Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 79 (2006) 441–453
The Journal of Systems and Software
www.elsevier.com/locate/jss

Adaptive schemes for location update generation in execution location-dependent continuous queries ☆

Kam-Yiu Lam [a,*], Özgür Ulusoy [b]

[a] Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong
[b] Department of Computer Engineering, Bilkent University, Bilkent, Ankara 06800, Turkey

Received 27 January 2004; received in revised form 1 July 2005; accepted 15 July 2005
Available online 26 August 2005

Abstract

An important feature that is expected to be owned by today's mobile computing systems is the ability of processing *location-dependent continuous queries* on moving objects. The result of a *location-dependent query* depends on the current location of the mobile client which has generated the query as well as the locations of the moving objects on which the query has been issued. When a location-dependent query is specified to be *continuous*, the result of the query can continuously change. In order to provide accurate and timely query results to a client, the location of the client as well as the locations of moving objects in the system has to be closely monitored. Most of the location generation methods proposed in the literature aim to optimize utilization of the limited wireless bandwidth. The issues of *correctness* and *timeliness* of query results reported to clients have been largely ignored. In this paper, we propose an *adaptive monitoring method (AMM)* and a *deadline-driven method (DDM)* for managing the locations of moving objects. The aim of our methods is to generate location updates with the consideration of maintaining the correctness of query evaluation results without increasing location update workload. Extensive simulation experiments have been conducted to investigate the performance of the proposed methods as compared to a well-known location update generation method, the plain dead-reckoning (pdr).

Keywords: Location-dependent continuous queries; Location update; Moving object database; Location management

1. Introduction

Recent advances in mobile communication and portable computer technologies have led to the emergence of various innovative mobile computing applications (Lee et al., 2002). Many of these new applications require the system to manage the locations of moving objects and to support the so-called *location-dependent queries* (*LDQs*) on moving objects (Sistla et al., 1997; Wolfson et al., 1998; Dunham and Kumar, 1998; Ren and Dunham, 2000; Seydim et al., 2001). The evaluation result of a LDQ depends on the location of the mobile client[1] which has issued the query, as well as the locations of the queried moving objects (Trajcevski et al., 2002; Zhang et al., 2003). For example, in a database management system maintaining the location information of ambulances, a typical LDQ that might be submitted to the system by the driver of an ambulance is: "identify all the other ambulances, which are within 5 km of my current position". The result of the query depends on the current location of all the ambulances. A LDQ may be submitted as a *continuous query* (*CQ*)

☆ This work described in this paper was supported by a research grant from the Research Grant Council of Hong Kong Special Administration Region, China [Project No. CityU 1076/02E].
* Corresponding author. Tel.: +852 27889807; fax: +852 27888614.
E-mail addresses: cskylam@cityu.edu.hk (K.-Y. Lam), oulusoy@cs.bilkent.edu.tr (Ö. Ulusoy).

[1] A mobile client is a moving object, which can generate queries.

0164-1212/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2005.07.015

which is a kind of query that exists in the system for a pre-defined period of time (Sistla et al., 1997; Sistla et al., 1998; Gök and Ulusoy, 2000). Submitting a LQ as a CQ could be an efficient way to monitor the status of the interested objects such that once they meet the condition of the query, the requesting client is informed immediately.

Since the values of the data items which record the locations of moving objects can be highly dynamic (Sistla et al., 1997; Gök and Ulusoy, 2000; Wolfson et al., 1998; Wolfson et al., 1997; Trajcevski et al., 2002; Zhang et al., 2003), continuous updates refreshing the location information have to be installed to ensure the correctness[2] of the results of location-dependent continuous queries (LDCQs). However, this would impose a serious performance and wireless bandwidth overhead.

A common approach used to reduce the number of updates is indexing of the current and future positions of moving objects. Tayeb et al. (1998) used PMR-Quadtree for indexing moving objects. Saltenis et al. (2000) introduced the time parametrized R-Tree (TPR-tree) which considers both the position and velocity of moving objects. Later, the same authors proposed an R^*-tree based access method that indexes the current and future positions, assuming that the positions expire after specified time periods (Saltenis and Jensen, 2002). A B^*-tree based indexing method for moving objects was introduced by Jensen et al. (2004). Based on this method, the authors provided efficient algorithms for range and nearest-neighbor queries, as well as for continuous queries. Xia et al. (2005) proposed a Mean Variance Tree index structure, which is built based on the mean and variance of the data that are relatively constant features compared to the actual data values, leading to reduced index update cost.

The uncertainty of trajectory of a moving object in query processing was studied by Trajcevski et al. (2002). The authors used a cylindrical volume in 3D to model the trajectory of a moving object. Cheng et al. (2003) modeled a moving object's location with a closed region and a probability density function that describes the distribution of the location of the object within the region. Probabilistic queries were then defined, which augment probabilistic guarantees to query answers. Algorithms for evaluating probabilistic range queries and nearest-neighbor queries were also studied in that work.

In (Sistla et al., 1998; Trajcevski et al., 2002; Wolfson et al., 1999; Wolfson et al., 1997), some efficient dead-reckoning methods were proposed for generating location updates with the objectives of better utilization of the limited wireless bandwidth and bounding the degree of uncertainty. The main problem with the plain dead-reckoning (pdr) method is the difficulty in defining the right update threshold value for each moving object. In order to minimize the uncertainty of location-dependent query results, the update threshold of a moving object may be set to a small value. However, this may result in a heavy update workload. On the other hand, if large values are used for update thresholds, the uncertainty may be high and correct results may not be reported in a timely fashion to the requesting clients. To resolve the problem, the *adaptive dead-reckoning* (*adr*) was proposed as an extension to *pdr* (Sistla et al., 1997, 1998). In *adr*, the threshold is not fixed and a new threshold is provided with each update. The new value is computed based on the uncertainty cost, deviation cost, and update cost. The objective is to minimize the total information cost per time unit until the next update. Although the methods aim to minimize the "cost", the calculation of which is based on the information cost and update cost, they ignore the consideration of how this cost is "paid for" in case of providing incorrect query results to mobile clients. If a moving object is not involved in any queries, no information cost needs to be paid for since no one is interested in the location of the object. Therefore, it is not necessary to bound the uncertainty for all the objects as only some of them are involved in query results. Another problem is location update under disconnection.

Taking these considerations into account, we propose two new methods for generation of location updates. With the first method, which we call *adaptive monitoring method* (*AMM*), locations of the moving objects which already satisfy or will soon satisfy the condition of a LDCQ are monitored more closely. Location uncertainty of the other objects is allowed to be large. Therefore, it becomes possible to provide timely and correct results for LDCQs, while avoiding high update workload in the system. An important feature of AMM is that the update period of a moving object is adaptive to when the object will satisfy the condition of a LDCQ. This is more cost effective than using a fixed period for location update generation. The other method we introduce with the same objectives as AMM, which is called *deadline-driven method* (*DDM*), requires each moving object to send a location update to the location server whenever it is about to satisfy the condition of a LDCQ. We also discuss how disconnections are handled with AMM to minimize the probability of observing incorrect query results, and what the impacts of network disconnection on DDM are.

