Generic Adaptive Moving Object Tracking Algorithms

Jing Zhou[†] Hong Va Leong[†] Qin Lu[†] Ken C.K. Lee[‡]
[†]Department of Computing
The Hong Kong Polytechnic University
Hong Kong
{csjgzhou, cshleong, csluqin}@comp.polyu.edu.hk
[‡]Dept. of Computing Science and Engr.
Pennsylvania State University
University Park, PA 16802
cklee@cse.psu.edu

Abstract

Moving Object Databases (MODs), the core component of location server to support location-related applications, keep track of the locations of moving objects which submit location update reports to the centralized server. In resource-limited wireless environments, the frequency and conditions for generating location update messages exert a strong impact on system performance in terms of update message cost and object location accuracy, hence the query result precision. Conceptually, moving objects are the sources of the location data while the MOD caches recently reported object locations for query processing. Owing to the inherent imprecision of the cached values, we impose a bounded level of inconsistency for the cached values, realized in the form of a "safe range" *for a moving object. The cached value needs not be invalidated so long as the deviation of the object's current location from its reported location is within the safe range. A smaller safe range results in a higher accuracy of the cached value and hence more accurate query result at the expense of higher update cost, and vice versa. Since the size of the safe range is the key to system performance, we derive a system cost model to determine its appropriate value. Furthermore, to cater for highly dynamic environments in which object movement, query access pattern and system workload always change, we propose two* adaptive *safe range adjustment algorithms. Through extensive simulation experiments, the benefits brought about by our algorithms are evidenced.*

1. Introduction

Moving object location tracking is an essential service in many location-related applications such as intelligent transportation system, logistics, fleet/cargo management, childcare system, wildlife animal monitoring, emerging service like E911 etc. Consider the intelligent transportation system, often launched in a city for answering queries from drivers, passengers, police and other interested parties. One typical task is to keep monitoring locations of interested moving objects and log the updated location value to a Moving Object Database (MOD) for centralized efficient query processing. Figure 1 depicts a MOD, the core component of location server, keeping track of the locations of a large number of moving objects such as persons, vehicles, and wildlife animals and processing user queries on these moving objects. Under this configuration, moving objects report their locations to the MOD either periodically, on demand, or when a certain condition materializes. Maintaining reported locations, MOD generates results to user queries about the locations of interested moving objects.

In our model, the system performance is attributed to two costs: *location update cost* and *query evaluation cost*. The location update cost is basically the cost to post location updates from battery-powered moving objects to the database server over a scare-bandwidth wireless channel. Clearly, it is impractical for an object to keep submitting location reports continuously along its trajectory. To reduce the update cost, state-of-art update policies like time-based [1], distance-based [11], dead-reckoning [14], adaptive dead-reckoning [7] and group-based location update [6] have been proposed. The query evaluation cost is resulted when the object location at server is imprecise and the server needs to page the object for its updated location.

Conceptually, the moving objects are the sources of their location information (source data) and the database server maintains a cache of latest reported object locations (cached data). The problem of location update and querying resembles the cache coherence problem. Though location update can be realized as cache update or cache invalidation initiated by moving objects to MOD, we adopt cache update approach since the payload of a location update message that includes object ID and spatial coordinates is very close to an invalidation message that contains the object ID only.

In this paper, we propose a distance-based update approach with adaptively tuned *safe range* to realize the concept of location cache at MOD. The safe range defines the degree of bounded inconsistency [16], which is the maximum deviation allowed between the cached location and the actual location. Thus, when an object moves to a new position within the safe range, the cached value remains valid; no update needs to be issued to the server. From the per-

cached location within the safe range, in the absence of an update. The object location may be stale, but accurate enough to answer queries with fair accuracy expectation. For queries with high accuracy requirement, relevant objects will be paged when queried. This on-demand paging cost, resulted when high accuracy queries access object location with low precision, constitutes the query evaluation cost. This object paging resembles cache verification.

