p.x \le a - \frac{\sqrt{2}-1}{2}\delta, \\ & \&\ p.y \le b - \frac{\sqrt{2}-1}{2}\delta, \\ \left(p.x, \frac{\delta^2}{2(a+\delta/2-p.x)} - \frac{2b+\delta}{2}\right), & \\ or\ \left(\frac{\delta^2}{2(b+\delta/2-p.y)} - \frac{2a+\delta}{2}, p.y\right), & \text{otherwise.} \end{cases}$$

If p is a nonresult object, the safe region is an inscribed rectangle of the outer region. Such rectangle has the longest perimeter when its corner point p^* is at $(a,0)$ or $(0,b)$. Similar to the case when p is a result object, this rectangle can serve as the safe region only if it contains the centroid of updating object p; otherwise, the safe region is chosen from the two dotted rectangles that has a longer perimeter.

5.2 Safe Region for kNN Query

We first consider the case when object p is the ith NN (denoted by o_i) of the query. By definition, its δ-square must be *closer* than the $bbox$ of o_{i+1}, but *farther* than the $bbox$ of o_{i-1}. However, the exact quarantine line (and hence, the inner or outer region) for p based on this line is complex. In what follows, we approximate the inner region with a ring centered at the query point q.

As the first step, we show that a circle centered at q splits a δ-square into *inside* and *outside* parts, and their areas are dependent on the angle of the δ-square to q.

Lemma 5.2. *Among all squares of the same size and the same distance to q, the diagonal square, whose diagonal coincides with the line of $\overline{pq}$, has the smallest inside part, while the side square, whose sides are parallel to $\overline{pq}$, has the largest inside part. (refer Fig. 7a).*

On the other hand, the area of the inside part also depends on the length of $\overline{pq}$. For example, in Fig. 7b, the two δ-squares are of the same angle, but the square that is closer to q has a larger inside part (area I) than the farther square (area II).

Lemma 5.3. *For squares of the same angle to q, the closer p to q, the larger the inside part.*

Applying these two lemmas, we can define the *lower* and *upper* bounding circles for an object o. In Fig. 8, there are two circles, plotted by solid arcs, that touch the near and the far endpoints of the $bbox$ of o. Then, there must be a diagonal square and a side square that are split by these two arcs into inside and outside parts of equal area, respectively. The lower and upper bounding circles, plotted by dotted arcs, are the circles that cross the centers of these two squares. By this definition, as long as the centroid of p's δ-square is within the lower bounding circle, p is always closer than o; on the other hand, as long as the centroid is beyond the upper bounding circle, p is always farther than o. The following proposition proves the correctness:

Proposition 5.4. *Any δ-square whose centroid is within (beyond) the lower (upper) bounding circle for object o must be closer (farther) to q than o.*

Proof. Since any point in the inside part of the diagonal square (i.e., area II) is always closer than any point in the $bbox$ of o, and since the inside part is half of the square, by definition, the diagonal square is closer than the $bbox$ of o. On the other hand, by Lemmas 5.2 and 5.3, any square whose center is closer than that of the diagonal square must be closer than the diagonal square. Applying the transitivity of the "closer" relation, any square whose centroid is within the lower bounding circle is closer to q than o. The proof for the upper bound is similar. □

Based on Proposition 5.4, the inner region for p (i.e., o_i) can be approximated by a ring that is formed by the lower bounding circle for o_{i+1} and the upper bounding circle for o_{i-1}. To find the radii of the upper and lower bounding circles, we further adopt an approximation algorithm as follows: As shown in Fig. 9, to compute the lower bounding circle, the diagonal square is first partitioned into $M \times M$ (e.g., 4×4) subsquares. Then, the distance between q and the farthest endpoint (the small hollow or solid circles in the figure) of each subsquare is computed. The medium (i.e., the $\frac{M^2}{2}$th shortest) distance is set to the radius for the lower bounding circle. This bounding circle is guaranteed to satisfy Proposition 5.4 because the subsquares of the first $\frac{M^2}{2}$ shortest distances (their farthest endpoints are shown as hollow circles) must be inside the bounding circle, and these subsquares already account for half of the total area.

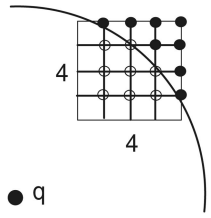
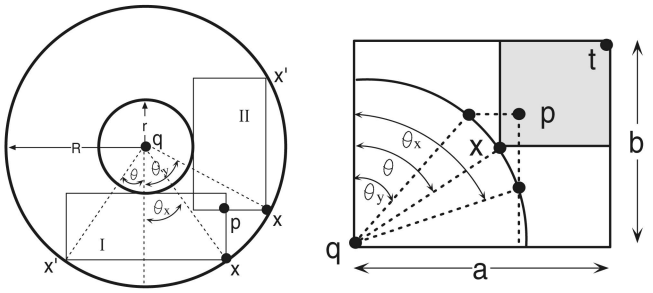


Fig. 9. Bounding circle.



(a) (b)

Fig. 10. Computing *Ir-lp*. (a) Ring. (b) Complement of a circle.

Therefore, this circle can serve as an approximation of the lower bounding circle. The same approximation can be applied to upper bounding circles. Obviously, better approximation can be achieved using larger M, which is at the cost of higher computation overhead.

