Hsueh YL, Zimmermann R, Ku WS. Efficient location updates for continuous queries over moving objects. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 25(3): 415–430 May 2010

Efficient Location Updates for Continuous Queries over Moving Objects

Yu-Ling Hsueh[1] (薛幼苓), Roger Zimmermann[2], *Member*, *ACM*, *Senior Member*, *IEEE*
and Wei-Shinn Ku[3] (顾维信), *Member*, *ACM*, *IEEE*

[1]*Department of Computer Science, University of Southern California, Los Angeles, LA 90089-0781, U.S.A.*

[2]*Department of Computer Science, National University of Singapore, 117417 Singapore*

[3]*Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849, U.S.A.*

E-mail: hsueh@usc.edu; rogerz@comp.nus.edu.sg; weishinn@auburn.edu

Received November 3, 2009; revised December 9, 2009.

Abstract The significant overhead related to frequent location updates from moving objects often results in poor performance. As most of the location updates do not affect the query results, the network bandwidth and the battery life of moving objects are wasted. Existing solutions propose lazy updates, but such techniques generally avoid only a small fraction of all unnecessary location updates because of their basic approach (e.g., safe regions, time or distance thresholds). Furthermore, most prior work focuses on a simplified scenario where queries are either static or rarely change their positions. In this study, two novel efficient location update strategies are proposed in a trajectory movement model and an arbitrary movement model, respectively. The first strategy for a trajectory movement environment is the *Adaptive Safe Region* (ASR) technique that retrieves an adjustable safe region which is continuously reconciled with the surrounding dynamic queries. The communication overhead is reduced in a highly dynamic environment where both queries and data objects change their positions frequently. In addition, we design a framework that supports multiple query types (e.g., range and c-kNN queries). In this framework, our query re-evaluation algorithms take advantage of ASRs and issue location probes only to the affected data objects, without flooding the system with many unnecessary location update requests. The second proposed strategy for an arbitrary movement environment is the *Partition-based Lazy Update* (PLU, for short) algorithm that elevates this idea further by adopting Location Information Tables (LITs) which (a) allow each moving object to estimate possible query movements and issue a location update only when it may affect any query results and (b) enable smart server probing that results in fewer messages. We first define the data structure of an LIT which is essentially packed with a set of surrounding query locations across the terrain and discuss the mobile-side and server-side processes in correspondence to the utilization of LITs. Simulation results confirm that both the ASR and PLU concepts improve scalability and efficiency over existing methods.

Keywords location updates, continuous queries, location-based services

1 Introduction

The impressive advancement of mobile communication technologies, such as IEEE 802.11 and cellular networks, together with ever more capable handheld devices with GPS sensors has sparked intense interest in location-aware services. The efficient evaluation of *continuous spatial queries* is a fundamental capability needed in many practical applications. An example range query launched from a fire engine while battling flames might be to "continuously locate other fire engines within two miles of my current location." Since all units (i.e., users) are constantly moving, frequent location updates often result in high server re-indexing costs and immense communication overhead. With the mobility introduced by portable and handheld devices, the performance bottleneck for continuous spatial query processing is often concentrated in the handling of the frequent location updates at the server and the utilization of the communication channel between the moving client objects (also called *mobiles*) and the server. Wireless bandwidth is generally still much more scarce than wired bandwidth and — adding to the challenge — the movement dynamics of such an environment require frequent mobile-server message exchanges that contain location information for the database engine to maintain an up-to-date view of the world.

Many existing techniques[1-5] have proposed

Regular Paper
This work is supported by NSF of USA under Grant Nos. IIS-0534761, CNS-0831502, CNS-0855251, and NUS AcRF under Grant No. WBS R-252-050-280-101/133.

continuous monitoring approaches without considering the cost of the communication overhead involved. Some prior work[6-8] has provided significant insight into these issues by assuming a set of computationally capable moving objects that cache query-aware information (e.g., thresholds or safe regions) and locally determine a mobile-initiated location update. In the simplest case, whenever an object moves it sends its new location to the server. Obviously this can be very wasteful, for example if the moving object is located in an area where it does not affect any query results. Making informed decisions when to communicate update messages becomes a key design issue to improve scalability. The message count can be reduced through the following optimizations. The mobile client may be equipped with computation capabilities to maintain a *safe region*[9] with the purpose that movements within the safe region will not affect any query results (hence no location updates must be sent to the server). Safe regions are bounded by the nearest query rectangles around a mobile client and must be recomputed when certain events take place such as a new query is inserted or a moving object moves beyond its safe region boundary. In some cases (e.g., query insertion) a moving object is initially unaware of the event and the server must *probe* its current location. However, the focus of these solutions is mainly on static queries or simple types of queries (e.g., range queries). Furthermore, because of the usually simple shape of safe regions (e.g., rectangles or spheres) they can only help to avoid a fraction of unnecessary location updates. If query movements are frequent, such systems suffer from repeated location detections to resolve location ambiguity (incurred on the objects that might become result points) and numerous down-link messages sent to refresh the query-aware information on those mobile objects.

In this paper, two novel efficient location update strategies are proposed for a trajectory movement model and an arbitrary movement model, respectively. Based on the nature of the object movement, we first present the ASR approach which utilizes adaptive safe regions to reduce the downlink messages of location probes due to query movements for a trajectory movement model. To further reduce the downlink messages, the ASR approach only probes a set of objects that might become part of the query results. Additionally, ASR allows for decoupled, query-aware information locally maintained by each moving object until the movement affects the query results. In an arbitrary movement environment, since moving objects can freely move to any positions, the ASR can no longer efficiently handle query requests. We propose another strategy termed PLU (*Partition-based Lazy Update*) algorithm. Its main contribution lies in the development of "smarter" safe regions represented via location information tables that enable enhanced (i.e., more independent) mobile-side decision making for location updates. We provide a comprehensive study and present the details of the algorithms in the following sections. The remainder of this paper is organized as follows. Section 2 describes the background and related work. Section 3 provides the system overview and assumptions. Section 4 and Section 5 provide the details of the ASR and PLU algorithms, respectively. Finally, we conclude in Section 6.

2 Related Work

Cai *et al.*[10] proposed the *Monitoring Query Management* (MQM) approach to leverage the computational capabilities of moving objects for efficient processing of continuous range queries. SINA[2] has been introduced as centralized solution to process continuous range and k nearest neighbor (kNN) queries over moving objects. Yu *et al.*[11] proposed an algorithm that computes the query results by defining a search region based on the maximum distance between the query point and the current locations of previous kNNs. However, the algorithm results in high re-computation costs when the query point is highly dynamic. Similarly, Xiong *et al.*[5] suggested the SEA-CNN framework which uses the concept of shared execution. SEA-CNN continuously maintains the *search radius* of the query point to avoid rebuilding the query result once the query point changes its location. As an enhancement, Mouratidis *et al.*[3] presented a technique called *CPM* that defines a conceptual partitioning of the space by organizing grid cells into rectangles. Location updates are handled only when objects fall into the vicinity of queries, hence improving system throughput.

In the moving object environments, the challenge of frequent updates issuing from moving objects has been addressed when the first continuous spatial queries were studied. The existing work has proposed different strategies to reduce locations updates and these approaches can be classified into the following categories.

2.1 Object Movement Prediction

Predicting the movement of objects (i.e., their motion functions or trajectory) has been used with R-tree-based structures (e.g., the *TPR*-tree[12] and its variants[4]) and B-tree-based structures (e.g., the B^x tree[1]). The most common motion function is a linear function and it describes an object's movement by $f(Y) = X_{\rm ref} + (t_{\rm cur} - t_{\rm ref})\boldsymbol{V}$, where $X_{\rm ref}$ is the reference position or the last updated position to the server and $\boldsymbol{V}$ is a velocity vector. However, the linear motion function severely limits the applicability, since in practice an object may have drastic motion patterns. Tao *et al.*[13] introduced a general framework for monitoring and indexing moving objects. A recursive motion function is proposed to support non-linear motion patterns. However, this method incurs extensive location updates due to the arbitrary movements of the moving objects. These techniques require location updates from the objects when the parameters (e.g., moving direction, or speed) of the motion function change.

2.2 Periodic (Time-Based) Updates

In order to handle arbitrary object movements, periodic (time-based) position updates are widely used[2-3,11,14]. However, with such a paradigm tree-based indices suffer from excessive node reconstructions when tracking object locations. Cheng *et al.*[15] proposed a time-based location update mechanism with low communication costs to improve the temporal data inconsistency for the relevant objects to queries. Data objects with significance to the correctness of query results are required to send location updates more frequently. The main drawback of these methods is that an object will repeatedly send location updates to the server when it is enclosed by a query, which consumes a large amount of bandwidth when the query density is high.

2.3 Safe-Region Updates

A number of pioneering techniques have been designed for processing of continuous queries over moving objects. Prabhakar *et al.*[9] first proposed two elementary techniques called *Query Indexing* and *Velocity Constrained Indexing* (VCI) and also introduced the important concept of safe regions. Subsequently, Hu *et al.*[6] proposed a generic framework to handle continuous queries by leveraging the concept of safe regions through which the location updates from mobile clients can be further reduced. However, these methods only address part of the mobility challenge since they are based on the assumption that queries are static. Nowadays, an extensive number of spatial applications require the capability to process moving objects in conjunction with dynamic continuous queries.