In our algorithms, the adaptation of safe ranges is mutually agreed between the server and the moving objects. Based on the safe range r, a location update is initiated only when the object is more than a distance of r away from its previously reported location. The area within which no update is needed is called the *safe region*. It is a circular area centered at the previously reported location with radius r.

Although to a certain degree, the precision of cached value is bounded, processing a query with a high precision requirement, as specified by users, could likely trigger object paging. We should set the safe range judicially to improve system performance. However, selecting a good safe range needs thorough analysis since a small safe range is not effective in reducing location update frequency, while a large safe range incurs high object paging cost, especially when a large volume of queries are to be served, despite suppressing the location updates. Even more complicated, the safe range is not necessarily fixed; it should be adaptive in highly dynamic environments in which object movement pattern and query access pattern always change. To address this, we propose two adaptive safe range determination algorithms that are both *movement-aware* and *query-aware* [18]. To evaluate, we conduct extensive experiments with synthetic datasets. The experimental results well indicate their effectiveness in balancing the update cost and query cost, resulting in an optimal system performance.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 describes the general system model and the operation of each component. Section 4 derives the system cost model, providing the basis to develop our adaptive algorithms discussed in Section 5. We conduct experiments to evaluate the performance of our adaptive algorithms, demonstrating their effectiveness in yielding minimal operational cost. Details are documented in Section 6. Finally, Section 7 summarizes our work and highlights some future research directions.

2. Related Work

In distributed information systems, the problem of *adaptive data caching* is related to adjusting the caching strategy dynamically as conditions change [3, 4, 13]. Most previous work does not focus on the problem of how to set cache precision optimally according to data value change and query situation [2, 10, 12, 17]. A distinguished work addressing the optimization problem considers interval approximation ing the precision of cached approximations [9]. One of our algorithms extends this methodology and makes the general approach applicable for MOD applications.

Although moving object tracking is essential in mobile computing applications, little work is done in MOD to address the location information caching optimization problem. A handful of tracking algorithms and updating policies such as time-based, distance-based, dead-reckoning are proposed [1, 7, 11, 14] and comparison work has been done [11]. However, none of the existing work focuses on optimization efforts. To our best knowledge, the most relevant work taking the optimization approach is from Wolfson [14]. In their work, an information cost model is proposed based on three separate costs: update cost, deviation cost and uncertainty cost. They derive the optimal settings for deviation threshold to yield the minimum information cost, based on the dead-reckoning policies. In their model, queries are not allowed to request for exact values from sources so server probe costs are not taken into account. Compared to their work, ours is much more adaptive in the sense that it is both *query-aware* and *movement-aware* [18].

The novelty of our approach lies in three aspects. First, Wolfson's work regards querying and updating as two *separate procedures* in MOD environment, in which the uncertainty and deviation impose a cost or penalty in terms of incorrect decision making. Based on this, the information cost function is designed to absorb both update cost and penalty for uncertainty, so as to derive the minimal cost for one trip by assigning the optimal value to the object's deviation threshold. They then consider the query issued in a next step. We believe that in most practical systems, the only reason to provide up-to-date location information is to provide answers precise enough to queries concerning these objects. Therefore, if no query is issued for the whole trip of a moving object, it is not necessary to "produce" the information for the non-existent "consumer". Thus, both movement pattern and query information are absorbed and integrated into our adaptive algorithms. Second, the dead-reckoning policies suffer from the dependency of the optimal deviation threshold value on a predicted deviation function. The deviation threshold at each update is adjusted to the current motion pattern, whose changes need to be reflected by parameter change on the predicted deviation function, such as the linear function, $d(t)$ [14]. In contrast, our adaptive algorithms do not need to be based on any predicted deviation function. Third, the dead-reckoning policies are only applicable when the destination and motion plan of the moving objects are known a priori [15]. The route would be fixed and known to both the moving object and the server. However the future route of a tracked object is not always known. Our work relaxes the assumption about predefined routes or destination about the motion plan.