Once the ring is obtained, the safe region is the inscribed rectangle of the ring that has the longest perimeter ($Ir-lp$). In [21], we showed the following proposition (see Fig. 10a):

Proposition 5.5. *The Ir-lp of a ring that is centered at q with inner radius r and outer radius R is the one of the following two Ir-lp which has a longer perimeter. The perimeter of the first (horizontal) Ir-lp is $4Rsin\theta_1 + 2(Rcos\theta_1 - r)$, where θ_1 is*

$$\theta_1 = \begin{cases} arctg2, & if\ \theta_x \le arctg2 \le \theta_y, or \\ \theta_x, & if\ \theta_x < arctg2, or \\ \theta_y, & if\ arctg2 < \theta_y, \end{cases}$$

where $\theta_x = arcsin\frac{p.x-q.x}{R}$ and $\theta_y = arccos\frac{q.y-p.y}{R}$. The perimeter of the second (vertical) Ir-lp is $4Rcos\theta_2 + 2(Rsin\theta_2 - r)$, where θ_2 is

$$\theta_2 = \begin{cases} arcctg2, & if\ \theta_x \le arcctg2 \le \theta_y, or \\ \theta_x, & if\ \theta_x < arcctg2, or \\ \theta_y, & if\ arcctg2 < \theta_y. \end{cases}$$

Finally, we reach the following proposition on the safe region for a result object o_i:

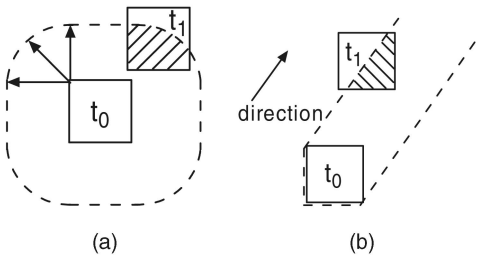
Proposition 5.6. *The safe region of the ith NN o_i is the Ir-lp of the ring that consists of the upper bounding circle for o_{i-1} and the lower bounding circle for o_{i+1}.*

It is noteworthy that for the first NN (i.e., $i = 1$), the ring degenerates to a circle. On the other hand, if object p is a nonresult object, we can approximate the outer region by the complement of the upper bounding circle of o_k. As such, the safe region is the $Ir - lp$ of the complement of a circle. In [21], we showed that (see Fig. 10b):

Proposition 5.7. *The Ir-lp of the complement of a circle centered at q with radius r is the inscribed rectangle with one corner being the cell corner corresponding to p and the opposite corner is x. x is either on the $1/4$ circle whose θ is*

$$\theta = \begin{cases} \pi/4, & if\ \theta_y \le \pi/4 \le \theta_x, or \\ \theta_x, & if\ \theta_x < \pi/4, or \\ \theta_y, & if\ \theta_y > \pi/4, \end{cases}$$

where $\theta_x = arcsin\frac{p.x-q.x}{r}$ and $\theta_y = arccos\frac{p.y-q.y}{r}$.



(a) (b)

Fig. 11. Problems with mobility knowledge. (a) Known maximum speed. (b) Known direction.

6 Dynamic Client Update Strategy

The standard update strategy, which updates when the centroid of δ-square is out of the safe region, guarantees 100 percent monitoring accuracy in the context of the most probable result. This is a static strategy where the decision is made independent of previous decisions. In this section, we discuss two dynamic strategies that achieve objectives other than monitoring accuracy.

6.1 Mobility-Aware Update Strategy

Previously, we ignore the fact that the server receives a series of location updates from an object. Although the server cannot speculate the genuine object location from an individual δ-square, by considering consecutive updates with certain background knowledge about the object's mobility, the server might produce better speculations.

Figs. 11a and 11b show two examples where the maximum speed v_m or the exact direction of the movement is known, respectively. In these examples, a δ-square is updated at time t_0, then at time t_1, the object must reside in the dotted shape, which is called the *reachable area* from t_0. In Fig. 11a, the reachable area is the Minkoski sum of the δ-square at t_0 and a circle with a radius of $v_m(t_1 - t_0)$, i.e., the δ-square expanded by the circle at each point. Likewise, in Fig. 11b, the reachable area is the half-open space formed by the rays whose ends are from the δ-square. If the δ-square at t_1 overlaps with the reachable area, then the object can only locate in the part that is inside the area (shaded in Figs. 11a and 11b).

To prevent the server from narrowing down the object location like this, we propose the following mobility-aware strategy:

Definition 6.1 (Mobility-aware update strategy). *Update when the centroid of δ-square is out of the safe region and the δ-square is completely inside the reachable area of all previous δ-squares.*

The intuitive version of this strategy must maintain the entire set of historic δ-squares. However, due to its dynamic property, we show in the following lemma that it is sufficient to maintain only the reachable area for the last δ-square:

Lemma 6.1. *For a set of δ-squares of $\{t_0, t_1, \ldots, t_n\}$ and $i \le j \le n$, the reachable area of t_j is completely inside that of t_i as long as the δ-square of any t_i is completely inside the reachable area of $t_{i-1} (i \ge 1)$.*

In what follows, we develop an algorithm to test whether a δ-square at t_1 is completely inside the reachable

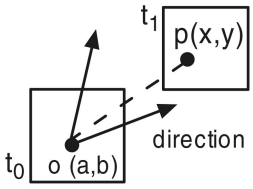


Fig. 12. Reachable area.

area of t_0 when the direction is known in the range of (β_l, β_h). This is a general case for Fig. 11b. The idea is to take an analytic view of this area. As is illustrated in Fig. 12, let o with coordinates (a, b) denote a point in the δ-square of t_0, and let p with coordinates (x, y) denote a point in the δ-square of t_1. Then, the condition that line $\overline{op}$ falls in between direction (β_l, β_h) is equivalent to the inequality $tg(\beta_l) \leq (y-b)/(x-a) \leq tg(\beta_h)$ (we consider only the first quadrant for simplicity). Therefore, to test whether the δ-square at t_1 ($x_l \leq x \leq x_h, y_l \leq y \leq y_h$) is completely inside the reachable area of t_0 is equivalent to testing whether any of the following two sets of inequalities with regard to x, y, a, b can be satisfied simultaneously:

$$\begin{cases} -(x-a)tg(\beta_l) + (y-b) \leq 0, \\ 0 \leq a \leq \delta, 0 \leq b \leq \delta, \\ x_l \leq x \leq x_h, y_l \leq y \leq y_h, \end{cases}$$

and

$$\begin{cases} (x-a)tg(\beta_h) - (y-b) \leq 0, \\ 0 \leq a \leq \delta, 0 \leq b \leq \delta, \\ x_l \leq x \leq x_h, y_l \leq y \leq y_h. \end{cases}$$

Either of them can be regarded as the set of linear constraints in a linear programming (LP) problem regarding variables x, y, a, b. We build two LP problems P_1, P_2 with a (dummy) objective function $C = 0$ and the same linear constraints as above. Determining whether any of the two sets of inequalities can be satisfied simultaneously is then equivalent to testing whether P_1 or P_2 has a feasible solution. The feasibility can be tested by any LP solver such as the classic Simplex or Ellipsoid method. The δ-square of t_1 is completely inside the reachable area of t_0 only if neither P_1 nor P_2 is feasible.