2.4 Threshold-Based Updates

A threshold-based algorithm is presented in [7] which assumes that moving objects have some computational capabilities and aims to minimize the network cost when handling c-kNN queries. A threshold is transmitted to each moving object and when its moving distance exceeds the threshold, the moving object issues an update. However, the system suffers from many downlink message transmissions for refreshing the thresholds of the entire moving object population due to frequent query movements. Cheng *et al.*[16] proposed a time-based location update mechanism to improve the temporal data inconsistency for the objects relevant to queries. Data objects with significance to the correctness of query results are required to send location updates more frequently. The main drawback of this method is that an object will repeatedly send location updates to the server when it is enclosed by a query region.

In contrast, our proposed techniques for efficient location updates aim to reduce the communication cost of dynamic queries over moving objects. The first strategy for a trajectory movement environment is the *Adaptive Safe Region* (ASR) technique that leverages the trajectory information and retrieves an adjustable safe region for each data object. The second mythology for an arbitrary movement model is the *partition-based lazy update* approach that significantly reduces unnecessary location updates by maintaining a *Location Information Table* (LIT) on each moving object. Because of the different movement models, the query-aware information (e.g., ASR vs. LIT) are formatted differently. These two techniques do not deteriorate when faced with high mobility rates as demonstrated by our simulation results and surpass the aforementioned solutions with higher scalability and lower communication cost.

3 System Overview and Assumptions

To enable a focused discussion we make some explicit assumptions. The communication between the centralized server and the mobile units is through cellular phone or WiMAX networks. A centralized server is assumed in the environment to process continuous queries. We assume an ideal network environment, that is, no communication delay between the server and moving objects. The mobile units such as vehicles or handheld devices (e.g., cell phones) consist of a set of dynamic query objects Q and a set of moving objects P. Both queries and moving objects are identified by a unique identifier to distinguish their types. The mobile units are able to provide the server with their positions from a GPS chip built into the devices and we assume that each mobile unit has enough computational capabilities and memory to carry out the required tasks. We assume no power constraints and virtually unlimited life time of devices. A main-memory grid G is used as the underlying structure to index moving objects because of its simplicity and ease-of-maintenance in a highly dynamic environment. For high performance an event-driven approach is adopted to evaluate continuous queries. To maintain the correctness of the query results, the server monitors registered query objects. Thus, the server can evaluate the queries based on their new locations. The details of the ASR and PLU approaches are described in Section 4 and Section 5, respectively.

4 Trajectory Movement Model for Moving Objects

We propose a framework to support multiple types of dynamic, continuous queries in the ASR approach. Our goal is to minimize the communication overhead in a highly dynamic environment where both queries and objects change their locations frequently. When a new query enters the system we leverage the trajectory information that it can provide by registering its starting and destination points as a movement segment for continuous monitoring. For example, a policeman might request the following query "send me the closest 5 police cars on the road as I am moving from point A to point B." For simplicity, we assume a straight movement segment between two points. This assumption can be easily extended to a more realistic scenario which may approximate a curved road segment with several straight-line sub-segments. We propose an adaptive safe region that reconciles the surrounding queries based on their movement trajectories such that the system can avoid unnecessary location probes to the objects in the vicinity (i.e., the ones which overlap with the current query region). Furthermore, our incremental result update mechanisms allow a query to issue location probes only to a minimum area where the query answers are guaranteed to be fulfilled. In particular, to lower the amortized communication cost for c-kNN queries, we obtain extra nearest neighbors (n more NNs) which are buffered and reused later to update the query results. Thus, the number of location updates incurred from the query region expansion due to query movement is reduced. An example is shown in Fig.1(a). The ASR of p_3 is determined based on the closest query q_1, since p_3 has a high probability of being covered by the query region of q_1 when q_1 moves in the future. The safe region of p_3 is adjusted to a reasonable size according to the segment information of q_1. The safe region for p_8 is simply set to the maximum non-overlapping area with the query region of q_2, because q_2 (due to its opposing moving direction) will never cover p_8. We buffer one extra NN for q_2 (a c-3NN query). When q_2 moves to q_2', and since the number of NNs is equal to 3, the query region remains unchanged. In the traditional approach (as shown in Fig.1(b)), the query region is expanded to cover p_5 (the first closest object outside the query region) such that the additional location probes to p_7, p_9, and p_{10} are issued. Therefore, our approach reduces the number of query expansions to find sufficient NNs and the number of location probes.

In this approach, each query object registers its movement trajectory with the server by uploading its starting and ending points (denoted by $\boldsymbol{q}_j = [q_j^s, q_j^e]$). Furthermore, all the data objects can move in a non-restricted fashion that allows them to move arbitrarily. The location updates of a query result point (result point for short) and a non-result point (data point for short) are handled with two different mechanisms. An Adaptive Safe Region (ASR) is computed for each data point. A mobile-initiated voluntary location update is issued when any data point moves out of its safe region. An example (p_8) is shown in Fig.1(a). To capture the possible movement of a result point, we use a Moving Region (MR) whose boundary increases by the maximum moving distance per time unit. For the result points, the location updates are requested only when the server sends server-initiated location probes triggered when the moving regions of the result points overlap with some query regions.

Fig.2 shows the system framework. When a request arrives from a data point (A) or from a query point (B) (e.g., a location update, insertion or deletion), the ASR

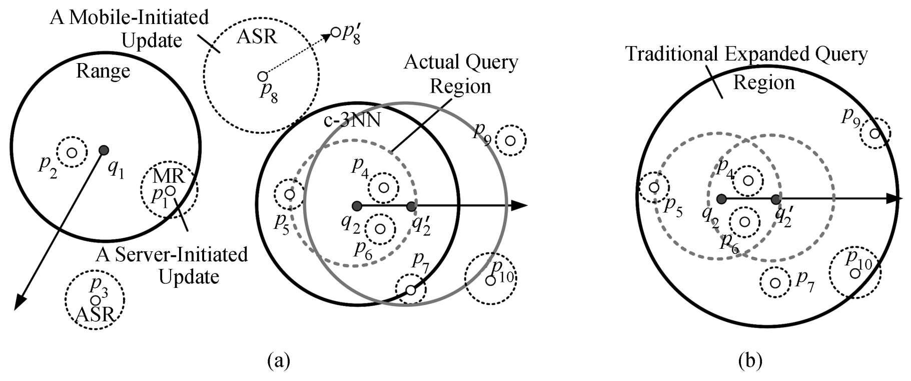


Fig.1. The overview of the ASR approach. (a) An example of ASRs. (b) A query expansion.

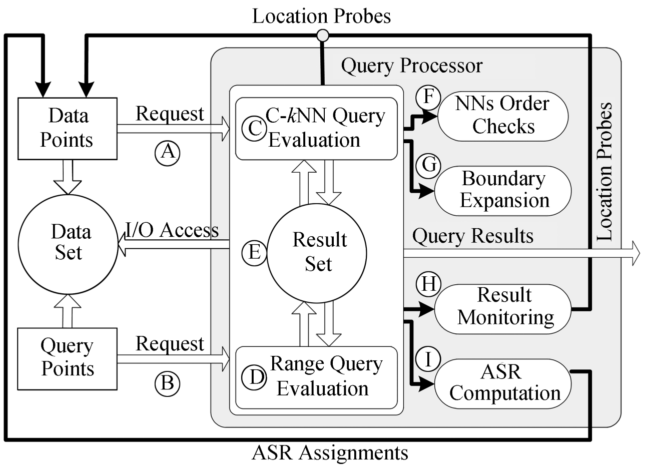


Fig.2. ASR approach overview.

Table 1. Symbols and Functions for the ASR Approach

Symbol	Description
Q	A set of query objects
P	A set of moving objects
G	A $w \times w$ object grid where objects are hashed to the grid cells based on their locations
δ	Maximum speed for any object
$p_i.ASR$	Adaptive safe region of object p_i
$p_i.MR$	Moving region of object p_i
$q_j.QR$	Query region of query q_j (the radius is denoted by $q_j.QR.radius$)
$\boldsymbol{q}_j$	Movement trajectory of q_j
q_j^s	Staring point of the movement trajectory for q_j
q_j^e	Ending point of the movement trajectory for q_j

query processor checks whether the point is part of a query result in modules (C) and (D). To incrementally update a query result, prior query results (E) are considered. For a c-kNN query, an NN order check (F) is performed during the query evaluation process. While there are less than k NNs in the result set, a query region expansion (G) is executed. Some server-initiated location probes might be needed to resolve location ambiguities. The points in the result set are monitored (H) through a passive mechanism — this result set is different from the non-result points that voluntarily issue location updates locally determined by the objects. Finally, an updated data point is assigned a new ASR based on the current query information in module (I). Detailed descriptions of the functionality of each component will be given in the following sections. Table 1 summarizes the symbols and functions we use throughout the following sections.