We have conducted extensive simulation experiments to evaluate the performance of our location update generation methods (AMM and DDM) as compared to a dead-reckoning method, and chosen *incorrect information rate* (i.e., an indicator of the degree of correctness

[2] Our definition of correctness is provided in Section 3.

of the results provided from the LDCQ), as the primary performance measure in our experiments. A related issue in processing LDCQs is determination of the time for transmitting query results to the originating clients (Gök and Ulusoy, 2000). Another contribution of our work is investigation of the performance relationship between the location update generation methods and the approaches used for location-dependent query result transmission, as these two issues are closely related to each other. In Section 2, we define our system model and specify the correctness requirements of the system. We introduce our update generation methods in Section 3. Section 4 is devoted to the performance evaluation of the proposed method. Finally, Section 5 concludes the paper.

2. System model

2.1. Model components

Our system model consists of a database server and a number of moving objects. The database server communicates with the moving objects using a low bandwidth mobile network. The server maintains a database, which contains data items for recording the locations of the moving objects. Data items associated with the moving objects are defined based on the *moving object spatio-temporal* (MOST) data model (Sistla et al., 1997; Sistla et al., 1998), in which the attributes of a moving data item can be static or dynamic. A static attribute changes only when an explicit update is applied. In contrast, a dynamic attribute changes over time according to a certain function. For example, each of the x and y coordinates of a moving object that specifies the position of the object in a two dimensional space, is a dynamic attribute. In the MOST data model, a dynamic attribute A is represented by three sub-attributes: $A.value$, $A.updatetime$ and $A.function$. $A.function$ is a function time (t) which has value 0 at $t = 0$. At time $A.updatetime$, the value of A is $A.value$. Thus, until the next update time, the value of A at time $A.time + t_0$ is given by $A.value + A.function(t_0)$. Under the MOST model, the results of an evaluation of a query will be a set of tuples with each tuple consisting of ⟨*object*, *begin time*, *end time*⟩. The *begin time* and *end time* of a tuple indicate the duration when the *object* satisfies the conditions of the query. The tuples for a query are ordered by their *begin times* and sent to the requesting mobile client, before their *begin time*, i.e., before the time when the values become true.

Moving objects generate updates to report their current locations to the database server through a mobile network. Each update is associated with a time-stamp, which specifies the time at which the current value will become valid. The time-stamp of an update is also recorded with the data value. Some moving objects (that we call mobile clients) may generate location-dependent queries on other objects in the system. Location-dependent queries are submitted as continuous queries each with a start time and an end time, such as LDCQ(*query_start_time*, *query_end_time*). The query is evaluated, and the results, in the form of a set of tuples ⟨*object*, *begin time*, *end time*⟩, are collected and grouped by their *begin times*, each indicating the beginning of the time period for which the *object* specified in the tuple satisfies the condition of the query. Once the results are ready, the server sends the selected tuples to the mobile client according to a query result transmission approach adopted by the system (Gök and Ulusoy, 2000). The query is re-evaluated when there is any change in the database state during the period of time between *query_start_time* and *query_end_time*.

2.2. Correctness requirements

The previous work on location updates generation is characterized by the objectives of bounding the location uncertainty and minimizing the update overhead. However, the issues on ensuring the *correctness* and *timeliness* of the results returned to the requesting clients have been ignored. In our work, we aim to maximize the correctness of the query results returned to mobile clients as they may take incorrect actions based on the incorrect results received. The correctness of a query result is affected by the uncertainty in locations of the objects selected as the result of the query. However, for those objects, which do not satisfy any query, or satisfy a query much ahead of the current time, the uncertainty in objects' locations does not need to be bounded as it will not affect the correctness of query results significantly.

Remember that with the MOST data model we adapted, the result of a LDCQ evaluation is specified as a set of tuples ⟨*object*, *begin time*, *end time*⟩. Tuples having an *end time* that is greater than the current time are sent to the client which has issued the query. We define the correctness of a query result based on its *begin time*. We also use the *actual* begin time of an object, which is the time when the object starts to satisfy the conditions of a query. Note that the actual begin time of an object for a query may deviate from the *begin time* produced for the corresponding result tuple of that query, because the database may contain out-dated information about the current location of the object. A mobile client observes incorrect result if:

(1) The actual begin time of an object for a query is earlier than the *begin time* of the result tuple produced for that object and current time is equal to the actual begin time, (we call this problem *missed information*); or

(2) The actual begin time of an object for a query is later than the *begin time* of the result tuple produced for that object and the begin time is equal to current time, (we call this problem *false information*).

In the first case, a mobile client is not aware of the fact that an object is satisfying the condition of its query. For the second case, a mobile client is informed that an object has met the condition of its query but actually it does not. Due to the delay in data transmission and processing, it is impossible to have an "instantaneous" location of a moving object. The degree of incorrectness in results is unknown to the server and the client since the server does not know the exact locations of the moving objects. Our performance objective is to minimize the difference between the actual and predicted actions. In practical systems, it is usually accepted that the recorded location of a moving object is considered to be the "*same*" as its current location if the deviation is very small, i.e., smaller than a pre-defined bound. We call this bound a *similarity bound* which is used to specify the accuracy of a query result. In this paper, we assume that each query is associated with a similarity bound. If the difference between a result tuple's *begin time* and the actual begin time is smaller than the similarity bound, the result is considered to be correct.

3. Location update generation methods

The biggest problem in location update for evaluation of LDCQ is the difficulty in determining when an object will satisfy the condition defined in the LDCQ. The problem is complicated by the variability in movement behavior of moving objects. An object may change its movement direction and speed dynamically. In this section, we present the *adaptive monitor method* (*AMM*) and the *deadline-driven method* (*DDM*) proposed for location update generation for moving objects with different movement behavior. The design principle of the method is based on the assumption that the next movement speed and direction are related to the current speed and direction. This principle is consistent with the movement behavior of most objects in real life application examples.

Both AMM and DDM are time based update schemes. Based on the *begin time* and *end time* of an object the server determines the next update time for the object with the aims to provide correct and timely query results to clients, and to minimize the location update cost. However, these two methods have different assumptions and are suitable for moving objects with different movement characteristics. AMM is a pessimistic approach. It adaptively monitors the progress of how the *begin time* and *end time* of an object change and determines the next update time using a feedback control scheme.[3] Each next update time is a fraction of the *begin time* of the object instead of the whole *begin time* as it assumes that the movement of the object may not follow exactly the prediction. On the other hand, the DDM is an optimistic approach as it assumes that the movement of the object will more or less follow the prediction. The next update is then determined mainly from the predicted *begin time* in the first estimation. These two methods are suitable for moving objects with different movement characteristics and they could impose very different update overhead. In general, AMM has a higher update cost but it can provide a closer monitor on movement of moving objects required by a query. Thus, it can provide query results with a higher degree of correctness while DDM has lower update cost but may provide a lower degree of correctness in query results.

3.1. Adaptive monitor method (AMM)

AMM is an adaptive update generation method. Similar to the dead-reckoning approaches, the generation of updates for moving objects in AMM depends on the deviation of object locations. An update threshold is defined for the location of each moving object. If the deviation between the actual and computed location values is greater than the update threshold, a location update is generated. As an improvement over defining a simple fixed update threshold for all the objects, we define *threshold bounds*, more specifically upper and lower threshold bounds, from which the actual update threshold of an object is evaluated based on some relevant characteristics of the object. The determination of the actual update threshold for an object is based on a mapping function, input of which is the *begin time* of the object for a LDCQ. The *upper threshold bound* can be a very loose (large) value and it is used for the objects whose location uncertainty can be allowed to be large. If the update generation follows the upper threshold bound, the total update workload will be low and will not significantly affect the system performance. The *lower threshold bound* is a tight (small) value and it is used for the objects which need close monitoring. If the update generation follows the lower threshold bound, every significant change in the location of moving objects is monitored so that the probability of losing track of their locations will be low.