6.2 Minimum-Cost Update Strategy

In previous sections, we use a rectangular safe region to approximate the *ideal safe area* in which the change of the centroid p of a δ-square does not change the most probable result of any query. Fig. 13 illustrates the relation between a safe region and the ideal safe area. The gap between them is inevitable and could be arbitrarily large due to the following reasons: 1) a safe region for an individual query is already a rectangular approximation of the inner or outer region for this query and 2) the whole safe region is obtained by intersecting the safe regions for all individual queries, which makes it far smaller than the ideal safe area.

To guarantee 100 percent monitoring accuracy on the most probable result, the standard strategy updates whenever p moves out of the safe region, but this could be an unnecessary update as p might still be in the ideal safe area. We therefore believe that in applications where 100 percent accuracy is not mandatory and location update costs are serious issues, a strategy that can trade accuracy with costs is desirable. In this section, we develop such a strategy that tries to minimize the cost by adding a λ-rule to the standard strategy to filter out unnecessary updates.

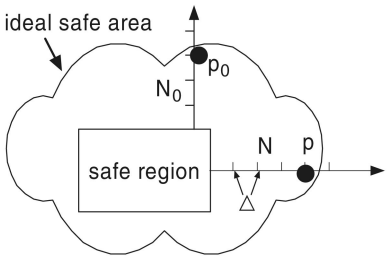


Fig. 13. Minimum-cost update strategy: λ-rule.

More specifically, symbol λ is the probability of p moving out of the ideal safe area. Let $cost_p$ denote the cost of not updating p in this case. As p causes a result change, $cost_p$ is essentially the penalty of loss of monitoring accuracy. On the other hand, let $cost_u$ denote the cost of updating p. Therefore, to minimize the expected cost, the λ-rule updates only if $cost_u < \lambda^* cost_p$, i.e., $\lambda > cost_u/cost_p$.

To test whether λ at p is larger than $cost_u/cost_p$ without sending it to the server, we need to know an additional point p_0 inside the ideal safe area (see Fig. 13). To find p_0, we continue to use the standard strategy. If the next updated location p^* causes no result changes (a feedback from the server), p^* is regarded as p_0. Otherwise, if p^* changes the result, the ideal safe area changes as well, so we continue to find p_0 for the new ideal safe area. In general, the λ-rule is only applicable after two consecutive location updates by the standard strategy, and the second update must cause no result changes from the first update. This prerequisite is useful in filtering out those ideal safe areas that are not significantly larger than their safe regions.

If we regard the space as a space of λ values, $\lambda = 0$ when p is in the safe region and gradually increases as p moves away from the safe region. As λ at any point is independent of the λ values at other points, the movement of λ from the border of the safe region to p can be regarded as a discrete random walk for simplicity (see Fig. 13). Initially, at the border $\lambda = 0$, and by taking steps of length Δ, it walks away from the border toward p. In each step, λ is increased by λ_Δ with a probability τ, and not increased with probability $1 - \tau$. As such, the total number of steps $N = dist(p, R)/\Delta$. Since the maximum value for λ is 1, $\lambda_\Delta = 1/N$. In any step, if the λ-rule is satisfied, the rule must also be satisfied at p, because λ is monotonously increasing as it moves. Thus, the strategy updates the location and stops the walk in any step when the λ-rule is satisfied. On the other hand, if the λ-rule is not satisfied till the last step, then the strategy does not update the location.

We are yet to estimate τ. As p_0 is known to be inside the ideal safe area, a random walk to p_0 can be conducted in the same way as above. By the *maximum likelihood estimation*, we should maximize the probability that λ at p_0 does not satisfy the λ-rule, i.e., the probability of $\lambda \leq cost_u/cost_p$. By the theory of Bernoulli process, λ at p_0 follows a Binomial

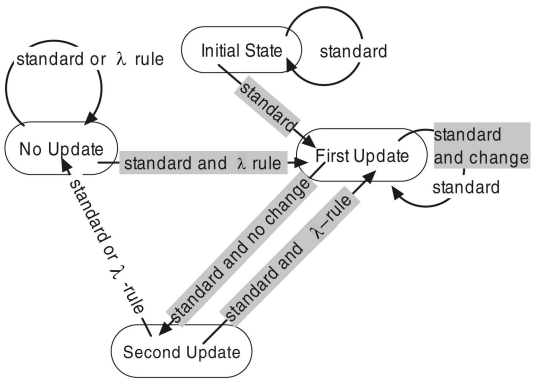


Fig. 14. State transition diagram.

distribution whose cumulative distribution function is bounded by $exp(-2\frac{(N_0\tau-\lfloor\lambda N\rfloor)^2}{N_0})$. The function reaches the maximum when $N_0\tau = \lfloor\lambda N\rfloor$. Putting $\lambda = cost_u/cost_p$, we have $\tau = \frac{\lfloor cost_u N/cost_p\rfloor}{N_0}$.

The state transition diagram of this strategy is illustrated in Fig. 14 where the shaded texts mean rule satisfaction and unshaded texts mean otherwise. There are four states in this strategy: "initial state," "no update," "first update," and "second update." The λ-rule is applicable only at "second update" and "no update" where p_0 is obtained. To transit to "second update," there must be two location updates, and the latter (regarded as p_0) must cause no result changes. Once the strategy decides to update, the λ-rule is suspended and the state is reset to "first update" to wait for the next p_0.