4.1 Adaptive Safe Region Computation

The existing work adopts safe regions to reduce unnecessary location updates such that the communication cost between the server and moving objects is reduced. A safe region in a traditional system is simply an area of maximal size around an object such that no query regions overlap. Fig.3(a) shows an example of two such safe region types (a safe sphere and a safe rectangle) for object p_1. However, this approach suffers from many location updates as a result of frequent query movements. When a query moves, the server initiates location probes to the data objects whose safe regions overlap with the query region to ensure the correctness of the query answers. In this paper, we propose a novel approach to retrieve an adaptive safe region (ASR), which is often smaller than a maximum non-overlapping region and yet is very effective in reducing the amortized communication cost in a highly dynamic mobile environment. The key observation lies in the consideration of some important factors (e.g., the velocity or orientation of the query objects) to reconcile the size of the safe regions. Fig.3(b) illustrates the concept of an ASR. The on-demand location probes are not issued as soon as any surrounding queries (q_1, q_2, or q_3) move. In this example, the distance z is the ASR radius of p_1, because in the worst case, after both q_3 and p_1 move by distance z and p_1 moves directly toward q_3, p_1 may become a result point of q_3. The following lemma establishes the ASR radius based on this observation.

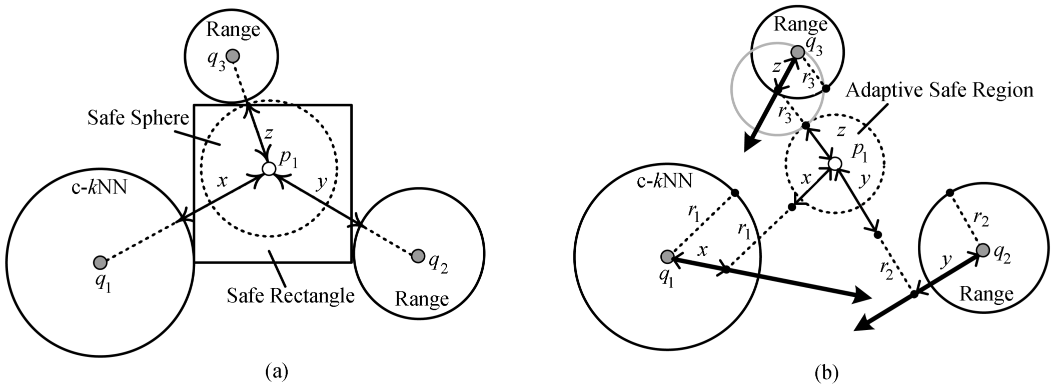


Fig.3. Traditional safe region vs. ASR. (a) A traditional safe region. (b) An adaptive safe region.

Lemma 1. *$p_i.ASR.radius = \min(CDist(p_i, q_j) - q_j.QR.radius)$, $\forall q_j \in Q$, where*

$$CDist(p_i, q_j) = \begin{cases} \overline{p_i f'} & \text{if } \theta_j \leqslant \frac{\pi}{2} \text{ and } \exists f', \text{ or} \\ \overline{p_i q_j^s} & \text{if } \theta_j > \frac{\pi}{2} \text{ or } \nexists f'. \end{cases}$$

As an illustration of Lemma 1 (and to explain the symbol notation), consider Fig.4, where the set of queries $Q = \{q_j, q_k\}$ are visited for retrieving the adaptive safe region (the dashed circle) of the data point p_i. We measure the Euclidian distance between a query and a data point (*CDist* in Lemma 1) and then deduct the query range. Lemma 1 captures two cases of *CDist*. The first case ($CDist(p_i, q_j)$) computes a distance $\overline{p_i f'} = \overline{q_j^s f}$ in the worst-case scenario where both p_i and q_j move toward each other (under the constraint of the maximum speed). f' represents the border point (on the border of $q_j.QR$ while q_j arrives at f on its movement segment), after which p_i would possibly enter the query region of q_j. f is the closest point to q_j^s on the trajectory of q_j, which satisfies the condition that the distance from p_i to f is equal to $\overline{p_i f'} + \overline{f' f}$, where $\overline{f' f} = q_j.QR.radius = r_j$. Let $\overline{p_i f'} = x$ for short. We can obtain the f and f' points by computing x first, which is considered the safe distance for p_i with respect to q_j. x can be easily computed with the trajectory information of q_j by solving the quadratic equation: $(x + r_j)^2 = h^2 + (\overline{q_j^s m} - x)^2$ (h is the height of triangle $\triangle p_i q_j^s m$). f on $\boldsymbol{q}_j$ exists only when θ_j ($\angle p_i q_j^s q_j^e$) is less or equal to $\frac{\pi}{2}$ and $(\overline{p_i q_j^e} - q_j.QR.radius) < \overline{q_j^s q_j^e}$ (triangle inequality). If the first case is not satisfied, we consider the second case ($CDist(p_i, q_k)$), which finds the maximum non-overlapping area with $q_j.QR$. Since $\theta > \frac{\pi}{2}$ in the second case, the query range of q_j can never cover p_i due to the opposing movement of q_j. In this example, the safe distance x (with respect to q_j) is smaller than y (with respect to q_k), so x is chosen as the radius of the adaptive safe region of p_i. In our system, since a c-kNN query can be considered an order-sensitive range query, we use the same principle to compute safe regions for each data object with respect to range queries and c-kNN queries. In case of a query insertion or query region expansion of a c-kNN query, the adaptive safe regions of the affected data objects must be reassigned according to current queries to avoid any missing location updates.

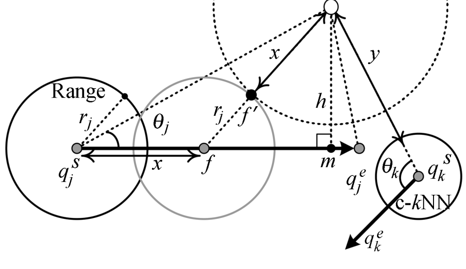


Fig.4. An adaptive safe region.

4.2 Query Evaluation with Location Probes

The initial query results of the range and c-kNN queries are obtained using *CPM*[6], and later the query results are updated in an event-driven fashion. Such events include the insertion or update of a query. In the following subsections, we propose our incremental query re-evaluation algorithms for both range and c-kNN queries. While updating the query answers, on-demand server-initiated location probes are issued whenever any location ambiguity exists. Specifically, the cost of updating c-kNN queries is usually higher than updating range queries. The reason is that a c-kNN search is an order-sensitive query. The system executes more location updates to ensure the order of the result points. Furthermore, to make sure that at least k result points are found for a c-kNN query, the query region often needs to be enlarged in a situation where both query and data objects are moving, which leads to more location probes. In our approach, the strategy to handle such increasing unnecessary location updates incurred from a c-kNN query is that the query processor computes $(k+n)$ NNs for a c-kNN query instead of evaluating exactly k NNs. This approach helps to reduce the number of future query region expansions to retrieve sufficient NNs for the queries. Since a c-kNN query is treated as an order-sensitive range query, we adopt the same principle that is used for a range query to find the new answer set in the current query regions first. A query region is expanded only when there are less than k NNs in the result set. Finally, an order-checking procedure is performed to examine the order of the result points and determine necessary location probes.

4.2.1 *Query Result Updates for Range Queries*

The query processor re-evaluates the range queries based on their current positions by the same principles as evaluating the initial query results. The traditional approach adopts the query region itself as the safe region for all the result points in the region to reduce the number of location updates. However, the approach incurs more network messages when a range query changes its position frequently, because the system needs to inform the result points of the new position of the query region to avoid missing location updates. An alternative approach basically monitors

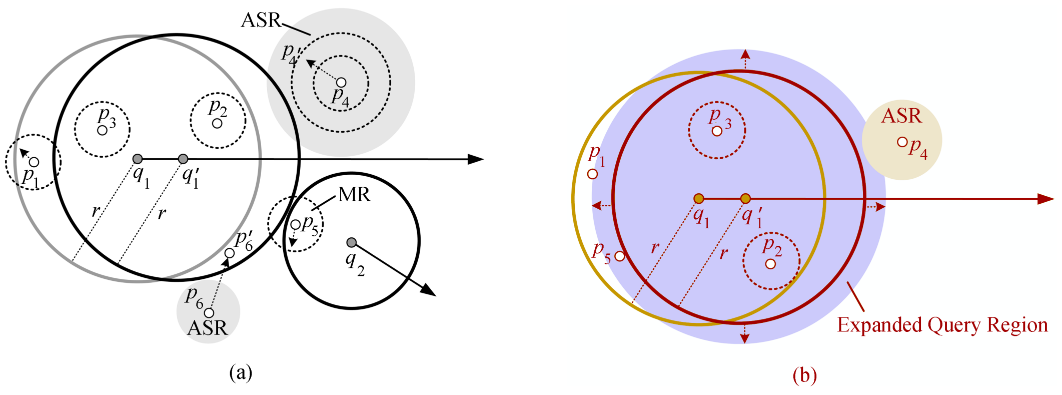


Fig.5. Query result updates in the ASR approach. (a) Result updates of a range query. (b) Result updates of a c-kNN query.

the entire set of result points to obtain the new correct results. However, such an approach is not scalable when there are large numbers of range queries. We use a Moving Region (MR) for each result point to estimate the possible movement at the server side. The query processor sends the on-demand location probes to those result points that might move out of the current query regions. An MR is indexed on the grid and the boundary increases at each time step by the maximum moving distance until the result point is probed by the server. Since the number of result points are relatively small, indexing MRs does not significantly increase the overall server workload. In Fig.5(a), when q_1 moves to q_1', the query processor checks p_1 and p_5, since their MRs intersect with $q_1'.QR$.