AMM consists of three phases of execution in evaluating a LDCQ: (1) classification of moving objects into "selected" and "unselected" object sets on the basis of whether they satisfy the condition of a query; (2) determination of the update generation threshold for selected objects; (3) determination of the update generation

[3] This approach was first introduced in our earlier work (Lam et al., 2001).

threshold for the mobile client which has submitted the query.

3.1.1. Construction of selected and unselected object sets

In AMM, the moving objects in the system are classified into two sets for each query. A moving object is in the *selected object set* of a query if:

(1) It satisfies the condition of the query currently, or
(2) It will satisfy the condition in the future based on the predicted paths of both itself and the requesting client. Both paths are determined by using the *A.functions* (see Section 2).

Otherwise, an object is placed in the *unselected object set* of the query. For each selected object, a tuple is generated to be placed in the answer set of the query. We do not need to monitor the unselected objects closely and we can simply make their update thresholds equal to the upper threshold bound. For those objects in the selected set, we need to assign smaller update threshold values to monitor their locations closely. The smallest possible value that can be assigned is the lower threshold bound.

3.1.2. Update generation process of selected objects

Once we have determined the objects in the selected object set for a LDCQ, we need to specify the update thresholds of the objects in the selected object set. Each member in the selected object set has a tuple following the format of the MOST model, i.e., ⟨*object*, *begin time*, *end time*⟩. In AMM, an adaptive update generation approach is used in which the location update threshold of a moving object is set based on the *begin time* of its corresponding tuple for a query. The idea is that if the *begin time* of the object is close to the current time, the update threshold should be small. The aim is to monitor the movement of the objects closely if they will soon satisfy the conditions of the query. Otherwise, the probability of having incorrect results, in the form of either false information or missed information, will be high. For the moving objects whose *begin times* are far in the future, or are not even satisfying the condition of the query, the update thresholds are set to be close to the upper threshold bound with the aim of reducing the update workload. Allowing a larger uncertainty on the locations of these objects will not significantly increase the probability of having incorrect results.

Based on the principles described above, we may define a function to *map* the *begin time* of an object to its update threshold. Assuming an exponential distribution for the mapping function:

$$\text{Update threshold of object } x_i = H - (H - L) \times e^{-\delta t}, \text{where} \quad (1)$$

H: Upper update threshold bound; L: Lower update threshold bound; $\delta t =$ *begin time* of object x_i − current time.

As an example, suppose that a mobile client has submitted a query, and in the first evaluation of the query at current time 15, the following tuples are generated as the result of the query:

$\langle x_i, 12, 20\rangle$
$\langle x_j, 20, 28\rangle$
$\langle x_k, 25, 35\rangle$

The update thresholds of objects, x_i, x_j and x_k are determined by using the update threshold formula (1) as shown in Fig. 1. Since the *begin time* of x_i is smaller than the current time, its update threshold is set to be the value of the lower threshold bound to monitor its location closely. The update threshold of x_j is smaller than that of x_k since the *begin time* of x_j is closer to the current time than that of x_k. The computed update thresholds are sent to the corresponding objects and the generation of updates follows these threshold values. Whenever the database server receives an update from the moving object or the client which has generated the LDCQ, the query is evaluated again and the new 7*begin times* and *end times* of the result tuples are redefined. Then, the new update thresholds are calculated and sent to the corresponding moving objects. When an object receives a new update threshold, it compares the new threshold with the amount of distance traveled since the last location update. If this distance is greater than the threshold value, the object generates a location update.

The determination of the mapping function is application dependent and is also affected by the movement characteristics of the objects. Different mapping functions can be adapted for generating the update thresholds of different objects which are accessed by different types of queries. The guideline for choosing the mapping functions and setting the threshold bounds would be the

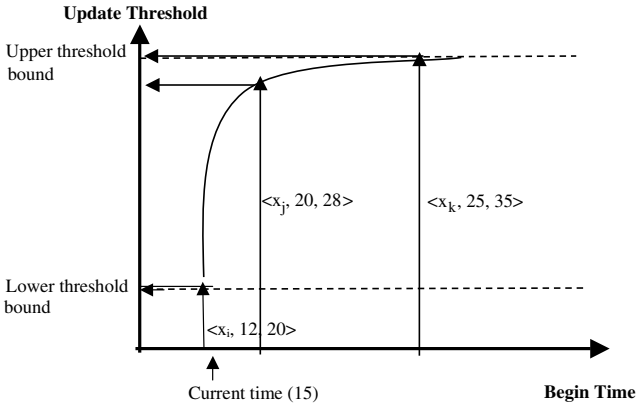

Fig. 1. Generation of update thresholds in AMM.

consideration of the cost of missing the information to the clients and the total update cost, as well as the accuracy of prediction in movement of objects. If the cost of missing the information to the clients is low, a large threshold bound may be used. On the other hand, if the total update workload in the system is low, smaller threshold bounds may be defined. If the movement of objects follows closely the prediction, i.e., the *begin time* does not change much after each location update, an exponential mapping function with a small mean value can be employed, so that a small update threshold is used only for cases where the *begin time* is very close to the current time.

A moving object might be in the answer set of more than one LDCQ. In that case, the update threshold of the moving object is set to be the minimum of all the thresholds calculated from all those queries. If an object's *begin time* is earlier than the current time in the first evaluation, its update threshold is set to be the lower update threshold bound. When a location update arrives, the set of related queries needs to be re-evaluated. The update arrival rate could be high if the system consists of a large number of moving objects selected by the queries. Therefore, it is important to minimize the computation cost for them. A simple way to reduce the evaluation cost and the cost for calculating the next update time is to use a linked list to connect the set of objects accessed by a query. For example, assume that the set of objects selected by queries Q_x and Q_y are $\{O_i, O_j, O_k\}$ and $\{O_i, O_l\}$, respectively. The object O_i is selected by both Q_x and Q_y. When an update from O_i arrives at the system, both Q_x and Q_y are evaluated following their links, and new *begin time* and *end time* for O_i are computed. Other queries will be unaffected. A query is removed from the system after its *end time*.

When a location update of a requesting client arrives at the database server, the server may find out that the actual location of the requesting client differs by δd from the predicted value determined using its *A.function*. In this case, all the *begin time*s of the selected tuples need to be adjusted by the difference, δd. Some of the already selected objects may be excluded from the selected object set, while some new objects may be included in the set. In order to minimize the impact of the mobility on the correctness of query results, active mobile clients generate updates more frequently (i.e., the lower update threshold bound is used), so that their locations can be closely monitored and any change in the evaluation results can be identified earlier.