7 Performance Evaluation

To evaluate the monitoring performance, we implement a simulation test bed, where N moving objects move within a unit-square space [0..1, 0..1]. Each object detects its point location at frequency f, encapsulates it into a δ-square, and forwards the square to the location updater. Each object has an individual δ and it follows a normal distribution with mean value μ. We compare our PAM framework with two other frameworks, namely, the *optimal monitoring* (denoted as OPT) and the *periodic monitoring* (denoted as PRD). In optimal monitoring, every object has the perfect knowledge of the registered queries and the δ-squares of other moving objects at any time. Therefore, it knows precisely when the most probable result of any query changes, and only then does it send a location update to the server. OPT serves as the lower bound for all monitoring frameworks. In periodic monitoring, all objects periodically send out location updates simultaneously and the server reevaluates all registered queries based on these updates. Obviously, its monitoring accuracy and cost depend on the updating interval. In this paper, we test PRD with updating intervals 0.1 and 1, denoted as PRD(0.1) and PRD(1) hereafter.

7.1 Simulation Setup

In the simulation test bed, each object moves according to the random waypoint mobility model: the client chooses a random point in the space as its destination and moves to it at a speed randomly selected from the range $[0, 2\overline{v}]$; upon arrival or expiration of a *constant movement period* (randomly picked from the range $[0, 2\overline{t_v}]$), it chooses a new destination and repeats the same process. This is a well accepted and studied model in the mobile computing literature [5].

TABLE 1
Simulation Parameter Settings

Parameter	Default Value	Parameter	Default Value
N	100,000 objects	W	1,000 queries
q_{len}	0.005	k_{max}	10
f	0.001 per time unit	μ	0.0005
$\overline{v}$	0.01 per time unit	$\overline{t_v}$	0.005 time unit
M	50		

The workload consists of W queries, half of which are range queries and half are kNN queries. For range queries, the query rectangle is a square and its side length is uniformly distributed in a range of $[0.5q_{len}, 1.5q_{len}]$. For kNN queries, the query points are randomly distributed and k ranges from 1 to k_{max}. In all the three frameworks, the database server maintains an in-memory grid index ($M \times M$ cells) on the queries and an R*-tree index [2] on the objects. The database server is simulated on a Pentium 4 2.4 GHz PC with 1 GB RAM running WinXP SP2. Table 1 summarizes the default parameter settings.

To eliminate the effect from hardware configuration, the simulation uses logical time units instead of clock time units. Each simulation run lasts for 5,000 time units or until the measured value stabilizes (for those simulations that take 12 hours or more).

The performance metrics for comparison include:

- **Monitoring accuracy:** The monitoring accuracy at time t, $ma(t) \in \{0, 1\}$, is defined as whether the monitored results for all queries accord with the results from the OPT framework. Note that the latter are the most probable results based on the δ-squares of all objects at t. The monitoring accuracy for a time period $[t_b, t_e]$ is defined as the amortized accuracy over time, i.e., $ma(t_b, t_e) = \frac{1}{t_e - t_b}\int_{t_b}^{t_e} ma(t)dt$.
- **Wireless communication cost:** It is the amortized number of location updates sent by a moving object over time.
- **CPU time:** This is measured by the amortized server CPU time, which includes the time for query evaluation and safe region computation.

7.2 Validity of Most Probable Result

The first set of experiments is to validate the definition of most probable result. Under various μ (the mean of δ) and f (the location detection frequency), we compare the most probable result from the OPT framework with the genuine result (the result as if all the point locations were known) for all W queries. Fig. 15a shows the consistency rate, i.e., the proportion of time when the two results are the same. As μ or f increases, the consistency rate drops. However, the curve is not linear: the drop becomes slower when μ and f become larger. As such, even when μ or f is very large, the consistency rate is above 70 percent. This justifies our claim that the most probable result is a nice approximation of the genuine result for monitoring tasks.

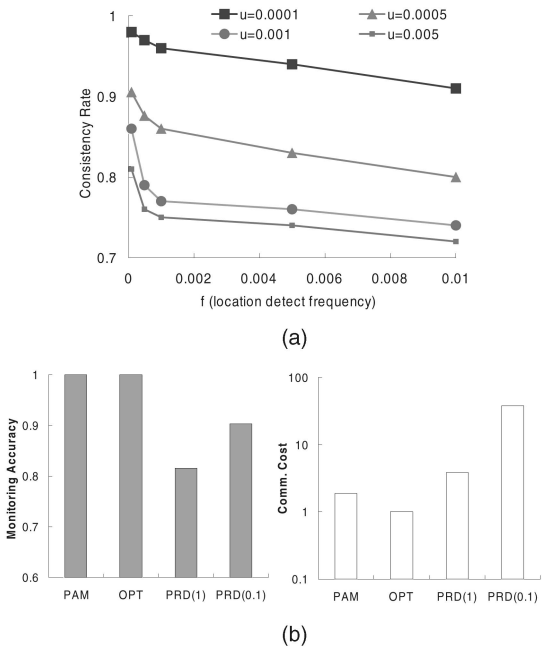


Fig. 15. Performance evaluation. (a) Consistency of most probable result. (b) The overall performance.

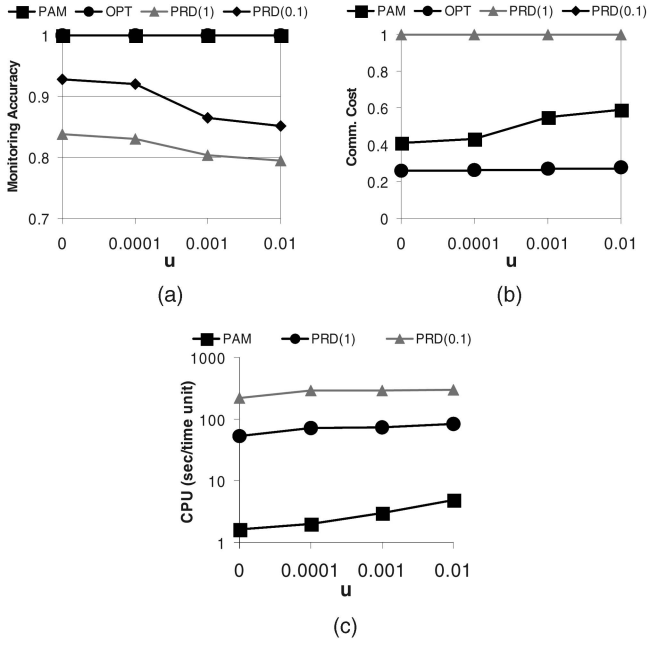


Fig. 16. δ versus monitoring accuracy, communication, and CPU cost. (a) Accuracy. (b) Communication cost. (c) CPU cost.