For a data point, in addition to its adaptive safe region, we also consider the current possible moving boundary to serve as an additional indicator for the server to determine a necessary location probe. Continuing the example in Fig.5(a), the gray circle surrounding p_4 is its ASR, and the dashed circles represent the possible moving boundaries (the radius is equal to the maximum moving distance since the last update of p_4) for different time steps. p_4 is checked because its $p_4.ASR$ overlaps with $q_1'.QR$. However, the server does not need to issue a location probe since the current moving boundary does not overlap with $q_1'.QR$. p_6' is a newly updated (p_6 moves out of its ASR) data point. The system also needs to check whether its current position is in the query region of q_1'. Algorithm 1 shows the pseudo code of the range query evaluation, where q_j' is the updated query of q_j. Lines 1~7 remove previous result points that are not in the current query region $q_j'.QR$. Lines 2 and 4 compute the *mindist* and *maxdist* between a query point and a result point, respectively. If a result point with an MR is completely contained in the query range, a location probe is ignored. In line 10, if p_i is a data point, the server uses the radius of ASR or the maximum moving distance since the last update, which ever is less to estimate its possible moving distance.

Algorithm 1. RangeQuery-Update(q_j')

```
1:  for (each $d \in q_j.RangeNN$) do
2:      if ($dist(d, q_j') - d.MR.radius) > q_j'.QR.radius$) then
3:          remove $d$
4:      else if ($dist(d, q_j') + d.MR.radius) > q_j'.QR.radius$)
        then
5:          probe $d$ and remove $d$ if its current position is
            outside of $q_j'.QR$
6:      end if
7:  end for
8:  for (each $c \in G$, which overlaps with the $q_j'.QR$) do
9:      for (each object $p_i$ which resides in $c$ or whose 1)
        ASR, or 2) MR overlaps with it) do
10:         let $r = p_i.MR.radius$, if $p_i$ is a result point; else
            let $r = \min(p_i.ASR.radius, \delta\Delta t)$
11:         if ($dist(p_i, q_j') - r < q_j'.QR.radius$) then
12:             if ($dist(p_i, q_j') + r < q_j'.QR.radius$), insert $p_i$ into
                $q_j'.RangeNN$
13:             else probe the position of $p_i$ and insert $p_i$ into
                $q_j'.RangeNN$, if $p_i$ is within $q_j'.QR$.
14:         end if
15:     end for
16: end for
```

4.2.2 *Query Result Updates for c-kNN Queries*

A c-kNN query is more complicated since it is order-sensitive. An intuitive solution enlarges a query region that covers at least all the previous result points (first k NNs) to retrieve new result points. This approach greatly increases the number of location updates since such an expansion (the query region is expanded by the moving distance of the query and result points) often results in more location probes, even though in reality only a small fraction of queries and data objects move. Therefore, in our design for the c-kNN queries, we propose a server-initiated update strategy with an event-triggered update mechanism. Furthermore, the query processor retrieves $(n + k)$ NNs to avoid immediate and successive query region expansions. We relax the definition of the query region, that is, a query region does not necessary include exact k NNs only. The query region remains unchanged until a c-kNN query does not contain enough NNs in the region. We summarize the following steps to update a c-kNN query result incrementally.

Step 1. Assume that q'_j is a c-kNN query after it moves from q_j position. Initially, set $q'_j.QR.radius = q_j.QR.radius$. Perform a range query update (as described in the previous subsection) to update result points in $q'_j.QR$. If the number of NNs in $q'_j.QR$ is equal to or larger than k, proceed to Step 3. Otherwise, continue to Step 2.

Step 2. Expand $q'_j.QR$ until there are $(k + n)$ NNs. Update $q'_j.QR.radius$ to the distance between q'_j to the $(k + n)$-th NN.

Step 3. Sort the order of the result points and issue the necessary location probes.

Step 1 ensures that $q'_j.QR$ covers at least k result points. Note that during the process, some discarded objects that are not in $q'_j.QR$ might be useful in Step 2, because these objects are often very close to $q'_j.QR$ and might be already probed by the server. Finding new NNs from these points first in Step 2 helps the query processor to avoid expanding the safe region to a farther level of cells. In Step 2, while expanding the query region to cover $(k + n)$ result points, a location update is required from any data object p_i whose safe region overlaps with the query region of q'_j. A new ASR is computed for the updated p_i, if p_i is still a data object. We use the same approach (query region expansion) to handle a query insertion. In Step 3, sorting the order of the result points does not require the current positions of the entire result points. The processor performs an *OrderCheck* procedure that examines the possible actual moving distance of two consecutive NNs to determine the order of the NNs, and issues a location probe only if there is a location ambiguity.

Fig.5(b) shows a query region expansion where $k = 3$ and $n = 1$. In Step 1, since p_1 and p_5 (probed during the process) are not in $q'_1.QR$, they are removed from the answer set and inserted into a buffer for "recycling" later. Step 2 is performed since there are only two result points in $q'_1.QR$. The query processor checks the data points in the buffer first, so the first two objects (sorted by the *mindist* to q'_1) are considered. The new $q'_1.QR.radius$ (the blue area) is set to the distance between q'_1 and p_1 to include at least 4 $(k + n)$ objects. p_4 is checked later since the safe region overlaps with $q'_1.QR$. Algorithm 2 shows the detailed process of a c-kNN query update. In line 2, the RangeQuery-Update procedure inserts the discarded objects into buffer B sorted by *mindist* in the ascending order. Line 4 computes the number (v) of NNs missing in the current query region. Line 12 executes *CPM* to further expand the query region by checking the surrounding cells only when the buffer is empty. The OrderCheck procedure in line 16 is performed after all the sufficient NNs are found. In the OrderCheck procedure, to determine a necessary location probe for kNN result points, we observe the following lemma. A proof of correctness is presented subsequently.

Algorithm 2. c-kNN-Update(q'_j)

```
1:  let $B = \phi$ be a buffer
2:  perform RangeQuery-Update($q'_j$), which finds new
    NNs in the current query region and inserts discarded
    objects into $B$, if any
3:  if ($q'_j.KNN.size < k$) then
4:      $v = k + n - q'_j.KNN.size$
5:      while ($v > 0$) do
6:          if ($B.size > 0$) then
7:              set $q'_j.QR.radius = dist(q'_j, V)$, where $V$ is the
                $v$-th NN in $B$, if $B.size \geqslant v$. Otherwise, set
                $dist(q'_j, L)$, where $L$ is the last object in $B$.
8:              empty $B$
9:              perform RangeQuery-Update($q'_j$) that inserts
                un-visited, discarded objects into $B$, if any
10:             $v = k + n - q'_j.KNN.size$
11:         else
12:             perform $CPM(q'_j)$ that checks the objects in
                the surrounding cells of $q'_j.QR$, until $(k + n)$
                objects are fulfilled, and terminate the loop.
13:         end if
14:     end while
15: end if
16: sort $q'_j.KNN$ by performing $OrderCheck(q'_j.KNN)$ that
    issues necessary location probes.
```

Lemma 2. *Let q'_j be the last reported position of the query object q_j, and let $\ell = \delta\Delta t$ be the maximum moving distance since the last update of q_j, where δ is the maximum speed and Δt is the time period from the last update time to the current time. $\forall i = 1$ to k, a result point p_i (the i-th result point sorted by the mindist to q'_j) needs to issue a location update when the following condition is satisfied:*

$$\ell \geqslant (mindist(q'_j, p_{i+1}) - mindist(q'_j, p_i)) \times \frac{1}{2}.$$

Proof. The proof is straightforward, since when the order of p_i and p_{i+1} changes, $mindist(p_i, q'_j) \geqslant mindist(p_{i+1}, q'_j)$. When considering the worst case that p_i moves in an opposing direction from q'_j and p_{i+1}

moves toward q'_j directly, the following inequality holds true:

$$mindist(p_i, q'_j) + \ell \geqslant mindist(p_{i+1}, q'_j) - \ell.$$

Therefore, we conclude that the order of p_i and p_{i+1} must change, when $\ell \geqslant (mindist(q'_j, p_{i+1}) - mindist(q'_j, p_i)) \times \frac{1}{2}$. It is necessary for the server to probe both locations of p_i and p_{i+1}. □

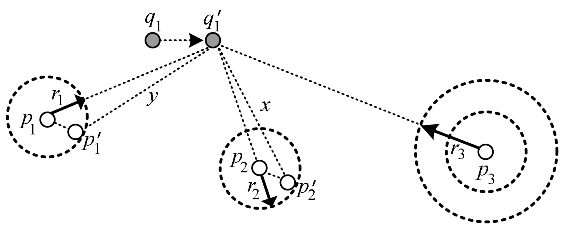


Fig.6. The order checks of a c-kNN query.