3.2. Deadline-driven method (DDM)

When a moving object sends a location update to the central database server, the answer tuple of each LDCQ referring to that object needs to be updated. The *begin* and *end times* of each such tuple may change. The tuples of LDCQs affected by a location update have to be recomputed to prevent the supply of incorrect information to the clients. The processing overhead in the system depends on the number of location updates. If the accuracy in prediction on movement of a moving object is high, some location updates may be unnecessary. Consider the example in Fig. 2. If the movement of the objects and the clients which initiate the LDCQ follows prediction (i.e., its *begin time* remains the same), the location update thresholds of that object will be getting smaller and smaller as the *begin time* approaches. The location updates which are far away from the *begin time* will not affect the correctness of query results (Fig. 2).

Taking this observation into account, we can say that location update may not be necessary for an object if it is not currently satisfying or about to satisfy any query and its movement follows prediction most of the time. Following this principle would also reduce the overhead of location updates. Therefore, if the *begin time* of a moving object in a result tuple is far from the current time and its movement is quite regular most of the time, we may set the next location update time "close" to the *begin time*.

Since a moving object knows its deviation from its last reported position and its previous movement pattern, it can determine when it will satisfy the condition of a LDCQ by assuming that the requesting client is stationary. Simply just before satisfying the query, the object can generate an update to report its current location to the database server, so that the server can find out that it will soon meet the condition of the query. In determining the update generation time, the offset of time, t_{os}, before the condition is met can be computed by summing up the time for sending the update to the server, the time for performing the update and the time for evaluating the query. By using this offset, we can define a deadline for the next update generation time of a moving object as shown in Fig. 3:

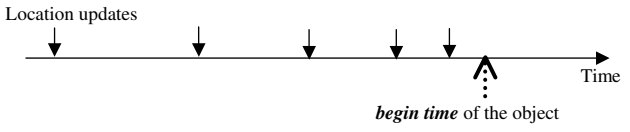

Fig. 2. An example to represent unnecessary location updates.

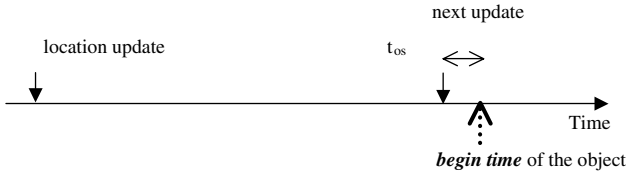

Fig. 3. Determining update generation time in DMM.

$$\text{Deadline} = f(\delta t - t_{os})$$

where δt = *begin time* of the tuple − current time.

The moving object has to generate an update before the deadline.

In the deadline formula given above, the movement of the requesting client of the query is not considered at all. Therefore, every time the requesting client sends an update on its location to the central database server, the server should re-calculate *begin* and *end times* for the result tuples and forward the modified values to the moving objects. A new deadline is then computed using the modified *begin time*. If a moving object is involved in more than one query, the deadline for its next update generation is set to the closest deadline from all the queries.

3.3. Impact of disconnection on AMM and DDM

An important characteristic of a mobile network is frequent disconnection of mobile computers connected to it. A disconnection might be either voluntary or involuntary. A voluntary disconnection aims to save the limited computer and network resources. Involuntary disconnection is mainly due to poor network services. In the following, we discuss how the AMM method deals with the involuntary disconnection problem.

The major problem of network disconnection is that the transmission of location updates from disconnected moving objects to the database server would not be possible. Consequently, it would become more likely to issue missed and/or false information to the clients. Since it is impossible to prevent disconnection (which is a communication issue), the main solution here is to inform the requesting clients and the database server that a moving object is disconnected and the deviation from its actual location may be greater than its update threshold bound. A query result involving the moving object may not be reliable.

The most trivial way to detect disconnection is to check regularly for disconnection by trying to communicate with the moving objects. However, this method increases the communication workload on the low bandwidth wireless channel especially if the checking period is set to be small. In AMM, communication between the central database server and the set of selected moving objects already occurs for location update transmission (from moving objects to the database server) and update threshold transmission (from the database server to moving objects). These messages can also be used to avoid unnecessary communication simply for checking of disconnection. If we assume that a moving object is traveling with a constant speed, the duration of time between the successive updates, called update period, will be proportional to the value of the update threshold. Therefore, the next location update of a selected moving object is done, either:

(1) when the deviation in location is greater than its update threshold, or
(2) when the current time becomes equal to the last update time + the current update period;

whichever is earlier.

The value of the current (say, ith) update period P_i of a moving object is determined by the following formula: $P_i = P_{i-1} \times U_i / U_{i-1}$, where U_i is the ith update threshold. The calculation of P_i is performed at the same time as the calculation of the update threshold U_i; and P_i and U_i are sent together to the moving object so that the object will know when it should generate the next update. If a server does not receive an update from a moving object after the expiration of its update generation time, this implies that the moving object is disconnected from the network. Note that the server only needs to check the update time of selected objects. For those objects which are not selected by any queries, they are not required to update their locations and their disconnection from the network will not affect the query results.

The problem of disconnection in DDM is more serious if a disconnection occurs when the deadline of a moving object is approaching, i.e., when a moving object decides to send its location update to the database server. This can cause the deadline of the object be missed. Thus, to overcome the disconnection problem in DDM, the offset of time can be set to a more pessimistic value such that it includes the re-transmission delay in case of disconnection.

4. Performance evaluation

4.1. Simulation model

We have designed and implemented a detailed simulation model to study the performance of the proposed methods, AMM and DDM, as compared to plain dead-reckoning (*pdr*) by employing different query result transmission strategies. Our simulation model is based on the performance models proposed in the previous related works such as (Gök and Ulusoy, 2000). These models have been extended to support modeling of processing LDCQs. As shown in Fig. 4, our simulation model consists of three basic components: mobile client model, communication network and server model.

Each mobile client in our model consists of three modules: a resource manager, a continuous query generator and an update generator. The resource manager models the CPU at the client machine for processing query requests and handling the query results sent by the server. It is assumed that there are TotalMO moving

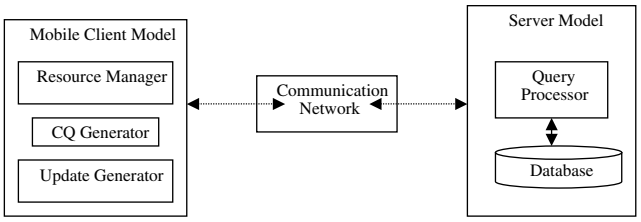

Fig. 4. The simulation model.

objects among which NumMH objects can generate LDCQs. The continuous query generator generates LDCQs which are sent to the server model through the communication network. The results of a query evaluation are returned to the requesting mobile client from the server also through the communication network. The lifetime of a LDCQ is chosen randomly between the values MinCQLife and MaxCQLife. A mobile client generates a new LDCQ after a think time following the completion of its previous LDCQ. The think time is exponentially distributed with a mean of ThinkTime. No I/O time is modeled in the resource manager module since we assume that the buffer pools of mobile clients are large enough to hold all the tuples received in response to an issued LDCQ.

In the simulation model, each moving object is assigned a default speed, S. Every time unit, the distance traveled by a moving object is calculated by using the speed $S \pm \mathrm{SF}$ where SF is a random variable uniformly distributed within a bound called speed bound (SB). Whenever the deviation of the location of an object becomes greater than the update threshold of the object, the update generator generates a location update to the database server. When the database server receives a new location of a moving object, it re-calculates the *begin time* of the query result tuples referring to the object. We assume that the change in the *begin time* of a tuple is inversely proportional to the distance traveled by the corresponding object. For example, if the distance traveled by an object is more than expected, the *begin time* will be updated to be earlier; and vice versa. We further assume for simplicity that the change in the *begin time* is proportional to the difference between the actual and expected values of the distance traveled.