7.3 Overall Performance

The next set of experiments evaluates the overall performance of three frameworks with default parameter settings. Fig. 15b shows the monitoring accuracy and communication cost (normalized by the cost of OPT). As is guaranteed, our PAM framework achieves 100 percent accuracy, while PRD gets only 80-90 percent. Obviously, PRD(0.1) is more accurate than PRD(1) but the performance gap is less than 10 percent. Further, it is at the cost of 10 times higher communication overhead. On the other hand, the communication cost of PAM is much smaller than PRD and remains close to OPT.

7.4 Effects of δ

In this section, we evaluate the effect of δ on the performance. We vary μ (the mean of δ) from 0 to 0.01 and Fig. 16a shows the corresponding monitoring accuracy. While PAM achieves 100 percent accuracy, the accuracy of PRD(1) and PRD(0.1) drops significantly as μ increases. The drop is mainly caused by the increasing spatial vagueness introduced by the δ-square. However, the rate of the drop decreases as μ increases, which, in turn, verifies the fact that the most probable result is stable for even large δ-squares.

Fig. 16b shows the communication cost. The cost for OPT is almost the same for all settings, because the change of μ (and thus, δ) merely changes the query results, not necessarily the frequency of result changes. Similarly, PRD(1) and PRD(0.1) (not plotted) have constant costs of 1 and 10, respectively. On the other hand, the cost of PAM consistently grows as μ increases, but even for $\mu = 0.01$, which is already large in practice, it still outperforms PRD(1) by more than 40 percent. Furthermore, the rate of the increase drops as μ increases, which indicates that the approximation ratio of the safe region to the quarantine area becomes steady.

Fig. 16c shows the CPU cost of PRD and PAM. PAM increases faster than PRD(1) and PRD(0.1), because the increase for PAM arises from two aspects—more location updates and more complex query reevaluation (especially for kNN queries), whereas the increase for PRD arises only from the latter. Nonetheless, even for $\mu = 0.01$, PAM is still about $1/20$ that of PRD(1). Therefore, we can conclude that PAM is robust and efficient for various privacy settings.

7.5 Scalability

This section evaluates the scalability of all frameworks in terms of the server's CPU time and communication cost. Fig. 17a shows the CPU time when the number of registered queries (W) increases from 10 to 1,000. PAM only increases by less than 10 times because the grid-based query index filters out most of the unaffected and irrelevant queries. However, for PRD(1) and PRD(0.1), the CPU time is linear to W, as they need to reevaluate every query at each batch of location updates. When $W = 1{,}000$, for one logical time unit, the server needs 1.6 CPU seconds to monitor the 100,000 moving objects using PAM, 53 seconds using PRD(1), and 217 seconds using PRD(0.1). As PRD updates locations periodically, the high CPU cost imposes on it a

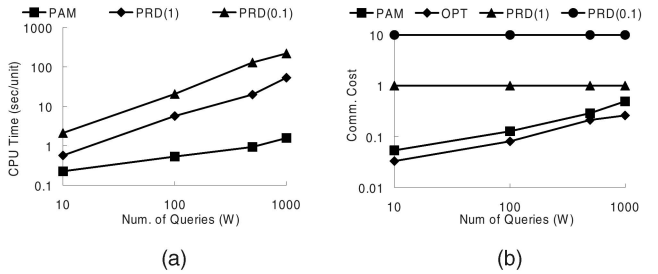


Fig. 17. Performance versus query numbers (W). (a) CPU time. (b) Communication cost.

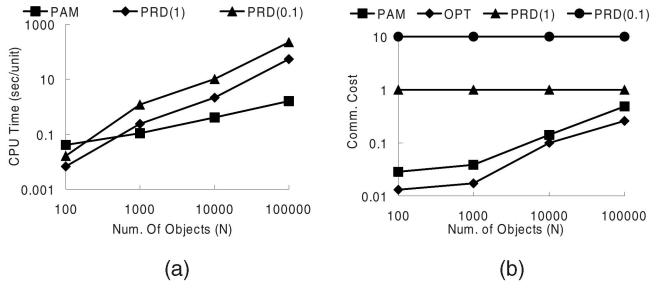


Fig. 18. Performance versus object numbers (N). (a) CPU time. (b) Communication cost.

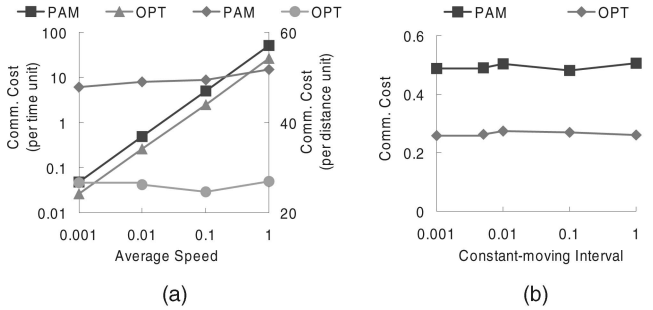


Fig. 20. Communication cost versus $\overline{v}$ and $\overline{t_v}$. (a) $\overline{v}$. (b) $\overline{t_v}$.

maximum update frequency: in this example, the update frequency is at most once every 21.7 seconds. PAM has no such limitation. In terms of communication cost, although PAM increases linearly with respect to W, it is still less than double of OPT. All the above results suggest that PAM is robust under various W settings.