In Fig.6, the result set of q'_1 is $\{p_2, p_1, p_3\}$ sorted by the distance between q'_1 and their positions at the server since the last updates. The OrderCheck procedure first checks p_2 and p_1. Since $dist(q'_1, p_2) + r_2 > dist(q'_1, p_1) - r_1$, the order of p_2 and p_1 might need to be switched. The system needs to probe p_2 and p_1. After the location probes, the order of the NNs becomes $\{p'_1, p'_2, p_3\}$. Thus, the procedure checks the next pair of p'_2 and p_3. Since $dist(q'_1, p'_2) < dist(q'_1, p_3) - r_3$, the location probe of p_3 is not necessary.

4.3 Experimental Evaluation

We evaluated the performance of the proposed framework that utilizes ASRs and compared it with the traditional safe region approach[6,8] and a periodic update approach (PER). The periodic technique functions as a baseline algorithm where each object issues a location update (only uplink messages are issued in this approach) every time it moves to a new position. We extended the safe region approach (SR*) to handle dynamic range and c-kNN queries where the result points are monitored the same way as in ASR. We preserve the traditional safe region calculations (maximum non-overlapping area) for the SR* approach. The simulation steps and the detailed simulation results are described in the following subsections.

4.3.1 Simulation Steps

We use a main memory grid as the underlying index structure for all the three approaches. Our datasets are generated on a terrain service space of $[0, 1024]^2$. We assume a maximum speed for any moving object in the range of $[0.48, 1.25]$. The mobility (the percentage of objects that move from time step to time step) for the objects is set in a range from 10% to 50%. The length q_{len} of a range query is set in the range of [1, 10] and k for a kNN query is set from 5 to 20. In the simulations, the main measurement is the cost of the communication overhead which includes uplink messages (e.g., a mobile-initiated location update) and downlink messages (e.g., a server-initiated location probe). The communication cost is measured by assuming that the cost of an uplink message ($c_{\text{up}} = 2$) is twice as costly as a downlink message ($c_{\text{down}} = 1$). Table 2 summarizes the default parameter settings in the following simulations.

Table 2. Simulation Parameters for the ASR Approach

Parameter	Default	Range
Number of objects (P)	100 K	50 K, 100 K, 150 K, 200 K
Number of queries (Q)	100	50, 100, 150, 200
Mobility rate	50%	10%, 20%, 30%, 40%, 50%
Number of NNs (K)	10	5, 10, 15, 20
Query length for range queries (q_{len})	5	1, 5, 10

4.3.2 Number of Extra NNs

First, we test the efficiency of using extra NNs (n) for c-kNN queries by varying the number of n, since this factor greatly affects the number of downlink messages. The choice of the number of extra NNs is a trade-off. If n is too large, the query processor evaluates more NNs for a query and the system is more likely to issue more location probes since a larger query region might overlap with more data objects for location probes. If n is too small, there are more query expansions which might also cause location probes. Fig.7 shows the number of overall communication cost (measured in thousands of messages) as a function of the number of extra NNs ranging from 0 to 20. When n is set to bigger than 10, the performance of ASR is degraded in terms of the communication cost. Therefore, we chose $n = 10$ for the rest of our experiments as this setting results in reduced communication cost.

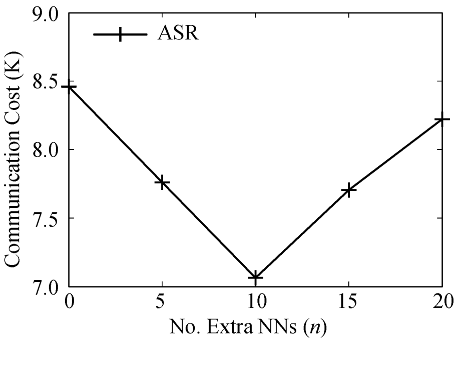


Fig.7. Extra NNs vs. communication cost.

4.3.3 Cardinality

We examined the effect of the query and object cardinality assuming that all query and object sets move with a mobility rate of 50%. Fig.8(a) shows the communication overhead of ASR, SR* and PER with respect to the object cardinality. ASR outperforms SR* and PER. The difference increases as the number of objects grows. Since an ASR reconciles the surrounding moving queries, a query movement does not incur many unnecessary location probes from the surrounding objects. SR* on the other hand, triggers many location probes from the objects whose safe regions overlap with a query region once the query changes its position. As the density of objects increases, there are more objects in the vicinity area of a query region. Hence SR* incurs an increasing number of location updates as the cardinality increases. Fig.8(b) shows the impact of the number of queries. Our algorithm achieves about 50% reduction compared with SR* and 90% reduction compared with PER.

4.3.4 Query Coverage

The query coverage varies with the number of queries, number of NNs (for kNN queries) and the query length (for range queries). Fig.9(a) shows the communication cost as a function of the number of NNs and Fig.9(b) illustrates the effect of the query length. Overall, the communication cost increases as a function of k and q_{len}. However, since ASR and PER utilize the OrderCheck procedure to reduce the number of location probes from the objects which do not violate the order of result sets, the communication overhead remains stable when k increases. This confirms the feasibility of the OrderCheck procedure as well as the c-kNN update mechanisms of our approach. The PER approach basically monitors all the moving objects. Therefore, the number of k is irrelevant to the communication cost; however, PER is not scalable when there is high query coverage.

4.3.5 Mobility

Finally, we evaluated the impact of the mobility rate. Figs. 10(a) and 10(b) show the communication cost as a function of the object and query mobility, respectively. The ASR approach achieves a reduced location update rate compared to the other two approaches for all mobility rates. PER and SR* have worse performance in terms of communication cost when the mobility rate is high. The degradation is caused by the location probes due to query movements.

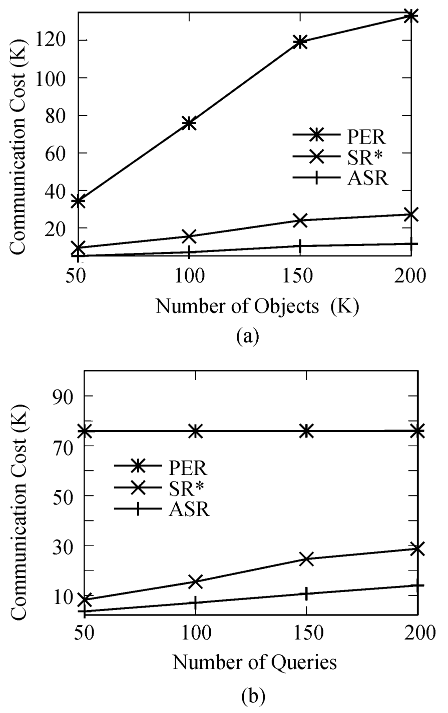


Fig.8. Object and query cardinality. (a) P vs. communication cost. (b) Q vs. communication cost.

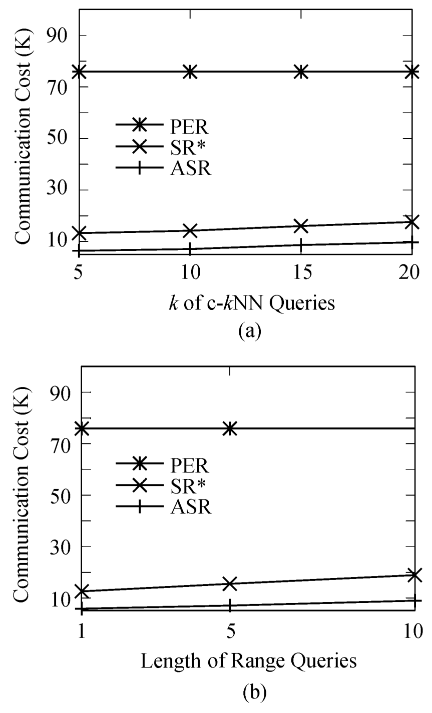


Fig.9. Effect of query coverage with k and q_{len}. (a) k vs. communication cost. (b) q_{len} vs. communication cost.

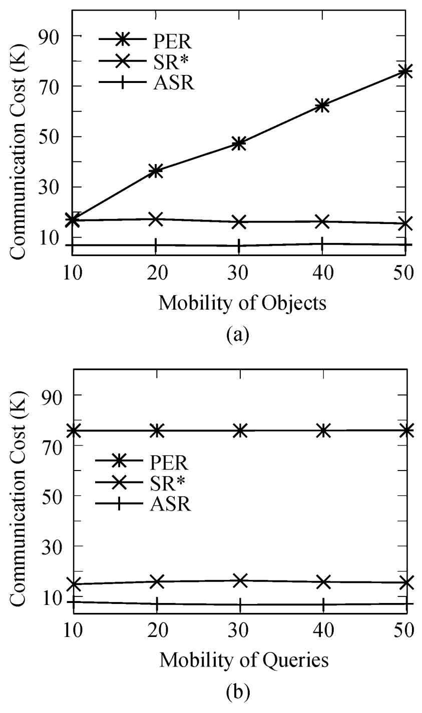


Fig.10. Object and query mobility. (a) Object mobility vs. communication cost. (b) Query mobility vs. communication cost.