All the messages exchanged between mobile clients and the server are carried by the communication network. It takes TupleTime seconds to transmit each tuple in the query result set and ControlMessageTime seconds to transmit each control message. For the transmission of the result tuples of a query, a control message is generated which contains the necessary information for controlling the transfer, such as the identification of the requesting client and the total number of tuples being transmitted.

The server model consists of a query processor and a database. The query processor processes LDCQs and location updates received from mobile clients. The server also controls the accesses to the server CPU and to the central database. No I/O time is modeled with the server as we assume that it contains a fast accessed secondary memory. Once the query processor receives a LDCQ, it determines the size of the query result in terms of the number of tuples in the answer set of the query. The maximum size of a query result is specified by the parameter CQSize. The result of a LDCQ is generated as a set of tuples $\langle object, begin\ time, end\ time\rangle$. The tuples are sent to the requesting mobile client in increasing order of their *begin times*. Both delayed and periodic transmissions (Gök and Ulusoy, 2000) of tuples are simulated in the model for transmitting tuples to the clients. With the *delayed transmission approach*, a tuple $\langle object, begin\ time, end\ time\rangle$ is transmitted to the mobile client just before the *begin time* (considering required communication and processing delays). According to the *periodic transmission approach*, at each w time units, all the tuples $\langle object, begin\ time, end\ time\rangle$ satisfying the condition $t \leqslant begin\ time < t + w$ where t is the current time, are transmitted to the mobile client which has issued the LDCQ. w is called the window size.

4.2. Simulation model parameters and performance metrics

Table 1 lists the simulation model parameters and their baseline settings. The baseline parameter values were chosen so as to be comparable to the previous related simulation studies, such as (Gök and Ulusoy, 2000; Bukhres and Jing, 1996; Leong and Si, 1997). In order to study the comparative performance of the proposed methods, we measure the *incorrect information rate* (*IIR*) which is defined as the number occurrences of false information and missed information (as defined in Section 3) over the total number tuples generated. IIR indicates the capability of the system in providing correct information (data values) to the queries from mobile clients. In addition to IIR, we also measure the re-transmission rate, control message overhead and update workload. As explained in Section 2, when a window based method is used for transmitting the query result tuples to its requesting client, some of the tuples may need to be re-transmitted due to the changes in the result. *Re-transmission rate* is defined as the total number of tuple re-transmissions over the total number of tuples transmitted. *Control message overhead* measures the total number of control messages per unit time. It is used to evaluate the communication overhead for location updates between mobile clients and the server. Note that different methods for location updates transmission, i.e., deferred and periodic, incur different control message overheads. *Update workload* measures the proportion of CPU utilization for processing location

Table 1
Simulation parameters and default values

	Baseline value
Mobile client parameters	
Total number of moving objects (TotalMO)	300
Number of moving objects which may generate LDCQ (NumMH)	50
Number of objects satisfying a LDCQ (CQsize)	10–20 objects
Minimum life of a LDCQ (MinCQLife)	240 s
Maximum life of a LDCQ (MaxCQLife)	360 s
Think time (ThinkTime)	1000 s
Speed bound (SB)	0–1 (for each time unit in the simulation)
Communication network parameters	
Time for sending a tuple (TupleTime)	0.1–0.2 s (normal distribution)
Time for sending a control message (ControlMessageTime)	0.05–0.1 s (normal distribution)
Server parameters	
Time for processing a tuple (ComputeTime)	0.05–0.1 s (normal distribution)
Update threshold limits	5–30 s
Window size for transmitting the query results	50 s
Similarity bound	10 s

updates of moving objects. It specifies the processing load introduced by location updates.

4.3. Performance results

The simulation program was implemented in CSIM-18, which is a simulation language based on the C programming language. We have performed four sets of experiments to compare the performance of AMM and DDM with plain dead-reckoning (pdr). In the first two sets of experiments, the upper update threshold bound of AMM is fixed at 30 s, and the lower threshold bound is varied from 2.5 to 30 s. The first set of experiments compares the performance of AMM and DDM with pdr under the delayed transmission method for transmitting LDCQ results, while the second set of experiments compares the performance of the three methods using the periodic transmission method with a window size of 50 s. Delayed and periodic transmission approaches are explained in Section 4.4. In the third set of experiments, the upper and lower threshold bounds are set to 10 and 30 s respectively for AMM and the update threshold for pdr is set to 10 s. The performance results of the three methods are compared under varying numbers of mobile objects. In the last set of experiments, the performance of AMM is investigated under different upper threshold bounds.

Fig. 5 depicts the incorrect information rate (IIR) results when different values are used for the lower threshold bound for AMM and the update threshold for pdr. The delayed transmission method is used for transmitting query results to mobile clients. It can be seen from the figure that the performance of AMM is consistently better than pdr, i.e., the IIR of AMM is considerably smaller than that of pdr for different threshold values. Consistent with our expectation, both curves are in V-shape. The best performance is achieved when the threshold value is around 12.5 s. The poor performance with small threshold values is due to heavy update workload as can be observed in Fig. 6 in which the update workload is close to 65% for AMM and 90% for pdr. Thus, under such settings, most of the system resources are devoted to process the location updates from the mobile objects and the system will not be able to generate timely responses to the continuous queries from mobile clients. When a large threshold is used, the degree of uncertainty in the location of a moving object will be high. Thus, the probability of providing incorrect information to mobile clients will also be high although the update workload is smaller.

The better performance of AMM is due to the better monitoring scheme used in generating location updates in AMM, i.e., a smaller update threshold is assigned

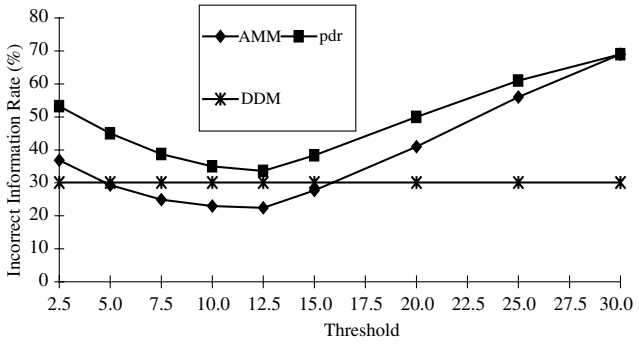

Fig. 5. The impact of lower threshold limit on incorrect information rate under delayed transmit.

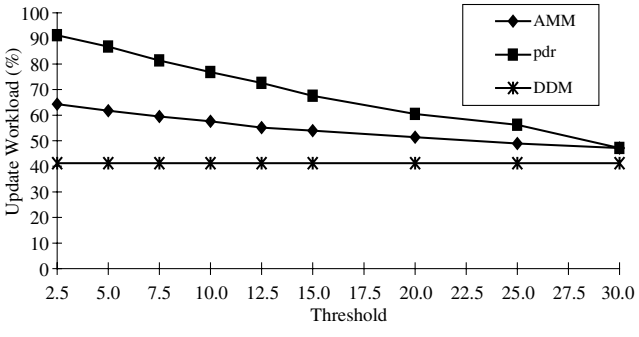

Fig. 6. The impact of lower threshold limit on update workload under delayed transmit.

to the moving object if it is currently satisfying or will soon satisfy the condition of a query. Even though determining the update thresholds in AMM requires communication between moving objects and the server, the total number of control messages required in AMM is still smaller than that in pdr due to smaller number of location updates. The lower levels of control message overhead with AMM can be observed in Fig. 7.