Similarly, we conduct simulations to vary the number of objects (N) from 100 to 100,000. Fig. 18a shows that the CPU cost only increases by about 40 times when N increases by 1,000 times, due to the R*-tree index which is incrementally maintained. In contrast, PRD(1) and PRD(0.1) both increase at least linearly to N, as they need to build a new R*-tree at each update to reevaluate all queries. Similarly, Fig. 18b shows that the communication cost of PAM only increases by about 200 times when N increases by 1,000 times. This suggests that although a denser object distribution makes safe regions shrink, only a decreasing portion of objects affects the quarantine area of any query, and hence, the safe region of any object. In summary, PAM is more scalable than PRD in terms of CPU and communication cost.

7.6 Effects of Query Types

In this section, we study the performance of PAM on range and kNN queries separately. We vary the average query length q_{len} of range queries and k_{max}—the maximum k of kNN queries. The communication costs are plotted in Figs. 19a and 19b, respectively. We observe that for any parameter setting, PAM's communication cost is at most three times as much as that of OPT. For range queries, as q_{len} increases, the communication cost of OPT always increases at a steady pace. However, the communication cost of PAM increases more slowly when q_{len} is relatively small (at 0.001) or large (≥ 0.01). This can be explained by the fact that when q_{len} is relatively small or large, the safe regions are determined more by the home cell than by the *relevant queries*. Since the size of a cell is fixed, the cost tends to saturate. On the other hand, for kNN queries, as k_{max} increases, the costs of both OPT and PAM grow steadily. Even so, PAM manages to narrow the gap when k_{max} becomes larger. This suggests that for a heavy workload when results change frequently, the safe region achieves even better approximation to the ideal safe area.

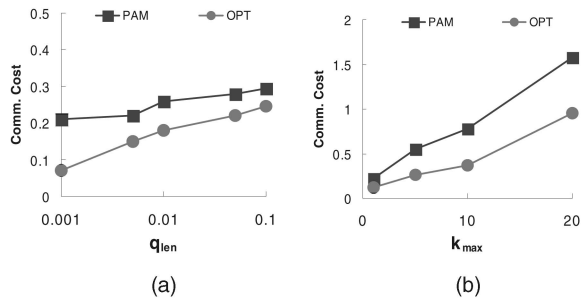


Fig. 19. Performance versus query types. (a) Range queries. (b) kNN queries.

7.7 Sensitivity of PAM

In this section, we study the sensitivity of PAM to other influential factors, namely, the average moving speed ($\overline{v}$) and the average constant movement period ($\overline{t_v}$) for the moving objects. Fig. 20a shows the communication cost when $\overline{v}$ varies from 0.001 to 1 per logical time unit. The costs of both PAM and OPT increase linearly as $\overline{v}$ increases, because the time of an object staying in a safe region is inversely proportional to $\overline{v}$. To eliminate this effect, we also plot the communication cost per distance unit on the secondary y-axis in the same figure, and observe that this cost is independent of $\overline{v}$. In other words, the update cost of PAM is not heavily dependent on the speed of object movement on a trajectory, but on the length of the trajectory. The CPU time shows a similar trend, and hence, is not plotted. In Fig. 20b, we also vary $\overline{t_v}$ from 0.001 to 1 time unit and find that it has little effect on the performance of PAM. As such, we conclude that PAM is robust to various moving parameters.

The next influential factor is the $M \times M$ grid partitioning of the query index. We vary M from 5 to 100 and plot both the communication cost and CPU time in Fig. 21. The larger is the value of M, the smaller is the grid cell size. The communication cost increases monotonously with M because the grid cell sets the largest possible safe region of an object. Nonetheless, the cost difference between $M = 5$ and $M = 50$ is not significant but there is a sharp increase from $M = 50$ to $M = 100$. The explanation is the same as in

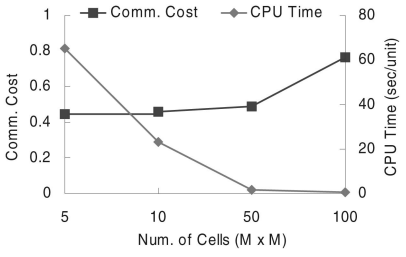


Fig. 21. Performance versus grid partitioning.

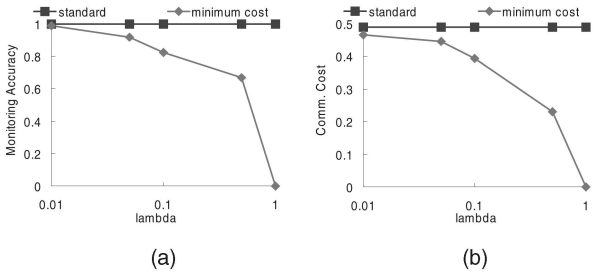


Fig. 22. Minimum cost strategy versus standard strategy. (a) Accuracy. (b) Communication cost.

Section 7.6, which is, when M is small and moderate, the safe regions are determined more by the relevant queries than by the grid cell, but when M is large, the cell becomes dominate. On the other hand, the CPU time decreases monotonously because the number of relevant queries in the cell decreases, and hence, the safe region computation is faster. In this figure, $M = 50$ yields a fairly low communication cost as well as a fairly low CPU time. From this experiment, we can see that it is advantageous to adapt the cell size to the server's workload: we first use a large M to partition the grid, and later if the workload turns out to be low, we enlarge the cell size by merging the cell where the update occurs with its neighboring cells within a certain distance. In this way, we can take full advantage of the CPU resource and reduce the communication cost (by enlarging the safe region) as much as possible.

7.8 Dynamic Update Strategy

The last set of experiments is conducted to evaluate the minimum cost update strategy. We vary the threshold for the λ-rule, i.e., $cost_u/cost_p$ from 0.01 to 1. Figs. 22a and 22b show the monitoring accuracy and communication cost in comparison with the standard strategy. The two curves of PAM show a similar trend as λ increases, which means that through λ, the strategy effectively trades accuracy for communication cost, or vice versa. Interestingly, when $\lambda \leq 0.1$, the decrease of the communication cost is accompanied by almost the same decrease of accuracy; however, when $\lambda > 0.1$, the accuracy drops more slowly than the communication cost. This shows that most ideal safe area is far larger than the safe region, so even an aggressive λ can still keep the object inside the ideal safe area.