5 Arbitrary Movement Model for Moving Objects

As the ASR can no longer work efficiently in an unconstraint, arbitrary movement environment, we propose the PLU approach to cover more real-world scenarios. To describe what motivates this approach, let us first illustrate how the traditional techniques operate with Fig.11 serving as an example. The gray areas represent the safe regions of two moving objects p_1 and p_2. A traditional safe region is either a rectangle or a sphere which is determined by the set of surrounding queries[9]. When an object moves outside of its safe region, it incurs a location update. From the example we can observe that, as p_1 or p_2 moves out of its safe region (in the direction of the arrow), it issues an unnecessary update because of the limited safe region information. Furthermore, the safe region of a moving object is determined based on its current location. When a query moves to a new location or a new query is inserted, the server triggers a location probe to the affected moving objects and re-calculates new safe regions for them. When receiving the location probes (downstream) from the server, the moving objects need to send their locations (upstream) back to the server. Once the server completes the safe region computations, it sends the safe regions (downstream) to those moving objects. Hence a total of three network messages are sent back and forth between the server and each mobile client. As illustrated, the safe region approach incurs significant network traffic in this scenario.

In contrast, we propose a partition-based technique by defining a grid-like LIT (also shown in Fig.11) which provides a moving object with a detailed view of the surrounding query locations across the terrain. As an additional advantage, an LIT is determined without referring to the locations of moving objects. If a query is inserted, the server can send the new LIT with the added query information (downstream) to the affected moving objects directly, and only a fraction of the mobile clients that receive the updated LIT must issue location updates (upstream) back to the server (namely if they are part of the new query result). Therefore, the number of network messages is reduced to at most two. The overall PLU process is discussed in detail in the subsequent sections.

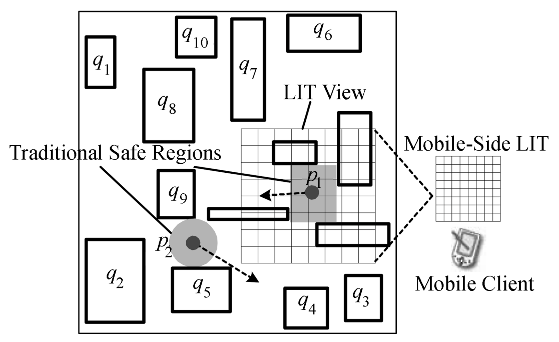
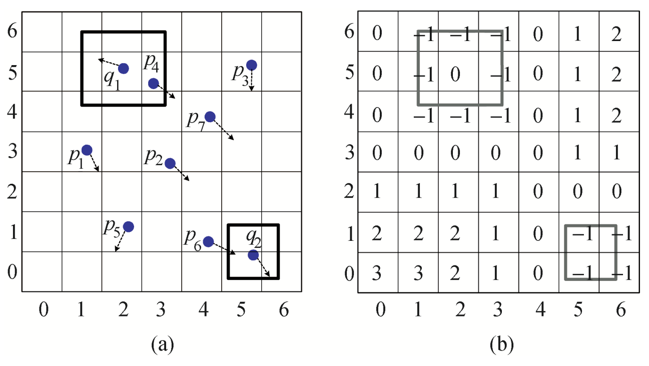


Fig.11. Illustration of concepts for the PLU approach.

5.1 LIT Details

An LIT_{serv} is generated initially at the server and updated when one of the following two events happen: 1) an existing query changes its location or 2) a new query is registered with the system. The general attributes described in this subsection for the sever-side LIT are also applicable to the mobile-side LITs extracted from it. A mobile-side LIT (LIT_{mob}) assigned to a moving object is a subset table of the server-side LIT due to memory limitations of moving objects and to reduce communication costs and it simply inherits all the attributes and query boundary information from the server-side LIT. However, each moving object maintains (i.e., updates) the mobile-side LIT locally after receiving it from the server based on a specific event. An $LIT.value$ for $LIT_{\text{serv}}(i,j)$ stores an integer number that represents a *safe distance*. The safe distance for $LIT_{\text{serv}}(i,j)$ is defined as the minimal linear distance in cells from the $LIT_{\text{serv}}(i,j)$ cell to the nearest query boundary. We distinguish two cases when assigning a value to $LIT_{\text{serv}}(i,j)$: $LIT.value \geqslant 0$, if $LIT_{\text{serv}}(i,j)$ does not overlap a query boundary; and $LIT.value = -1$, if $LIT_{\text{serv}}(i,j)$ is covered by a query boundary. Fig.12(a) shows an object grid with a set of registered queries and moving objects on the terrain at time t_0. The corresponding server-side LIT created at t_0 is illustrated in Fig.12(b).

In this example we assume that the server-side LIT size is the same as the object grid. The LIT values of the cells that overlap with the boundaries of queries q_1 and q_2 are set to -1. We define two types of cell zones: a *border zone* (LIT value $= -1$) and a *zero zone* (LIT



6	0	−1	−1	−1	0	1	2
5	0	−1	0	−1	0	1	2
4	0	−1	−1	−1	0	1	2
3	0	0	0	0	0	1	1
2	1	1	1	1	0	0	0
1	2	2	2	1	0	−1	−1
0	3	3	2	1	0	−1	−1
	0	1	2	3	4	5	6

Fig.12. The object grid and a server-side LIT example. (a) P vs. communication cost. (b) Q vs. communication cost.

value = 0). A *border zone* consists of cells that overlap with the boundaries of some queries. A *zero zone* is essentially a prediction zone which might be covered by nearby moving queries as time proceeds. Since a zero zone has a safe distance equal to zero, it is more likely to be covered by a moving query, say q_1, soon. Both border and zero zones are important indicators for a moving object to decide on a location update. In order to predict the moving query locations, each moving object updates its local LIT and marks the new prediction cells as zero zones. The detailed update mechanism for mobile-side LITs will be described later.

5.2 Mobile-Side Processing

Each moving object independently performs the following two major tasks to achieve the desired location update traffic reduction: progressive revision of the mobile-side LIT and determination of when to send location updates. Each time a moving object transmits its location to the server, an up-to-date mobile-side LIT will be sent to the moving object. However, since we consider dynamic queries, the LITs are subject to change whenever the queries change their locations during the course of the execution. Instead of sending a new mobile-side LIT with the latest query locations to each moving object repeatedly, we propose a periodic LIT update method to independently adjust the mobile-side LIT to reflect all the possible query movements while ensuring the correctness of the query results. We first discuss how a moving object updates its local LIT and then describe the mechanisms for triggering a location update based on the mobile-side LIT.

Mobile-Side LIT Updates. Under the maximum speed λ constraint, we can estimate the possible query locations in the mobile-side LIT. Continuing the example shown in Fig.12, each moving object p is given a mobile-side LIT by the server at t_0 as shown in Fig.12(b). Fig.13(a) illustrates the current locations of mobile units at time t_1 and Fig.13(b) shows the mobile-side LIT updated by p at t_1. One can observe that in the worst case, by considering that a query may move to its surrounding cells in any direction, the area between two dashed rectangles shows all the possible coverage of the query boundary with such movements. Since a border zone may overlap with more than one query boundary anywhere within the zone, the two solid rectangles represent the outermost query boundaries of the zone. For simplicity, we draw two dashed rectangles inwardly (shrunk) and outwardly (expanded) by extending the solid rectangle by the length of the maximum moving distance χ ($= \lambda \times \Delta t$) for every time instance. The cells that are newly covered by the area between the dashed rectangles become zero zones. As a final step, the LIT values of the remaining cells need to be updated by decrementing the LIT values by one when the surrounding cells become new zero zones.

Location Update Check. The event-driven procedure for deciding on a location update is performed by the moving object only when it moves to a new location. We continue with the example of Fig.13(a) that shows the new locations of queries and moving objects at time t_1. Referring to the mobile-side LIT in Fig.13(b), p_2 in $LIT_{\text{mob}}(3, 3)$ steps into a zero zone in $LIT_{\text{mob}}(4, 2)$, so p_2 might overlap with a query at this moment. p_4 was in a border zone and it changed its location since the latest update to the server, so it may exit or enter a query boundary. Therefore, both p_2 and p_4 have to issue a location update at t_1. Fig.13(c) shows a one-level 3×3 LIT_{mob} for p_3 and after some time instances at t_3, p_3 moves out of the LIT boundary to $LIT_{\text{mob}}(4, 3)$, and therefore it must issue a location update at t_3.

