An Efficient Method for Generating Location Updates for Processing of Location-Dependent Continuous Queries

Kam-Yiu Lam[1], Özgür Ulusoy[2], Tony S.H. Lee[1], Edward Chan[1] and Guohui Li[3]

Department of Computer Science[1]
City University of Hong Kong
83 Tat Chee Avenue, Kowloon, Hong Kong

Department of Computer Engineering[2]
Bilkent University
Ankara, TURKEY

School of Computer Science.& Technology[3]
Huazhong University of Scie.& Tech.
Wuhan, Hubei, P. R. China

Email: {cskylam|csedchan}cityu.edu.hk, oulusoy@cs.bilkent.edu.tr

Abstract

Recent advances in mobile computing and mobile communication technology have led to the emergence of many innovative mobile-computing applications. Some of them require providing support to location-dependent continuous queries (LDCQs) on moving objects. The result of a location dependent query depends on the current locations of the moving objects. When the query is specified as continuous, the requesting client can get continuously changing result. In order to provide correct and timely results to requesting clients, the locations of moving objects have to be closely monitored. In this paper, we propose an adaptive monitoring method (AMM) for managing the locations of moving objects to maintain the correctness of the results of query evaluation without significantly increasing the wireless bandwidth requirements. Extensive simulation experiments have been conducted to investigate the performance of the proposed method as compared to the plain dead-reckoning (pdr).

Keywords: Mobile computing, moving object database, location management, update generation and data similarity

1 Introduction

Various innovative mobile computing applications are emerging as a result of the advances in mobile communication and portable computing devices. Many of these applications require to manage the locations of moving objects and to support the so-called *location-dependent queries (LDQs)* on moving objects [DK98]. The evaluation result of a LDQ depends on the location of the originating mobile client[1]. An example of LDQ submitted by the driver of an ambulance might be: "identify all other ambulances, which are within 5 km of my current position". The result of the query depends on the current location of the ambulance.

A LDQ can become much more complex to process if it is submitted as a *continuous query (CQ)* which is a query that exists in the system for a period of time [GU00]. Within the specified period, a CQ is evaluated continuously and the query results are transmitted to the originating mobile client from time to time. Submitting a query as a continuous query can 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 will be informed immediately.

The traditional database technology is not sufficient for processing of *location-dependent continuous queries (LDCQs)* [AFZ97, SWCD97]. In order to process LDCQs efficiently, a data model called *Moving Objects Spatio-Temporal (MOST)* was proposed [SWCD97, SWCD98]. In the model, a location prediction function is defined as a dynamic attribute of a moving object to predict the future locations. With the MOST data model, the result of the evaluation of a LDCQ is a set of tuples *<object, begin time, end time>*, where *begin time* and *end time* define the time bounds when the *object* satisfies the condition of the query. The *end time* is greater than the current time.

It is clear that in order to provide correct and timely results to mobile clients, one of the most important issues is to monitor the locations of all the moving objects including those which generate LDCQs frequently. Since the objects are moving, the values of the data items, which record the current locations, can be highly dynamic [SWCD97, GU00, WXCJ98, WCDJ97]. It is obvious that this would impose a serious performance overhead to the wireless bandwidth. On the other hand, if generations of updates are not frequent enough, the uncertainty of the locations of the moving objects will be high and the correctness of the results of a LDCQ returned to the requesting mobile client cannot be guaranteed. In [SWCD98, WSCY99, WCDJ97], some efficient dead-reckoning methods were proposed for generating updates with the objectives to better utilize the limited wireless bandwidth and to bounding the degree of uncertainty.

After processing a query, in order to reduce the data transmission overhead, the entire set of objects corresponding to the query result at different times may be sent immediately to the requesting client. However, it comes the problem of data re-transmission if the position of any of the objects in the result set changes after it has been sent to the mobile client. In [GU00], various methods for query result transmission were investigated for the systems using the MOST data model with the objective of minimizing data transmission cost. However, up to now, it is completely lack of an integrated study on the performance relationships between the methods used for update generation and query result transmission although these two issues are closely related to each other.