8 Conclusions

This paper proposes a framework for monitoring continuous spatial queries over moving objects. The framework is the first to holistically address the issue of location updating with regard to monitoring accuracy, efficiency, and privacy. We provide detailed algorithms for query evaluation/reevaluation and safe region computation in this framework. We also devise three-client update strategies that optimize accuracy, privacy, and efficiency, respectively. The performance of our framework is evaluated through a series of experiments. The results show that it substantially outperforms periodic monitoring in terms of accuracy and CPU cost while achieving a close-to-optimal communication cost. Furthermore, the framework is robust and scales well with various parameter settings, such as privacy requirement, moving speed, and the number of queries and moving objects.

As for future work, we plan to incorporate other types of queries into the framework, such as spatial joins and aggregate queries. We also plan to further optimize the performance of the framework. In particular, the minimum-cost update strategy shows that the safe region is a crude approximation of the ideal safe area, mainly because we separately optimize the safe region for each query, but not globally. A possible solution is to sequentially optimize the queries but maintain the safe region accumulated by the queries optimized so far. Then, the optimal safe region for each query should depend not only on the query, but also on the accumulated safe region.

Acknowledgments

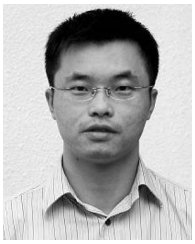
This work was supported by the Research Grants Council, Hong Kong SAR, China under Project No. HKBU211206, HKBU211307, HKBU210808, HKBU1/05C, HKBU/FRG08-09/II-48, RGC GRF 615806, and CA05/06.EG03.

References

[1] S. Babu and J. Widom, "Continuous Queries over Data Streams," *Proc. ACM SIGMOD,* 2001.
[2] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD,* pp. 322-331, 1990.
[3] R. Benetis, C.S. Jensen, G. Karciauskas, and S. Saltenis, "Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects," *Proc. Int'l Database Eng. and Applications Symp. (IDEAS),* 2002.
[4] A. Beresford and F. Stajano, "Location Privacy in Pervasive Computing," *IEEE Pervasive Computing,* vol. 2, no. 1, pp. 46-55, Jan.-Mar. 2003.
[5] J. Broch, D.A. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols," *Proc. ACM/IEEE MobiCom,* pp. 85-97, 1998.
[6] Y. Cai, K.A. Hua, and G. Cao, "Processing Range-Monitoring Queries on Heterogeneous Mobile Objects," *Proc. IEEE Int'l Conf. Mobile Data Management (MDM),* 2004.
[7] J. Chen and R. Cheng, "Efficient Evaluation of Imprecise Location-Dependent Queries," *Proc. IEEE Int'l Conf. Data Eng. (ICDE),* pp. 586-595, 2007.
[8] J. Chen, D. DeWitt, F. Tian, and Y. Wang, "NiagaraCQ: A Scalable Continuous Query System for Internet Databases," *Proc. ACM SIGMOD,* 2000.
[9] R. Cheng, D.V. Kalashnikov, and S. Prabhakar, "Querying Imprecise Data in Moving Object Environments," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 9, pp. 1112- 1127, Sept. 2004.
[10] H.D. Chon, D. Agrawal, and A.E. Abbadi, "Range and kNN Query Processing for Moving Objects in Grid Model," *ACM/Kluwer MONET,* vol. 8, no. 4, pp. 401-412, 2003.
[11] C.-Y. Chow, M.F. Mokbel, and X. Liu, "A Peer-to-Peer Spatial Cloaking Algorithm for Anonymous Location-Based Services," *Proc. ACM Int'l Symp. Geographic Information Systems (GIS),* pp. 171-178, 2006.
[12] D. Pfoser and C.S. Jensen, "Capturing the Uncertainty of Moving-Objects Representations," *Proc. Int'l Conf. Scientific and Statistical Database Management (SSDBM),* 1999.
[13] B. Gedik and L. Liu, "MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System," *Proc. Int'l Conf. Extending DataBase Technology (EDBT),* 2004.
[14] B. Gedik and L. Liu, "Location Privacy in Mobile Systems: A Personalized Anonymization Model," *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS),* pp. 620-629, 2005.