5.3 Server-Side Processing

When a new query q is inserted, instead of informing the entire registered moving objects population (that lack this new query boundary information) of the changes, the server performs the QurInsert procedure to determine a set of moving objects O that may enter the new query boundary. Then it sends the latest

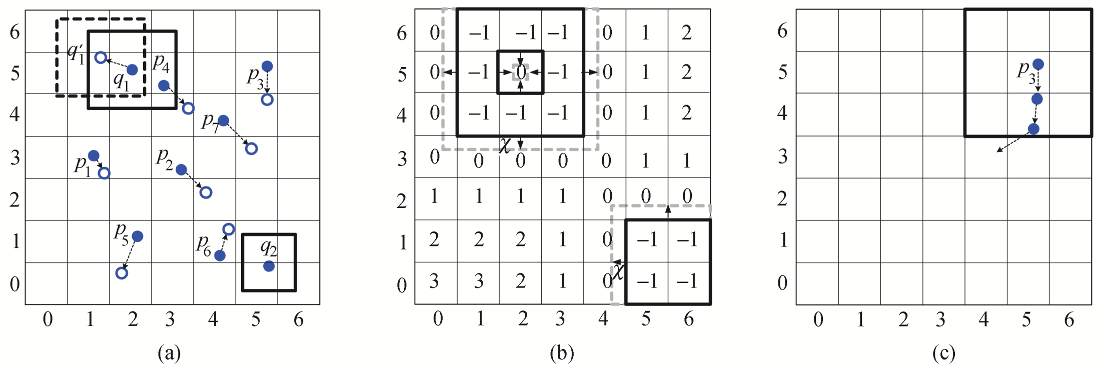


Fig.13. The object grid and mobile-side LITs. (a) Moving objects at t_1. (b) Mobile-side LIT at t_1. (c) One-level LIT_{mob}.

mobile-side LIT to these objects only. First, QurInsert checks each moving object p in the set of LIT cells C, where the objects have the mobile-side LIT overlapping with R (the set of LIT cells covered by the new query boundary). Let $c_r \in R$ be the nearest LIT cell of p. The procedure computes the minimum distance x between p and c_r and the distance y between c_r and the nearest border zone to R (denoted by c_i). If $x > y$, the server does not have to inform p of the query insertion. This is because through the mobile-side LIT updates on p, the area covered by R will become zero zones before p moves into that area. Therefore, the query insertion will not cause any missed location updates. To estimate the distance y for object p on the server side, the procedure checks the LIT value of c_r (which is $p.LIT.value$), because it represents the nearest distance (in cells) to the border zone. Since we consider the worst case to ensure the correctness of query results, the distance y is set to $p.LIT.value + k$, where k is the maximum distance in cells between c_p and c_r.

Consider the following example. A new query q registers with the system at t_1. Assume that each moving object is assigned a 3×3 (level $\ell = 1$) mobile-side LIT. Fig.14 shows an object p in C with its $LIT_{\rm mob}$. The new query q covers the gray area R. Since a one-level mobile-side LIT is assigned to each moving object, the area C is one-level larger than R. Assume that $p.LIT.value = 1$, so the estimated LIT value of the closest cell c_r at $(2, 3)$ is 2, which is the value of y. Since $x < y$, p may reach c_r before c_r becomes a zero zone through p's mobile-side LIT updates. Therefore, the server needs to inform p of the new insertion.

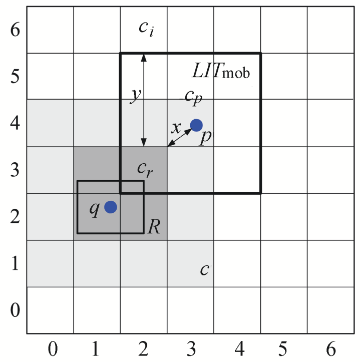


Fig.14. A query insertion example.

5.4 Spatial Data Compression for Mobile-Side LITs

While a mobile-side LIT provides more detailed query boundary information than a safe region, the data transmission of a potentially large LIT needs to be broken into more packets which may adversely affect performance. We use the Internet standard for the largest data packet payload size (MTU) equal to 1500 bytes. We apply three consecutive lossless data compression methods: delta encoding, Run-Length Encoding (RLE) and Huffman encoding. First, we decorrelate the LIT values by subtracting pairs of adjacent LIT numbers. Second, RLE is utilized to take advantage of the large amount of spatial redundancy in an LIT and we use a Hilbert curve as the data scanning path along which we count repeated numbers. Finally, we perform Huffman encoding which is based on the frequency of occurrence of a data item and uses a lower number of bits to encode the data that occur more frequently. Overall, our experimental result shows the combination of these methods can reduce the size of an LIT by up to 79% from its original size.

5.5 Experimental Evaluation

We implement the extended safe region approach[6,8] with safe rectangles (SR*-Rec) and safe spheres (SR*-SP) in addition to a periodic update approach (PER) as our compared work.

5.5.1 *Simulation Steps*

We use similar simulation settings used in the ASR approach. We select an optimal size n for the sever-side LIT from [64, 512] per side and choose the level ℓ from [1, 10] for a mobile-side LIT. The main measurement in the following simulations is the number of network messages sent between the server and moving objects. We count the number of messages (downstream) from probing an object's location and sending an LIT to a mobile unit and the number of messages (upstream) from issuing a location update to the server. Table 3 summarizes the default parameter settings in the following simulations.

Table 3. Simulation Parameters for the PLU Approach

Parameter	Default	Range
P	100K	-
Q	1000	300, 500, 700, 1000
$f_{\rm move}$	50%	10%, 30%, 50%, 70%, 100%
λ	1.25	0.48 (35mph)∼1.25 (90mph)
$q_{\rm len}$	5	1, 5, 10
n	256	64, 128, 256, 512
ℓ	5	1, 5, 10

5.5.2 *LIT Size*

First, we measure the overall number of network messages including upstream and downstream directions of the PLU algorithm by varying the server-side

LIT size. The choice of the server-side LIT size is a trade-off between the number of network messages and the server performance. Figs. 15(a) and 15(b) show the number of overall network messages and CPU overheads vs. the LIT sizes ranging from 64×64 to 512×512, respectively. When the LIT size is set to more than 512 per side, the performance of PLU is degraded in terms of the number of network messages and CPU time because it incurs more LIT value calculations for all the LIT cells. The LIT size 256×256 constitutes a good tradeoff between the number of network messages and CPU time. Therefore, 256×256 is chosen as the server-side LIT size for the rest of our experiments.

Next we examine the size for a mobile-side LIT. Fig.15(c) measures the effect of varying the size of the mobile-side LIT from level 1 (3×3) to level 10 (21×21) in terms of network messages. The size of a mobile-side LIT significantly affects the number of network messages. When a mobile-side LIT is small, a moving object issues more network messages because it has more chance to move out of the LIT boundary. When a mobile-side LIT is large, it also incurs more network messages from the query insertion process since the procedure needs to check more objects from a larger area where the moving objects have the mobile-side LITs overlapping with the new query boundary. We choose $\ell = 5$ as the mobile-side LIT size for the remaining experiments, because it achieves better performance in terms of the network messages.

5.5.3 *Query Coverage*

The query coverage on the terrain is a crucial factor in the performance of continuous query algorithms. The query coverage varies with the number and side length of the queries. Fig.16(a) shows the network messages as a function of the number of queries and Fig.16(b) illustrates the corresponding communication cost. Overall, the number of network messages and communication cost increase as a function of the number of queries, because the chance of moving into the query boundaries for a moving object is high. PLU achieves a significant reduction in the number of updates compared to the other techniques. For the PER approach, since the server does not perform any computations regarding the location update reduction, we only count the number of network messages sent from the mobile clients. PER approach is independent of the query coverage, because the number of updates depends on the mobility only in PER approach. Therefore, the network messages remain the same in this simulation. In Fig.16(c), we evaluate the side length of queries with the values [1, 5, 10]. Obviously, when the length of queries increases, SR*-Rec and SR*-SP incur more updates, because SR*-Rec and SR*-SP perform server-side probes to those objects which have the safe

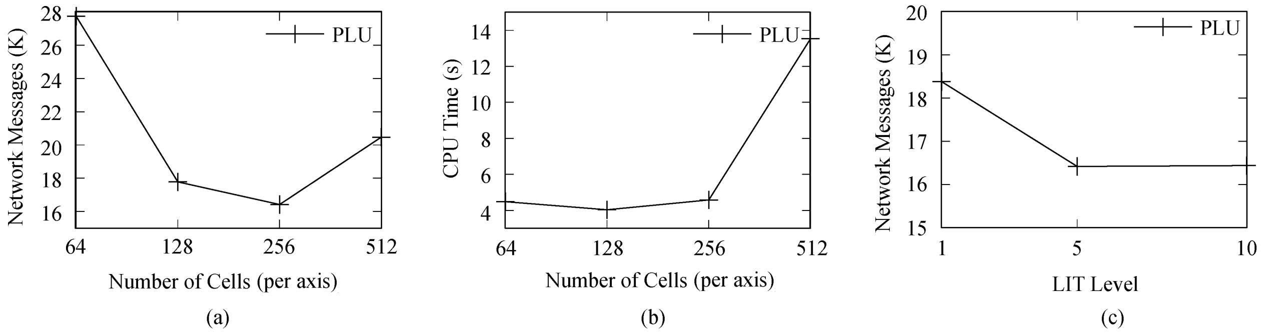


Fig.15. Performance vs. LIT size. (a) and (b) Server-side LIT size (n). (c) Mobile-side LIT size (ℓ).

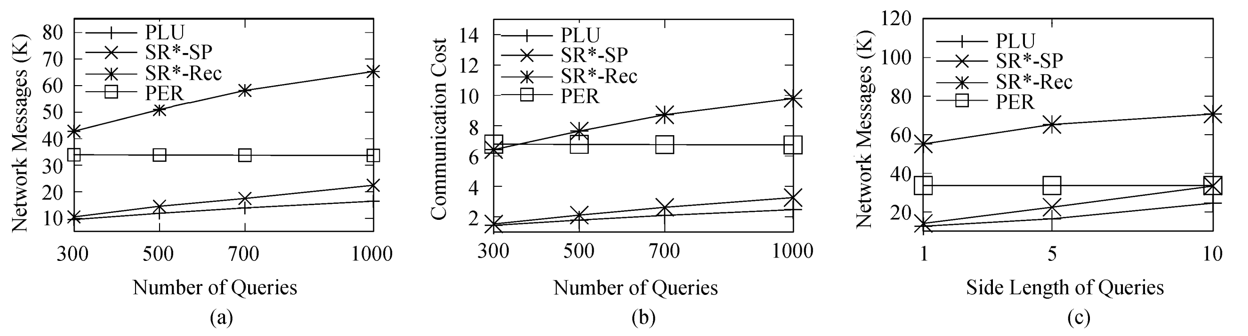


Fig.16. Effect of query coverage with Q and $q_{\rm len}$. (a) Q vs. network messages. (b) Q vs. communication cost. (c) $q_{\rm len}$ with $Q = 1000$.