[1] A mobile client is a moving object. When we refer to it as a mobile client, it means that it can generate queries.

In this paper, we present a detailed simulation model with which both the update generation and query result transmission issues can be investigated. Based on the dead-reckoning approaches, we have designed a new method to monitor the locations of moving objects so that the "correctness" of the results returned to the requesting clients can be maximized and at the same time minimizing the update cost. We emphasis on providing timely and correct results to LDCQs, instead of simply minimizing the message cost or bounding the location uncertainty.

In Section 2, we review some of the important research findings on generation of updates for tracking the locations of moving objects and the methods for transmitting results to the originating clients. In Section 3, we define our system model. In Section 4, we specify the correctness requirements of the system. We introduce our update generation method in Section 5. Section 6 is devoted to the performance evaluation of the proposed method. Finally, the conclusions of our work are provided in Section 7.

2 Related Work

A mobile network is characterized by frequent disconnection, unreliable, low communication bandwidth and fast changing locations of mobile clients. All such characteristics make traditional techniques used for distributed computing inadequate and raise new challenging research problems, such as cached data management, data dissemination to mobile clients, and location management.

In [SWCD97, SWCD98], the *Moving Object Spatio-Temporal (MOST)* data model is proposed for managing the locations of moving objects and for the prediction of their future locations. According to the MOST model, 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. 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 of 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 resulting tuples for a query are ordered by the *begin times* and sent to the requesting mobile client, before their *begin time*.

Although the MOST data model can be used to predict the future locations of a moving object, updates are still required to track the actual current locations of the objects and to re-define the functions for location predictions. Thus, an important issue is when to generate updates for keeping track of the current locations of moving objects. In [WCDJ97], the *plain dead-reckoning (pdr)* method is proposed in which an update is generated to refresh the location of an object and re-define its location function whenever the deviation of its current location is greater than the last update by a pre-defined threshold. In [SWCD97, SWCD98], the *adaptive dead-reckoning* (*adr*) is proposed by extending *pdr*. 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. In [WSCY99], the methods are further extended to *disconnected detecting dead-reckoning* (*dtdr*) to deal with the problem of network disconnection. *dtdr* avoids the regular process of checking for disconnection by trying to communicate with the moving objects. Instead, in *dtdr*, the threshold continuously decreases as the time interval since the last location update increases.

In addition to update generation, another important issue is transmission of results from a query evaluation to the requesting mobile client. In [GU00], various methods, i.e., immediate transmission, delayed transmission, periodic transmission, and adaptive transmission, are proposed and studied for data transmission to mobile clients. In the immediate transmission, the tuples are sent to the requesting client once they are determined. In the delayed transmission, each tuple is sent to the client just before its *begin time*. In the periodic transmission, the transmission of the tuples is periodic such that all the tuples which have *begin time* within that period are sent. The adaptive periodic transmission is an optimization of the periodic transmission such that the communication cost can be minimized. Besides the communication overhead, the methods are also evaluated in terms of the availability of tuples in the result set of a LDCQ in case of disconnection of the requesting mobile client.

3 System Model

Figure 1 depicts the system architecture of a mobile computing system that supports moving objects and location dependent continuous queries (LDCQs). The system consists of a database server and a number of moving objects. The database server communicates with the moving objects using a low bandwidth network. The server maintains a database, which contains data items for recording the locations of the moving objects. The data items for the moving objects are defined based on the MOST data model described in the preceding section.

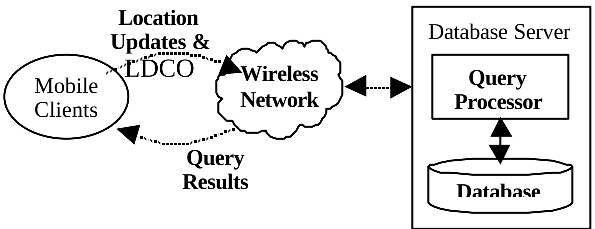

Figure 1: System architecture of a mobile computing system.