[15] B. Gedik and L. Liu, "Protecting Location Privacy with Personalized k-Anonymity: Architecture and Algorithms," *IEEE Trans. Mobile Computing,* vol. 7, no. 1, pp. 1-18, Jan. 2008.
[16] G. Ghinita, P. Kalnis, and S. Skiadopoulos, "Mobihide: A Mobile Peer-to-Peer System for Anonymous Location-Based Queries," *Proc. Int'l Symp. Spatial and Temporal Databases (SSTD),* 2007.
[17] G. Ghinita, P. Kalnis, and S. Skiadopoulos, "Prive: Anonymous Location-Based Queries in Distributed Mobile Systems," *Proc. Int'l World Wide Web Conf. (WWW '07),* pp. 371-380, 2007.
[18] M. Gruteser and D. Grunwald, "Anonymous Usage of Location-Based Services through Spatial and Temporal Cloaking," *Proc. MobiSys,* 2003.
[19] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD,* 1984.
[20] G.R. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," *ACM Trans. Database Systems,* vol. 24, no. 2, pp. 265-318, 1999.
[21] H. Hu, J. Xu, and D.L. Lee, "A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects," *Proc. ACM SIGMOD,* pp. 479-490, 2005.
[22] G. Iwerks, H. Samet, and K. Smith, "Continuous k-Nearest Neighbor Queries for Continuously Moving Points with Updates," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2003.
[23] G.S. Iwerks, H. Samet, and K. Smith, "Maintenance of Spatial Semijoin Queries on Moving Points," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2004.
[24] C.S. Jensen, D. Lin, and B.C. Ooi, "Query and Update Efficient B+-Tree Based Indexing of Moving Objects," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2004.
[25] D.V. Kalashnikov, S. Prabhakar, and S.E. Hambrusch, "Main Memory Evaluation of Monitoring Queries over Moving Objects," *Distributed Parallel Databases,* vol. 15, no. 2, pp. 117-135, 2004.
[26] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias, "Preventing Location-Based Identity Inference in Anonymous Spatial Queries," *IEEE Trans. Knowledge and Data Eng.,* vol. 19, no. 12, pp. 1719-1733, Dec. 2007.
[27] A. Khoshgozaran and C. Shahabi, "Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy," *Proc. Int'l Symp. Spatial and Temporal Databases (SSTD),* 2007.
[28] H. Kido, Y. Yanagisawa, and T. Satoh, "An Anonymous Communication Technique Using Dummies for Location-Based Services," *Proc. Second Int'l Conf. Pervasive Services (ICPS),* 2005.
[29] M.-L. Lee, W. Hsu, C.S. Jensen, B. Cui, and K.L. Teo, "Supporting Frequent Updates in R-Trees: A Bottom-Up Approach," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2003.
[30] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks," *Proc. USENIX Symp. Operating Systems Design and Implementation (OSDI),* 2002.
[31] M.F. Mokbel, C.-Y. Chow, and W.G. Aref, "The New Casper: Query Processing for Location Services without Compromising Privacy," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 763-774, 2006.
[32] M.F. Mokbel, X. Xiong, and W.G. Aref, "SINA: Scalable Incremental Processing of Continuous Queries in Spatio-Temporal Databases," *Proc. ACM SIGMOD,* 2004.
[33] G. Myles, A. Friday, and N. Davies, "Preserving Privacy in Environments with Location-Based Applications," *Pervasive Computing,* vol. 2, no. 1, pp. 56-64, 2003.
[34] J.M. Patel, Y. Chen, and V.P. Chakka, "STRIPES: An Efficient Index for Predicted Trajectories," *Proc. ACM SIGMOD,* 2004.
[35] S. Prabhakar, Y. Xia, D.V. Kalashnikov, W.G. Aref, and S.E. Hambrusch, "Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects," *IEEE Trans. Computers,* vol. 51, no. 10, pp. 1124-1140, Oct. 2002.
[36] K. Raptopoulou, A. Papadopoulos, and Y. Manolopoulos, "Fast Nearest-Neighbor Query Processing in Moving Object Databases," *GeoInfomatica,* vol. 7, no. 2, pp. 113-137, 2003.
[37] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest Neighbor Queries," *Proc. ACM SIGMOD,* 1995.
[38] S. Saltenis, C.S. Jensen, S.T. Leutenegger, and M.A. Lopez, "Indexing the Positions of Continuously Moving Objects," *Proc. ACM SIGMOD,* 2000.
[39] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu, "Prediction and Indexing of Moving Objects with Unknown Motion Patterns," *Proc. ACM SIGMOD,* 2004.
[40] Y. Tao, D. Papadias, and Q. Shen, "Continuous Nearest Neighbor Search," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2002.
[41] Y. Tao, D. Papadias, and J. Sun, "The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2003.
[42] W. Wu, W. Guo, and K.-L. Tan, "Distributed Processing of Moving k-Nearest-Neighbor Query on Moving Objects," *Proc. IEEE Int'l Conf. Data Eng. (ICDE),* 2007.
[43] X. Xiong, M.F. Mokbel, and W.G. Aref, "SEA-CNN: Scalable Processing of Continuous k-Nearest Neighbor Queries in Spatio-Temporal Databases," *Proc. IEEE Int'l Conf. Data Eng. (ICDE),* 2005.
[44] J. Xu, X. Tang, and D.L. Lee, "Performance Analysis of Location-Dependent Cache Invalidation Schemes for Mobile Environments," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 2, pp. 474-488, Mar./Apr. 2003.
[45] M.L. Yiu, C.S. Jensen, X. Huang, and H. Lu, "Spacetwist: Managing the Trade Offs among Location Privacy, Query Performance, and Query Accuracy in Mobile Services," *Proc. IEEE Int'l Conf. Data Eng. (ICDE '08),* 2008.
[46] T.-H. You, W.-C. Peng, and W.-C. Lee, "Protect Moving Trajectories with Dummies," *Proc. Int'l Workshop Privacy-Aware Location-Based Mobile Services,* 2007.
[47] X. Yu, K.Q. Pu, and N. Koudas, "Monitoring k-Nearest Neighbor Queries over Moving Objects," *Proc. IEEE Int'l Conf. Data Eng. (ICDE),* 2005.
[48] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D.L. Lee, "Location-Based Spatial Queries," *Proc. ACM SIGMOD,* 2003.

Haibo Hu received the PhD degree in computer science from the Hong Kong University of Science and Technology (HKUST) in 2005. He is an assistant professor in the Department of Computer Science, Hong Kong Baptist University (HKBU). Prior to this, he held several research and teaching posts at HKUST and HKBU. His research interests include mobile and wireless data management, location-based services, and privacy-aware computing. He has published 20 research papers in international conferences, journals, and book chapters. He is also the recipient of many awards, including the ACM Best PhD Paper Award and Microsoft Imagine Cup.



Jianliang Xu received the BEng degree in computer science and engineering from Zhejiang University, Hangzhou, China, in 1998, and the PhD degree in computer science from the Hong Kong University of Science and Technology in 2002. He is an associate professor in the Department of Computer Science, Hong Kong Baptist University. He was a visiting scholar in the Department of Computer Science and Engineering, Pennsylvania State University, University Park. His research interests include data management, mobile/pervasive computing, wireless sensor networks, and distributed systems. He has published more than 70 technical papers in these areas, most of which appeared in prestigious journals and conference proceedings. He currently serves as a vice chairman of ACM Hong Kong Chapter. He is a senior member of the IEEE.

Dik Lun Lee received the BSc degree in electronics from the Chinese University of Hong Kong, and the MS and PhD degrees in computer science from the University of Toronto, Canada. He is currently a professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He was an associate professor in the Department of Computer Science and Engineering, Ohio State University, Columbus. He was the founding conference chair for the International Conference on Mobile Data Management and served as the chairman of the ACM Hong Kong Chapter in 1997. His research interests include information retrieval, search engines, mobile computing, and pervasive computing.