Figure 1 depicts the general updating and querying activities in the MOD, which resides inside some kind of location server providing location-dependent querying services. The location server communicates with moving objects via a low bandwidth network and records their locations. Upon receiving queries issued by users, location server processes them and returns the results. It is normally the responsibility of the moving objects to generate update reports on their locations to the server. Both server and moving objects are aware of the update policies and the agreeed safe range of location information stored in the MOD.

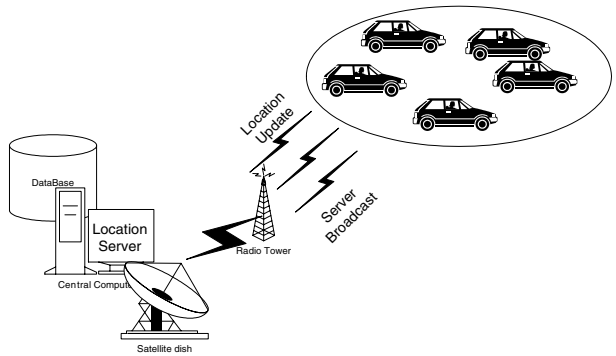

Figure 1. Updating and querying in MOD

Let the set of moving objects be $\mathcal{O} = \{o_1, o_2, o_3, ...\}$ and the set of queries issued be $\mathcal{Q} = \{q_1, q_2, q_3, ...\}$. Our adaptive algorithms are built based on the distance-based update policy. A moving object, $o \in \mathcal{O}$, issues an update whenever the distance between the current object location, $o.\mathsf{loc}_{\mathsf{current}}$, and the stored location, $o.\mathsf{loc}_{\mathsf{stored}}$, exceeds the *safe range* r, i.e., $|o.\mathsf{loc}_{\mathsf{current}}, o.\mathsf{loc}_{\mathsf{stored}}| > r$. Upon each update, a tuple of the current location and chosen safe range r_i for each object o_i is stored at the server. Queries processed at the server are of the form $q_j = \langle o_i, p_j \rangle$, where o_i is the identification for the required tracking object, and p_j is the precision constraint for query $q_j \in \mathcal{Q}$. In general, to process q_j, if $r_i \leq p_j$, the stored location at the server satisfies the precision requirement and is returned to the query issuer immediately. If $r_i > p_j$, the stored location is inadequate in precision. The server needs to page the moving object o_i for its current location, $o_i.\mathsf{loc}_{\mathsf{current}}$.

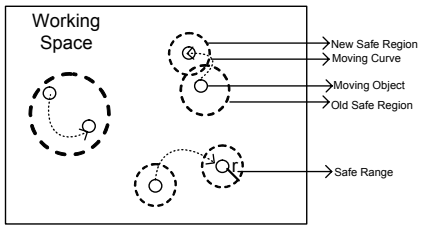

Figure 2. Moving object safe region

The procedures executed at a moving object o_i and the server S in order to maintain the MOD and to process the queries are illustrated in Figures 3 and 4. There is a *safe* within which there is no need for the object to report its location. An update is only needed when the object moves out of its safe region. The safe region is a circle centered at the previously reported location (the stored location at the server), with radius r_i. It is represented by the dotted circle in Figure 2 for each object. In other words, an object reports its new location if the deviation in location exceeds the safe range. Upon receiving a location update, the server installs the new location, determines the new safe range and informs the moving object. Meanwhile, the server monitors querying activities to adjust the safe range. The new safe range r_i for o_i is determined based on our adaptive algorithms discussed in Section 5.