Moving objects generate updates to report their current locations to the database server. Each update is associated with a time-stamp, which specifies the time when the current value will become valid. Some moving objects may generate location-dependent continuous queries with a start time and an end time such as LDCQ(*start_time*, *end_time*). The results, in the form of a set of tuples, are collected and grouped by their *begin times*, each indicating the beginning of the time period for which the *object* satisfies the condition of the query. Once the results are ready, the server may send the selected tuples to the mobile client according to a query result transmission approach adopted by the system.

4 Correctness Requirements

The previous work on the design of methods for generating location updates is aimed at bounding the location uncertainty and minimizing the update overhead. However, the issue on ensuring the correctness of the results returned to the requesting clients has been ignored. Note that the main purpose of submitting a query as a LDCQ is to closely monitor the status of the moving objects in the system so that once they have satisfied the conditions of the query, the requesting client will be informed immediately.

In this paper, our main objective is to ensure the correctness of the query results returned to the mobile clients since they may make incorrect actions based on the received incorrect results. Of course, 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 are not selected by any query or for those objects selected but have *begin times* much greater than the current time, the uncertainty in their locations will not affect the correctness of query results significantly.

Intuitively, a query result is incorrect if it provides false information, either in *begin/end times* or in value, to a client. Here, we define the correctness of a result based on its *begin time* by assuming that correct results will always be generated from the database server if the server is provided the most current locations of the moving objects. Firstly, we define the *actual begin time* of an object as the time when a moving object starts to satisfy the conditions of a query. Note that the actual begin time of an object may be different from the *begin time* of the corresponding tuple for the same query since the database may contain out-dated information about the current location of a moving object. A mobile client observes incorrect result if:

(1) the actual begin time of an object equals to the current time and the corresponding tuple's *begin time* is greater than the actual begin time (we call this problem *missed information*); or
(2) the tuple's *begin time*, which has been sent to a client, is smaller than the corresponding object's actual begin time and the tuple's *begin time* is equal to the current time (we call this problem *false information*).

For the first case, a mobile client is not able to identify the moving object which has satisfied the condition of its query before it becomes true. For the second case, a mobile client is informed that an object has met the condition of its query but actually it does not. It is obvious that the major causes of these problems are the out-dated database state and a late transmission (or re-transmission) of the selected tuples to the requesting client. To overcome these problems, location updates have to be continuously reported by moving objects to the database server once the locations are significantly different from their last reported values. Any new results or changes in results have to be sent immediately by the server to the clients which have generated the queries.

Due to the delay in data transmission and processing, it is impossible to have an "instantaneous" location of a moving object. 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. We call this bound a *similarity bound* which is a system or user specific parameter to specify the accuracy of query result. In this paper, we assume that each query is associated with a similarity bound. If the difference between the tuple's *begin time* and the actual begin time is smaller than the similarity bound, the result will be considered to be correct.

5 Update Generation Method

5.1 Overview

The main problem of the plain dead-reckoning (*pdr)* method [SWCD98] is the difficulty in defining the right update threshold value for each moving object. If the values of update thresholds are small, the total update workload will be very high. On the other hand, if large values are used, the uncertainty will be high and correct results cannot be timely reported to the requesting clients. In order to minimize the probability of providing incorrect query results to mobile clients, the update threshold of a moving object may be set close to the similarity bound value of the queries. However, this may result in a heavy update workload since the similarity bound is usually quite small. Although the adaptive dead-reckoning (*adr)* method [SWCD98] adaptively changes the update threshold, the optimization is aimed to minimize the information cost and update cost. It ignores the cost for providing incorrect query results to mobile clients. If a mobile 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 moving object. Therefore, it is not necessary to bound the uncertainty for all the objects.