As presented in Fig. 5, the performance of DDM is better than AMM when the update threshold value is very small or when this value is large. This is mainly due to smaller update workload with DDM since it eliminates the location updates for those objects which are far from satisfying the condition of a query. With AMM, when the update threshold is very small, the update workload will be heavy, while a large update threshold will lead to a high level of uncertainty.

For the resulting tuples which were identified as incorrect, we have also observed the distribution of the difference between the *begin time* of a tuple generated for an object and the actual time the object starts to meet the condition of the query. Table 2 presents the error distribution results obtained with AMM when the lower threshold limit is set to 15 s. Similar behavior was observed with all the other settings of the threshold limit, and the error distribution presented in the table was chosen as representative. The results presented show that the deviation between the predicted and actual values for the incorrect results is on the average 15.5 s, with a maximum of 23 s. Remember that if the deviation is within the similarity bound, which was set to 10 s in our experiments, the query result is considered to be correct. Therefore, the average error observed in incorrect result tuples is about 5.5 s. The error distribution can be regarded as a normal distribution with about 3.1 s standard deviation.

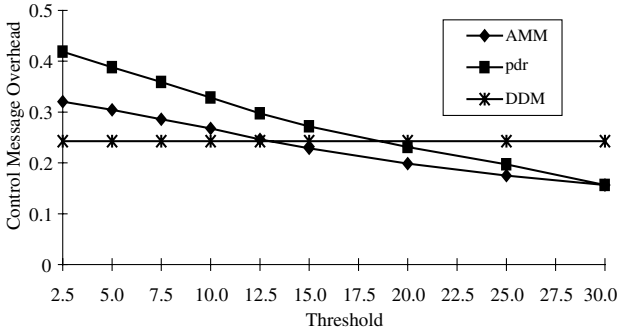

Fig. 7. The impact of lower threshold limit on control message overhead under delayed transmit.

In the second set of experiments, periodic transmission of query results is employed with a window size of 50 s (please see Section 4.4 for the description of this method). The results obtained are shown in Figs. 8–11. Consistent to the results obtained in the previous experiments, the performance of AMM is consistently better than that of pdr, and DDM is better than AMM when the update threshold of AMM is very small or large. Comparing the results displayed in Fig. 8 with those in Fig. 5, we can see that the performance of all three methods is improved when the periodic transmission method is used. Although under the periodic transmission, retransmissions are required once a tuple result changes after it has been transmitted to the requesting client, the total control message overhead is still smaller than the case with delayed transmission. The lower control message overhead can be observed by comparing the results shown in Fig. 11 with those in Fig. 7. As shown in Figs. 9 and 10, the better performance of AMM as compared to pdr is also due to the smaller update workload and re-transmission rate experienced with AMM.

In the third set of experiments, the impact of the number of moving objects on the performance of the three methods is investigated. The lower and upper threshold bounds for AMM are fixed at 10 s and 30 s, respectively. The update threshold for pdr is set to 10 s. The results are displayed in Figs. 12–15. As expected, IIR increases with the increasing number of moving objects due to heavier workload and control message overhead as shown in Fig. 12. The performance of both AMM and DMM is consistently better than that of pdr, again due to much smaller update workload, retransmission rate and control message overhead as shown in Figs. 13–15, respectively. As the number of mobile objects increases, IIR also increases as shown in Fig. 12. Although the update workload of AMM is similar to that of pdr when the number of moving objects is large, i.e., the number of moving objects is close to 500, the performance of AMM is still significantly better than that of pdr. This result is due to the better monitoring scheme used in AMM to generate location updates.

In the last set of experiments, we investigate the performance impact of the upper threshold bound of

Table 2
Error distribution observed for incorrect result tuples with the lower threshold limit of 15 s

Error (s)	11	12	13	14	15	16	17	18	19	20	21	22	23
%	9	11	10	13	13	10	7	8	5	6	5	2	1

The values presented in the first row correspond to the difference between the begin time of a tuple generated for an object and the actual time the object starts to meet the condition of the query. Each value in the second row corresponds to the approximate percentage of the incorrect tuples with the error value specified above it.

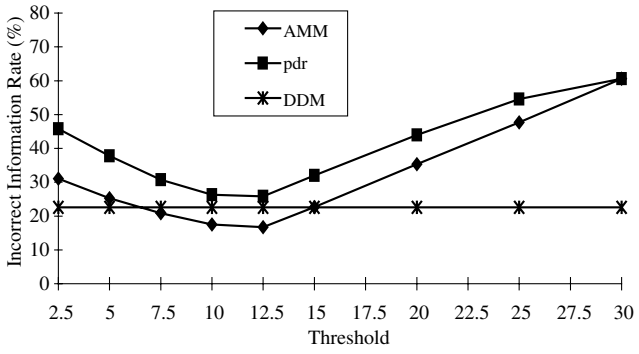

Fig. 8. The impact of lower threshold limit on incorrect information rate under periodic transmit with window size of 50 s.

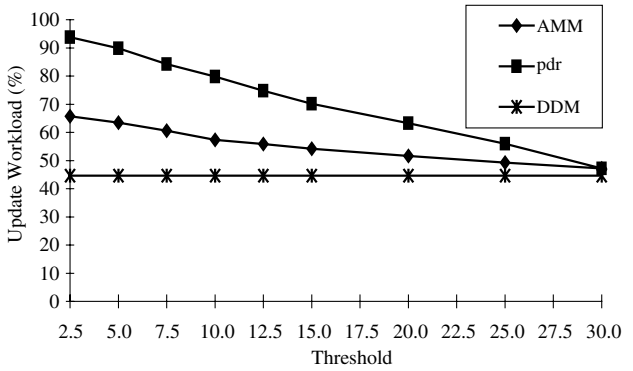

Fig. 9. The impact of lower threshold limit on update workload under periodic transmit with window size of 50 s.

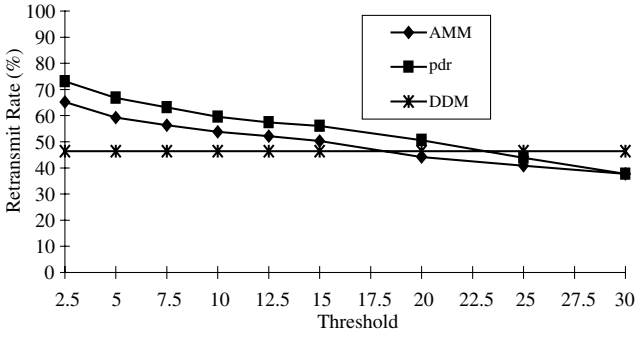

Fig. 10. The impact of lower threshold limit on re-transmit rate of LDCQ results under periodic transmit with window size of 50 s.