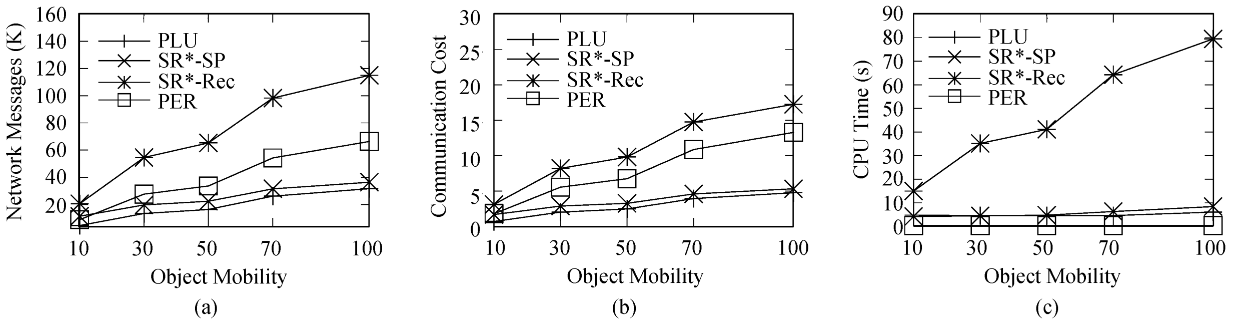


Fig.17. Performance vs. object mobility. (a) $f_{\rm move}$ vs. network messages. (b) $f_{\rm move}$ vs. communication cost. (c) $f_{\rm move}$ vs. CPU time.

regions overlapping with the queries. When the length of the queries increases, the server needs to probe more moving objects when queries change to new locations or when new queries are inserted. The simulation results confirm the importance of adopting the PLU approach which significantly reduces the network messages and hence decreases the communication cost.

5.5.4 Mobility

Finally, we evaluate the impact of the mobility rate. Fig.17(a) shows the number of network messages as a function of the object mobility and the communication cost is also shown in Fig.17(b). The PLU approach achieves a higher location update reduction than the other three approaches for all mobility rates. Fig.17(c) illustrates the CPU time vs. the object mobility. Although PLU applies more server-side procedures (e.g., QurIns) to reduce the network messages. PLU still has a competitive CPU performance with SR*-SP. However, SR*-Rec has the worst performance in terms of network messages/communication cost and CPU overheads. The degradation is caused by the expensive calculations of safe rectangles. SR*-Rec in general computes larger safe regions for moving objects than SR*-SP, so SR*-Rec incurs many server-side probes to the moving objects when queries change their locations.

6 Conclusions

We address two challenging issues in efficient query evaluation and low communication cost related to frequent location updates. We propose the ASR and PLU algorithms to reduce the number of location updates for different movement models. We have designed an ASR-based framework for trajectory movement environments. The novel concept of an adaptive safe region is introduced to provide a mobile object with a reasonable-sized safe region that adapts to the surrounding queries. Hence, the communication overhead resulting from the query movements is greatly reduced. To further decrease network traffic caused by c-kNN query region expansions to cover sufficient NNs for the result sets, our approach caches extra NNs. The PLU approach is designed for arbitrary movement environment where mobile units may freely change their locations to any positions. The novel concept of an LIT table is introduced to provide a mobile object with information about queries, hence enabling it to estimate query movements and transmit a location update to the server only when it affects the query results. To further reduce network traffic the server uses smart on-demand location probes. Finally, the proposed mechanism efficiently determines the set of objects that are affected by a query insertion, improving scalability. Experimental results demonstrate that both approaches scale better than existing techniques in terms of the communication cost and the outcome confirms their feasibility.

References

[1] Jensen C S, Lin D, Ooi B C. Query and update efficient B+-tree based indexing of moving objects. In *Proc. VLDB*, Toronto, Canada, Aug. 31-Sept. 3, 2004, pp.768-779.

[2] Mokbel M F, Xiong X, Aref W G. SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In *Proc. SIGMOD Int. Conf. Management of Data*, Paris, France, June 13-18, 2004, pp.623-634.

[3] Mouratidis K, Hadjieleftheriou M, Papadias D. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In *Proc. SIGMOD Int. Conf. Management of Data*, Baltimore, USA, Jun. 14-16, 2005, pp.634-645.

[4] Tao Y, Papadias D, Sun J. The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In *Proc. VLDB*, Berlin, Germany, Sept. 9-12, 2003, pp.790-801.

[5] Xiong X, Mokbel M F, Aref W G. SEA-CNN: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *Proc. ICDE*, Tokyo, Japan, Apr. 5-8, 2005, pp.643-654.

[6] Hu H, Xu J, Lee D L. A generic framework for monitoring continuous spatial queries over moving objects. In *Proc. SIGMOD Int. Conf. Management of Data*, Baltimore, USA, Jun. 14-16, 2005, pp.479-490.

[7] Mouratidis K, Papadias D, Bakiras S, Tao Y. A threshold-based algorithm for continuous monitoring of k nearest neighbors. *IEEE Transactions on Knowledge and Data Engineering*, 2005, 17(11): 1451-1464.

[8] Prabhakar S, Xia Y, Kalashnikov D V, Aref W G, Hambrusch S E. Query indexing and velocity constrained indexing:

Scalable techniques for continuous queries on moving objects. *IEEE Transactions on Computers*, 2002, 51(10): 1124-1140.

[9] Prabhakar S, Xia Y, Kalashnikov D V, Aref W G, Hambrusch S E. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Transactions on Computers*, 2002, 51(10): 1124-1140.

[10] Cai Y, Hua K, Cao G. Processing range-monitoring queries on heterogeneous mobile objects. In *Proc. MDM*, Berkeley, USA, Jan. 19-22, 2004, p.27.

[11] Yuu X, Pu K Q,Koudas N. Monitoring k-nearest neighbor queries over moving objects. In *Proc. ICDE*, Tokyo, Japan, Apr. 5-8, 2005, pp.631-642.

[12] Saltenis S, Jensen C S, Leutenegger S T, Lopez M A. Indexing the positions of continuously moving objects. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, Dallas, USA, May 15-18, 2000, pp.331-342.

[13] Tao Y, Faloutsos C, Papadias D, Liu B. Prediction and indexing of moving objects with unknown motion patterns. In *Proc. SIGMOD Int. Conf. Management of Data*, Paris, France, June 13-18, 2004, pp.611-622.

[14] Mokbel M F and Aref W G. GPAC: Generic and progressive processing of mobile queries over mobile data. In *Proc. MDM*, Ayia Napa, Cyprus, May 9-13, 2005, pp.155-163.

[15] Cheng R, Lam K Y, Prabhakar S, Liang B. An efficient location update mechanism for continuous queries over moving objects. *Information Systems*, 2007, 32(4): 593-620.

[16] Cheng R, Lam K Y, Prabhakar S, Liang B. An efficient location update mechanism for continuous queries over moving objects. *Information Systems*, 2007, 32(4): 593-620.

Yu-Ling Hsueh received her Ph.D. and M.S. degrees in computer science from University of Southern California (USC) in 2009 and 2003, respectively. Her research interests are temporal/spatial databases, moving object processing, scalable continuous query processing and spatial data indexing. She is currently working for Teradata Corporation.

Roger Zimmermann is an associate professor with the Department of Computer Science at the National University of Singapore (NUS) where he is also an investigator with the Interactive and Digital Media Institute (IDMI). Prior to joining NUS he held the position of research area director with the Integrated Media Systems Center (IMSC) at the University of Southern California (USC). He received his Ph.D. degree from USC in 1998. His research interests are in the areas of distributed and peer-to-peer systems, collaborative environments, streaming media architectures, georeferenced video search, and mobile location-based services. He has co-authored a book, four patents and more than a hundred conference publications, journal articles and book chapters in the areas of multimedia and information management. He is an associate editor of the ACM Computers in Entertainment magazine and the ACM Transactions on Multimedia Computing, Communications and Applications journal. He is a senior member of the IEEE and a member of ACM.

Wei-Shinn Ku received his Ph.D. degree in computer science from the University of Southern California (USC) in 2007. He also obtained both the M.S. degree in computer science and the M.S. degree in electrical engineering from USC in 2003 and 2006 respectively. He is an assistant professor with the Department of Computer Science and Software Engineering at Auburn University. His research interests include spatial and temporal data management, mobile data management, geographic information systems, and security and privacy. He has published more than 40 research papers in refereed international journals and conference proceedings. He is a member of the ACM and the IEEE.