In order to overcome the problems of *pdr* and *adr*, in the following sub-sections we propose a new method, called *adaptive monitoring method (AMM),* for generation of updates. The main objective of AMM is to closely monitor the status of the moving object so as to provide timely and correct result of a LDCQ to the requesting clients and at the same to minimize the total update workload in the system.

5.2 Adaptive Monitor Method (AMM)

AMM consists of three parts: (1) division of the set of moving objects into selected and unselected sets; (2) determination of the update generation threshold for mobile clients which have submitted LDCQs; (3) determination of the update generation threshold for moving objects.

Similar to the dead-reckoning approaches, the generation of updates for moving objects in AMM also depends on the deviations of the locations of moving objects. An update threshold is defined for each moving object. If the deviation is greater than the update threshold, a location update is generated. However, instead of defining a simple fixed update threshold, we define *threshold bounds,* called 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 *upper threshold bound* can be a very loose (large) value and it is for 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 for 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.

5.2.1 Selected and Unselected Objects

In AMM, the set of moving objects in the system is divided into two sub-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 its predicted path and the predicted path of the requesting client. Both are determined by using the *A.functions* (See Section 2).

Otherwise, an object is grouped into the *unselected object set* of the query. For each selected object, a tuple is generated and placed in the answer set of the query. If we are not certain about the location of a moving object, we include it in the selected set if it satisfies the condition of the query by including the uncertainty into its location. For example, if the query is:

`SELECT moving objects within 5km of location x.`

In evaluating the query, the system will convert it to:

`SELECT moving objects within (5km + uncertainty) of location x.`

The uncertainty can be determined based on the current update thresholds defined for the moving object and the requesting client. Note that the tuples in the selected set will be re-evaluated after each update and a new uncertainty value will be used in query evaluation if the objects' update thresholds have changed.

Although including the uncertainties in the locations of the moving objects will increase the size of the selected object set, the probability that an unselected object will satisfy the query in a short period of time will be low. Therefore, 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.

5.2.2 Update Generation Process of Mobile Clients

As we can see that an important factor that can affect the correctness of query results is the mobility of the mobile clients which have submitted LDCQs. Whenever there is a change, that over the prediction, in the location of such a moving object, all the selected tuples of its query have to be updated. Therefore, it is necessary to pay special attention to defining the update generation process of the requesting clients. We call a mobile client *active* if it has submitted a LDCQ and the query is still being processed. When a location update of a requesting client arrives at the central 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*. For this case, all the *begin times* 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 have to generate updates more frequently, i.e., using the lower update threshold bound, so that their locations can be closely monitored and any change in the evaluation results can be identified earlier.

5.2.3 Update Generation Process of Selected Objects

Once we have determined the objects in the selected object set for a query, we need to determine their update thresholds. Each member in the selected object set is a tuple following the format, <*object, begin time, end time*>. In AMM, an adaptive update generation approach is used in which the update threshold of a moving object is set based on the *begin time* of its corresponding tuple for a LDCQ. The principle is that if the *begin time* of the object is close to the current time, the update threshold is set to be smaller. The aim is to monitor the movement of the objects closely if they will soon satisfy the conditions of the LDCQ. For the moving objects whose *begin time*s are far in the future or are even not currently satisfying the condition of the LDCQ, the update thresholds are set to be close to the upper threshold bound for reducing the update workload.

Assuming an exponential distribution, the update threshold of an object is determined based on the *begin time* of the object, by using the following formula:

Update threshold of object $x_i = H - (H - L) \times e^{-\delta t}$
H: Upper update threshold bound
L: Lower update threshold bound
δt = *begin time* of object x_i – current time

When a moving object is in the answer set of more than one LDCQ, the update threshold of it is set to be the minimum of all the thresholds calculated from all the related LDCQs. If an object's *begin time* is already smaller than the current time in the first evaluation, its update threshold is set to be the lower update threshold bound.