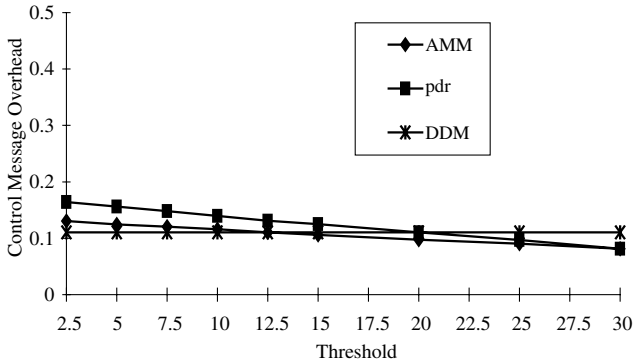

Fig. 11. The impact of lower threshold limit on control message overhead under periodic transmit with window size of 50 s.

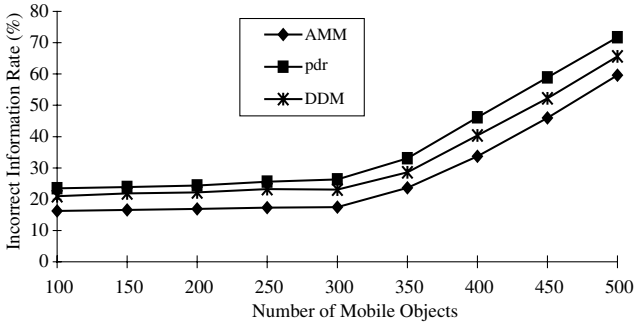

Fig. 12. The impact of number of mobile objects on incorrect information rate under periodic transmit with window size of 50 s.

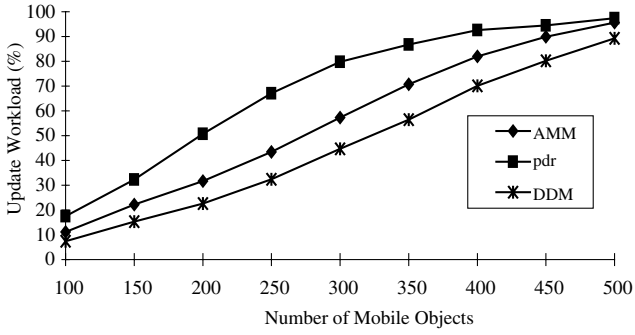

Fig. 13. The impact of number of mobile objects on update workload under periodic transmit with window size of 50 s.

AMM. The results obtained are displayed in Figs. 16–18. The curves labeled 'AMM-20', 'AMM-30', 'AMM-40', 'AMM-50' and 'AMM-75' indicate the performance when the upper update threshold is set to 20, 30, 40, 50 and 75, respectively. In all three figures, the results are displayed as a function of the lower threshold bound which is varied from 2.5 to 30 s. As shown in Fig. 16, the best performance is achieved when a medium upper threshold bound is chosen (i.e., with AMM-30 and AMM-40). If a small upper threshold bound is used, the update workload becomes heavy as shown in Fig. 17. When a large upper threshold bound is used, the uncertainty in the locations of the moving object becomes larger although the update overhead and the control message overhead becomes lower (see Figs. 17 and 18). The update thresholds of objects become tight when a small upper bound (i.e., AMM-20) is used, and IIR may become greater, even larger than that of pdr. It is also observed that, the system performance with a large

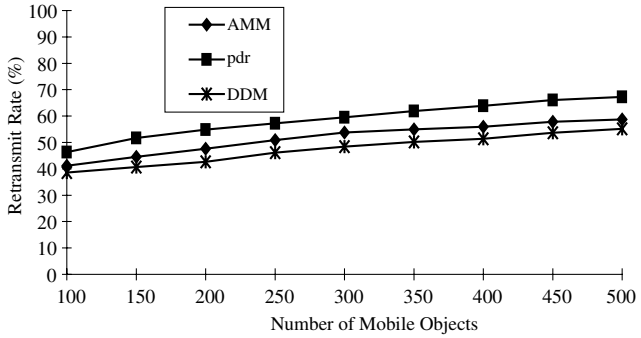

Fig. 14. The impact of number of mobile objects on re-transmit rate of LDCQ results under periodic transmit with window size of 50 s.

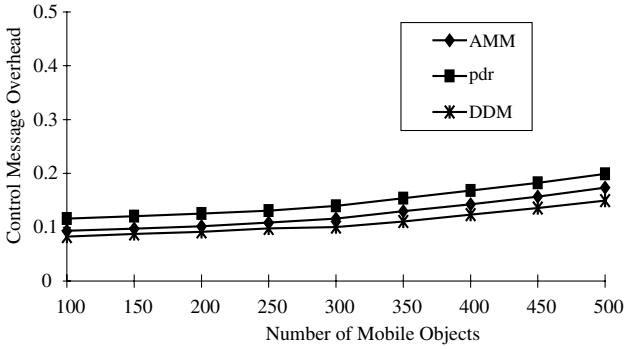

Fig. 15. The impact of number of mobile objects on control message overhead under periodic transmit with window size of 50 s.

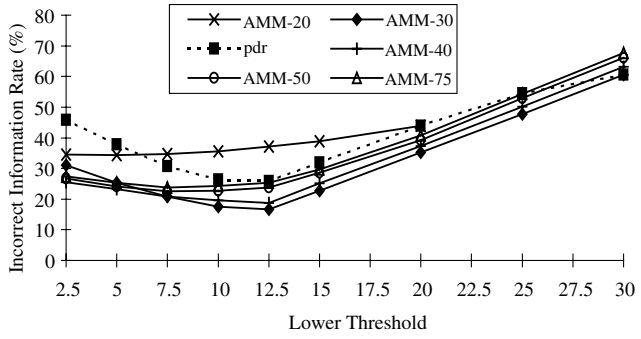

Fig. 16. The impact of upper threshold limit on incorrect information rate under periodic transmit with window size of 50 s.

upper threshold bound is better at a small lower threshold bound. This result can be explained by the fact that, with a small lower threshold bound, the thresholds for mobile objects are small enough to keep an effective update frequency.

4.4. Discussion

Under the delayed transmission method for transmitting LDCQ results, the incorrect information rate (IIR) of AMM was observed to be consistently smaller than

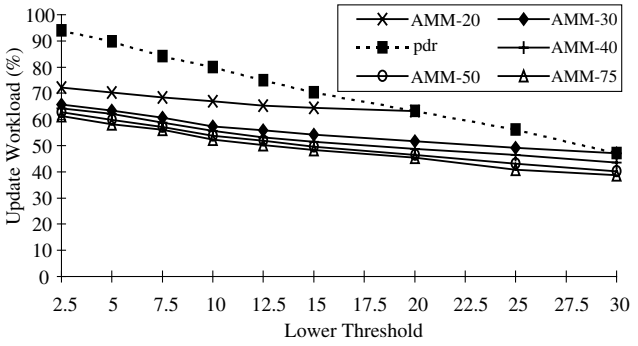

Fig. 17. The impact of upper threshold limit on update workload under periodic transmit with window size of 50 s.