Procedure for moving object o_i
1: monitor its own location $o_i.\mathsf{loc}$ via devices such as GPS
2: if $|o_i.\mathsf{loc} - o_i.\mathsf{loc}_{\mathsf{reported}}| > r_i$ then
// move out of safe region
3: send update report to server S: $\langle o_i, o_i.\mathsf{loc} \rangle$
4: $o_i.\mathsf{loc}_{\mathsf{reported}} \leftarrow o_i.\mathsf{loc}$
5: coordinate with S for its new safe range r_i
6: endif

Figure 3. Moving object responsibility

Procedure for server S
1: receive the next message
2: if it is an update report from o_i then
3: update the stored location of o_i with $o_i.\mathsf{loc}$
4: determine the new safe range r_i
5: endif
6: if it is a query $q_j = \langle o_i, p_j \rangle$ from a user then
7: if $r_i \leq p_j$ then
8: return the stored location to the query issuer
9: else page o_i for its exact current location
10: return the result to the query issuer
11: endif
12: determine the new safe range r_i
13: endif
14: piggyback the new safe range r_i to o_i

Figure 4. Server responsibility

4. Cost-based Analysis

The two kinds of costs, namely, wireless communication cost for location update and the query evaluation cost at the database server, are the most important factors on the system performance. Both are directly related to the location safe range, i.e., the value of the threshold r to generate an update report. In order to optimize the system performance, we adopt the cost-based approach in analyzing the performance, striking at the minimal cost with an optimal setting for r. The total system cost, $\mathcal{C}$, can be considered as a sum of the two component costs, C^T and C^Q, where C^T represents the tracking cost and C^Q stands for the querying cost. Thus, $\mathcal{C} = C^T + C^Q$. This cost function definition sets up the ultimate goal for our adaptive algorithms: minimization

key factors that are influential to each of the costs and derive the appropriate settings to effect our adaptive algorithms design in optimizing the total cost $\mathcal{C}$.

4.1. Tracking Cost for Moving Objects

We assume in this paper that the tracking of moving objects is achieved by voluntarily issuing update reports from moving objects to the server. Cost of sending updates upon a paging request from the server belongs to probing cost and will be associated with the query instead, as discussed in next section. The cost for tracking mainly depends on the cost for update activities and we define the tracking cost, C_i^T, for an object o_i as:

$$C_i^T = C_u \rho(r_i) \tag{1}$$

where C_u is the unit cost for an update activity. In other words, it is the cost paid for one object to send one update report to the server. The value of C_u is application-dependent. $\rho(r_i)$ is update rate for moving object o_i, i.e., the number of updates initiated from o_i per time unit, with safe range r_i. Intuitively, larger r_i leads to smaller rate. Hereafter, we will drop the subscript when it is clear from the context. Obviously, $\rho(r)$ depends on the movement pattern of the objects.

To define the formula for $\rho(r)$, we need to analyze the object movement behavior. Let us assume that the moving objects obey a 2-dimensional random walk model. All the objects move in steps and in each step, each object travels a distance of d along an arbitrary direction. Each step takes a duration of L time units.

Lemma 1 *If the movement of an object o_i follows the random walk model, each movement step lasting for a period of time L_i and the distance moved in each step being d_i, then the rate at which o_i moves out of the safe region is $d_i^2/(L_i r_i^2)$.*

Proof. To prove this lemma, let us review the well-known *Drunken Person Problem*.

> Drunken Person Problem. A drunken person moves following the random walk model. Suppose that in every step, she moves a unit distance. After n steps, the distance between her current location and the starting point is $\sqrt{n}$. ■

The movement of an object is similar to the drunken person. The lemma can be proved with result from the Drunken Person Problem. For an object o_i to move out of its safe region, it should move at least a distance of r beyond its safe region center, corresponding to the starting point in the drunken person problem. The normalized distance from the starting point to the boundary is $\frac{r_i}{d_i}$. Let T be the expected time that o_i moves out of its safe region. It is obvious that $\rho(r_i) = \frac{1}{T}$. Suppose at t_0, o_i is located at safe region center and at time $t_0 + T$, it is expected to distance moved between this time period is $D_1 = \frac{r_i}{d_i}$ and the number steps that o_i moves is $n = \frac{T}{L_i}$. Thus, the distance between the starting point and the current point is $D_2 = \sqrt{n} = \sqrt{\frac{T}{L_i}} = \sqrt{\frac{1}{L_i \rho(r_i)}}$. Since $D_1 = D_2$, we have $\frac{r_i}{d_i} = \sqrt{\frac{1}{L_i \rho(r_i)}}$. Thus, $\rho(r_i) = \frac{d_i^2}{L_i r_i^2}$. □