As an example, suppose that a client x_l has submitted an LDCQ, and in the first evaluation of the query at current time 12, the following tuples have been generated as the result of the query:

<x_i, 10, 20>, <x_j, 15, 18>, <x_k, 25, 30>

The update thresholds of objects, x_i, x_j and x_k are determined as shown in Figure 2. The computed update thresholds are sent to the corresponding objects and the generation of updates follows these threshold values. Whenever the central database server receives an update from the moving object or client which has generated the LDCQ, the query is evaluated again and the new *begin times* and *end times* of the set of selected tuples are re-defined. Then, the new update thresholds are calculated using the above formula and sent to the corresponding moving objects.

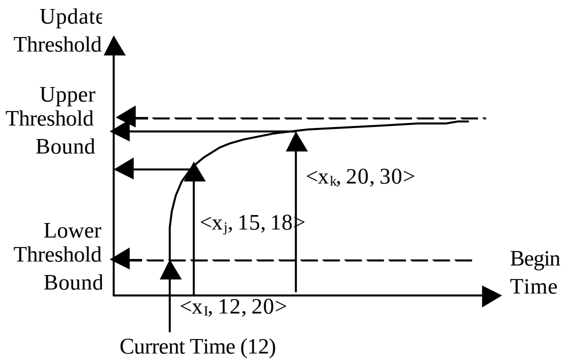

Figure 2: Generation of update thresholds in AMM

Note that in the update threshold formula given above for AMM, we assume an exponential distribution for illustration.

Different systems can adapt different functions for generating the threshold and the guideline would be the consideration of the cost of missing the information as well as the update cost. Similarly, the specification of the upper and lower threshold bounds can also be based on these two cost factors.

5.2.4 Disconnection Operation

A typical characteristic of a mobile network is frequent disconnection of the devices connected to it. A disconnection may be voluntary or involuntary. A voluntary disconnection aims to save the limited computer and network resources. Involuntary disconnection is mainly due to poor network services. Usually, the disconnection can be re-established by retransmission, or after the mobile client moves to a new location. 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 updates from disconnected moving objects to the central database server would not be possible. Thus, 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 threshold bound. A query result involving the moving object may not be reliable and the error in the result may be greater than the uncertainty bound.

In AMM, communication between the central database server and the moving objects already occurs for update transmission and update threshold transmission. Therefore, these messages can also be used for check disconnection. If we assume that a moving object is travelling 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 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, i^{th}) 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 i^{th} 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.

6 Performance Experiments

We have designed and implemented a detailed simulation model to study the performance of the proposed method, AMM, as compared to *pdr* when different query result transmission strategies are employed. Our simulation model is based on the performance models proposed in the previous related work such as [GU00]. These models have been extended to support modeling of processing LDCQs.

As shown in Figure 3, the simulation model consists of three basic components:

- Mobile client model
- Communication Network
- Server model

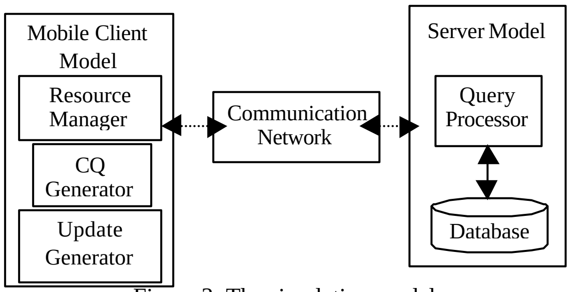

Figure 3: The simulation model

6.1 Mobile Client Model

Each mobile client in our model consists of three components: a resource manager, a continuous query generator and an update generator. The resource manager models the CPU at the client machine for processing queries and presenting query results to the mobile user. It is assumed that there are TotalMO mobile objects among which NumMH 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 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 system, each moving object is assigned a default speed, S. Every time unit, the distance traveled by a moving object is calculated by $S \pm SF$ 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 will generate 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 tuples corresponding 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. To simplify the parameter set, we further assume that the change in the begin time equals to the difference between actual distance traveled and the expected distance traveled.