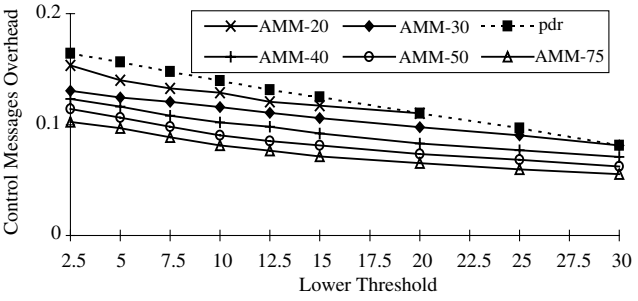

Fig. 18. The impact of upper threshold limit on control message overhead under periodic transmit with window size of 50 s.

that of pdr for different threshold values, due to the effective monitoring scheme used in generating location updates. DDM can outperform AMM only when the update threshold value is very small or very large. DDM eliminates the location updates for those objects which are far from being a target of a query. With AMM, when the update threshold is very small, the update workload becomes heavy, while a large update threshold leads to a high level of uncertainty.

All the methods evaluated (i.e., AMM, DDM and pdr) perform better when the periodic method is used for the transmission of query results. This result is due to the lower level of control message overhead experienced with this transmission method. When we evaluated the performance impact of the number of moving objects, we observed that the rate of incorrect query results increases with the increasing number of moving objects due to the heavier workload and control message overhead involved. AMM still performs better than the others, although its update workload becomes as high as that of the other methods when the number of moving objects is large.

It was also shown through the experiments that the upper threshold bound used for location updates should not be very small (as this leads to heavy update workload) or large (since this results in high levels of uncertainty in location information).

5. Conclusions

An important issue that needs to be considered in designing a mobile computing system is to supply the system with the capability of processing location-dependent continuous queries. Data values used for maintaining locations of moving objects are highly dynamic and may possess real-time properties. Location-dependent queries from mobile clients may also be associated with some constraints on response time. In this paper, we have analyzed the issues related to the generation of location updates from moving objects in a mobile computing environment. Two new methods, called adaptive monitor method (AMM) and deadline-driven method (DDM) have been proposed with the aim of maintaining the correctness of query evaluation results without increasing the location update cost. With the AMM method, the objects which are going to be included in a query result soon are monitored more closely. Update thresholds used for generating updates for the location of moving objects are computed based on how close the objects are to begin satisfying the condition of a query. DMM is an extension of AMM proposed with the aim of further reducing the amount of location updates. With this method, each moving object sends a location update only when it is about to satisfy the condition of a query. In general, it can reduce the location update overhead if the movements of the objects follow the prediction most of the time while AMM is suitable for the systems where disconnection is frequent. Extensive simulation experiments have been conducted to investigate the performance of the proposed methods as compared to a well-known location update generation method, the plain dead-reckoning (pdr). The simulation results show that the performance of AMM and DDM is significantly better than that of pdr under different settings of system parameters. The relative performance of DDM and AMM is determined by the update threshold values set with AMM.

References

Bukhres, O., Jing, J., 1996. Performance analysis of adaptive caching algorithms in mobile environments. Information Sciences 95 (1), 1–27.

Cheng, R., Prabhakar, S., Kalashnikov, D.V., 2003. Querying imprecise data in moving object environments. In: Proceedings of International Conference on Data Engineering (ICDE'03), Bangalore, India, pp. 723–725.

Dunham, M.H., Kumar, V., 1998. Location dependent data and its management in mobile databases. In: Proceedings of International Workshop of Database and Expert Systems Applications, Vienna, Austria, pp. 414–419.

Gök, H.G., Ulusoy, Ö., 2000. Transmission of continuous query results in mobile computing systems. Information Sciences 125 (1–4), 37–63.

Jensen, C.S., Lin, D., Ooi, B.C., 2004. Query and update efficient B+-tree based indexing of moving objects. In: Proceedings of International Conference on Very Large Databases (VLDB'04), Toronto, Canada, pp. 768–779.

Lam, K.Y., Ulusoy, Ö., Lee, T.S.H., Chan, E., Li, G., 2001. An efficient method for generating location updates for processing of location-dependent continuous queries. In: Proceedings of International Conference on Database Systems for Advanced Applications, Hong Kong, pp. 218–225.

Lee, D.L., Lee, W.C., Xu, J., Zheng, B., 2002. Data management in location-dependent information services. IEEE Pervasive Computing 1 (3), 65–72.

Leong, H.V., Si, A., 1997. Database caching over the air storage. The Computer Journal 40 (7), 401–415.

Ren, Q., Dunham, M.H., 2000. Using semantic caching to manage location dependent data in mobile computing. In: Proceedings of International Conference on Mobile Computing and Networking, Boston, MA, USA, pp. 210–221.

Saltenis, S., Jensen, C.S., 2002. Indexing of moving objects for location-based services. In: Proceedings of International Conference on Data Engineering (ICDE'02), San Jose, CA, USA, pp. 463–472.

Saltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A., 2000. Indexing the positions of continuously moving objects. In: Proceedings of ACM SIGMOD International Conference on Management of Data SIGMOD'00, Dallas, Texas, USA, pp. 331–342.

Seydim, A.Y., Dunham, M.H., Kumar, V., 2001. Location dependent query processing. In: Proceedings of ACM International Workshop on Data Engineering for Wireless and Mobile Access , Santa Barbara, CA, USA, pp. 47–53.

Sistla, A.P., Wolfson, O., Chamberlain, S., Dao, S., 1997. Modeling and querying moving objects. In: Proceedings of International Conference on Data Engineering, Birmingham, UK, pp. 422–432.

Sistla, A.P., Wolfson, O., Chamberlain, S., Dao, S., 1998. Querying the uncertain position of moving objects. Temporal Databases: Research and Practice. Lecture Notes in Computer Science Springer Verlag, vol. 1399, pp. 310–337.

Tayeb, J., Ulusoy, Ö., Wolfson, O., 1998. A quadtree based dynamic attribute indexing method. The Computer Journal 41 (3), 185–200.

Trajcevski, G., Wolfson, O., Xu, B., Nelson, P., 2002. Real-time traffic updates in moving objects databases. In Proceedings of International Workshop on Mobility in Databases and Distributed Systems, Aix-en-Provence, France, pp. 698–704.

Trajcevski, G., Wolfson, O., Zhang, Fengli, Chamberlain, Sam, 2002. The geometry of uncertainty in moving objects databases. In Proceedings of International Conference on Extending Database Technology (EDBT'02), Prague, Czech Republic, pp. 233–250.

Wolfson, O., Sistla, P., Chamberlain, S., Yesha, Y., 1999. Updating and querying databases that track mobile units. Distributed and Parallel Databases 7 (3), 257–287.

Wolfson, O., Xu, B., Chamberlain, S., Jiang, L., 1998. Moving objects databases: issues and solutions. In: Proceedings of International Conference on Scientific & Statistical Database, Capri, Italy, pp. 111–122.

Wolfson, O., Chamberlain, S., Dao, S., Jiang, L., 1997. Location management in moving objects databases. In: Proceedings of International Workshop on Satellite-Based Information Systems, Budapest, Hungary, pp. 7–13.

Xia, Y., Prabhakar, S., Lei, S., Cheng, R., Shah, R.l., 2005. Indexing continuously changing data with mean variance tree. In: Proceedings of ACM Symposium on Applied Computing (SAC'05), Santa Fe, New Mexico, USA, pp. 1125–1132.

Zhang, J., Zhu, M., Papadias D., Tao Y., Lee D.L., 2003. Location-based spatial queries. In: Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'03), San Diego, CA, USA, pp. 443–454.