Similar to [9], we can generalize the relationship between $\rho(r)$ and r as:

$$\rho(r) = \alpha / r^2 \tag{2}$$

where α is a parameter that represents other factors except r that will affect the value of $\rho(r)$. By intuition, we know that α depends on the movement pattern of the objects.

4.2. Querying Cost for Probing Moving Objects

Besides object tracking cost, the other cost for MOD is the query processing cost. Assuming negligible CPU cost, the query processing cost is attributed to unsatisfactory safe range in location information for queries with high precision constraint. The server then needs to probe for more precise location of the moving objects involved. The costs of probing and updating from those objects comprise the querying cost, C^Q. We define the querying cost C_{ji}^Q involving an object o_i to answer a specific query q_j as:

$$C_{ji}^Q = C_p \phi(r_i) \tag{3}$$

where C_p is unit cost for probing moving objects for the current location, which translates into the cost paid to page one object and receiving the reply back. $\phi(r_i)$ is query probe rate for moving object o_i, i.e., the number of probes generated per time unit, when the query q_j is not satisfied with an object o_i with safe range r_i, for which the query precision constraint is not fulfilled. Obviously, $\phi(r)$ depends on querying pattern, including query arrival rate and precision requirement. Intuitively, with a larger value of r, higher probe rate will be resulted.

To define function $\phi(r)$, we assume a simplified case where queries arrive with rate λ, each being accompanied by a *precision constraint* sampled from a uniform distribution, $\mathcal{U}(0, p_{max})$. Then $\phi(r)$ is the number of queries issued per time unit multiplied by the probability that the precision constraint of the query is not satisfied. The probability that the precision constraint of the query is not satisfied can be computed as $Pr(p < r) = r/p_{max}$. We can evaluate the function $\phi(r)$ for an object with safe range r as $\phi(r) = \lambda r / p_{max}$. We can then generalize the relationship between r and $\phi(r)$ as:

$$\phi(r) = \beta r \tag{4}$$

where β is a summarizing factor other than r that could affect $\phi(r)$. Obviously, β depends on the distribution of query precision constraint and query arrival pattern.

With $\phi(r)$ and $\rho(r)$ determined, we can define a complete general cost function for moving objects. Table 1 summarizes all the parameters we use in the cost function and in our simulation study. We would relax the assumption in the derivation of the functions in our performance study to illustrate the robustness of our algorithms.

Symbol	Description
$\mathcal{C}$	Total cost for the system
C^T, C^Q	Tracking and querying cost
C_u, C_p	Unit cost for location update and probe
r, r^*	Safe range and its optimal value
$\rho(r), \phi(r)$	Object update and query probe rate
α	Parameter due to object movement behavior
β	Parameter due to querying behavior
δ	Cost ratio, defined as $\frac{\phi(r)}{\rho(r)}$
δ^*	Optimal cost ratio, $2\frac{C_u}{C_p}$
τ	Rate of adaptation
ω	Exponential aging parameter
ϵ	Ping-pong effect barrier parameter

Table 1. Symbols and parameters

Recall that $\mathcal{C} = C^T + C^Q = C_u\rho(r) + C_p\phi(r)$. With appropriate parameter values of α and β, $\mathcal{C} = \alpha/r^2 + \beta r$. Through mathematical analysis, we can derive the minimal value for the total cost via differentiation. This occurs when $r^* = 2C_u(\sqrt[3]{\alpha/\beta})/C_p$.

For example, if the moving object follows the random walk model, queries are generated with a stable arrival rate of λ, and precision constraints are uniformly distributed within 0 to p_{max}, We can derive α and β as $\alpha = d^2/L$ and $\beta = \lambda/p_{max}$. We can then easily achieve the optimal cost by setting the safe range to $r^* = 2(C_u/C_p)(\sqrt[3]{d^2 p_{max}/\lambda L})$.

Unfortunately, setting the safe range r to the optimal r^* is difficult unless object moving behavior and querying pattern are *stable* and *known in advance*, as assumed in most related work, since both parameters α and β depend on these factors. We need to adjust the system in an attempt to reach out for the optimal safe range r^*, despite the unstable and unknown object movement and query patterns.

5. Adaptive Safe Range Algorithms

With unknown system parameters and changing system conditions, one feasible way to strive for optimal system performance is the adaptive adjustment approach. We design two adaptive algorithms for setting the safe range towards minimizing total system cost. The basic idea behind the algorithms is from an observation on the intuitive relationship between the two kinds of costs and safe range r. Intuitively, larger r leads to lower tracking cost but higher querying cost and vice versa. Whenever a location update occurs, it is a signal that the safe range should be larger and the value of r should be increased for the next updating assignal that the safe range is too large for the precision constraint of the existing query and r should be decreased. This approach adapts the value of r according to changing situation. We then need to determine by what amount should r be adjusted. Too large a jump will create a strong ping-pong effect, defeating the purpose of the adaptation. Too small a shift will make it a lengthy process to adapt to a new object movement and querying pattern.

To determine the adjustment, we look at the system property at the optimal performance point and maneuver the adjustment based on the existing deviation from the optimal property. We observe that the ratio between $\phi(r)$ and $\rho(r)$ is a constant at the optimal safe range r^*. When $\phi(r)/\rho(r) = 2(C_u/C_p)$, $r = r^* = 2(C_u/C_p)(\sqrt[3]{\alpha/\beta})$. We thus define $\delta^* = 2(C_u/C_p)$, and the optimal safe range r^* is resulted when $\delta = \delta^*$. Our problem then reduces to adjusting the safe range r so that $\delta/\delta^* = 1$. The value of r is adjusted by an amount of τ, a tunable parameter.

Figures 5 and 6 present two adaptive algorithms for setting r at the server. The meaning of symbols used can be found in Table 1. Both algorithms attempt to adjust the system parameters in order to make the ratio of the observed query probe rate and location update rate equal to δ^*.

History-tracking Algorithm (HA) keeps track of the location update and query probe rates, in order to compute δ and make the ratio δ/δ^* close to 1 by adjusting r. Since larger r leads to higher query probe and lower location update rate and hence larger δ, HA decreases r when the ratio δ/δ^* is larger than the value in optimal condition, i.e., 1. Note that since the value of δ can change dynamically, HA needs to track changes in object movement and query patterns, by means of an exponentially weighted moving average of the metric, with an adjustable weight ω. To avoid the ping-pong effect, a change is initiated only when the ratio is beyond a certain threshold ϵ from the target value of 1.

Non-History-tracking Algorithm (NHA) only makes use of the local property of location update and query probing observed at the server under normal operation, without attempting to track for the object movement pattern, query pattern and query precision, nor storing their history. As with HA, NHA attempts to make the ratio δ/δ^* close to 1, but by means of probabilistic approach. To explain the algorithm, let us examine the simplest case when $\delta^* = 1$. To make the ratio $\delta/\delta^* = 1$, the system should keep $\delta = 1$, i.e., query probe rate should be equal to location update rate. To balance the likelihood of the two types of updates, NHA would decrease or increase r on a query probe or object update in order to reach for the optimal setting. If $\delta^* > 1$, a larger r which leads to a higher query probe rate and smaller object update rate is preferred. Therefore, NHA would still increase r on a location update but would just decrease r with a probability $1/\delta^*$ on a query probe. Conversely, if