6.2 Communication Network

All the messages passed between mobile clients and the server must go through the communication network. Each tuple needs TupleTime seconds to go through the network and each control message needs Control-MessageTime seconds. For each transfer, one control message is needed before the tuples are sent. For example, if a mobile client transfers five tuples to the server as a whole, one control message is needed. However, if it transfers the tuples one by one, five control messages are needed for the transfer.

6.3 Server Model

The server model consists of a query processor and a database. The query processor processes LDCQs and updates received through the communication network. It also controls the access to CPU and the database. No I/O time is modeled in the server as we assume that it contains a fast accessed secondary memory.

Once the query processor receives a LDCQ, it will determine the size of the query in terms of the number of tuples in the answer set. The maximum size of a query is specified by the parameter CQSize. The result of a LDCQ is a set of tuples <*S*, *begin time*, *end time*>. The tuples are sent to the requesting mobile client in increasing order of their *begin times*. Both delayed and periodic transmission [GU00] of tuples to the clients are simulated. With the delayed transmission approach, a tuple 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 satisfying the condition $t \leq$ *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.

When a location update is received, the server first determines whether the update is from an active client. If it is, a minimum update threshold is set for the active client and all the *begin times* of the associated objects for the client's query are adjusted. For a non-active client, the *begin times* of the related tuples are also adjusted. After the adjustment, the new update threshold is recalculated for the tuples and sent to the corresponding mobile clients.

6.4 Model Parameters and Measures

The following table summarizes the model parameters and the baseline settings. The baseline parameter settings are determined based on the values used in [GU00]. To study the 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 4) over the total number tuples generated. IIR indicates the capability of the system in providing correct information to the queries from the mobile clients. In addition to IIR, we also measure the retransmission rate (RR), control message overhead (CMO) and update workload. As explained in Section 2, when a window based method is used for transmitting the results of a tuple to its requesting client, a tuple may need to be retransmitted due to the changes in the result. RR is defined as the total number of tuple re-transmissions over the total number of tuples transmitted. CMO measures the total number of control messages per unit time. It measures the communication overhead between mobile clients and the server. It includes both the control messages from mobile clients to the server and server to mobile clients. It specifies the network loading. Update workload measures the proportion of CPU utilization for processing of location updates of the moving objects in the system.

Mobile Client Parameters	**Baseline Value**
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 sec
Maximum life of a LDCQ (MaxCQLife)	360 sec
Think time (ThinkTime)	1000 sec
Speed Bound (SB)	0–1 (for each time unit)
Communication Network Parameters	**Baseline Value**
Time for sending a tuple (TupleTime)	0.1 – 0.2 sec (normal distribution)
Time for sending a control message (ControlMessageTime)	0.05 – 0.1sec (normal distribution)
Server Parameters	**Baseline Value**
Time for processing a tuple (ComputeTime)	0.05 – 0.1sec (normal distribution)
Threshold limits	5 – 30 sec
Window size for transmitting results	50 sec
Similarity bound	10 sec

6.5 Performance Results

The simulation program was implemented in CSIM-18 which is a simulation language based on the C programming language. The length of each simulation run is 10 hours. This value has been determined from a number of trial runs until stable results have been obtained. We presented three sets of experimental results to illustrate the performance of AMM compared with pdr. In the first two sets of experiments, the upper threshold bound of AMM is fixed at 30 seconds and the lower threshold bound is varied from 2.5 to 30 seconds. The first set of experiments compares the performance of AMM with pdr under the delayed transmission method for transmitting LDCQ results, while the second set of experiments compares the performance of the two methods using the periodic transmission method with a window size of 50s. In the last set of experiments, the performance of AMM under different upper threshold bounds is investigated.

Figure 4 depicts the incorrect information rate (IIR) when different values are used for the upper 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 that the performance of AMM is consistently much better than pdr, As expected, both curves are in V-shape. The best performance is achieved when the threshold value is around 12.5s. The poor performance with small threshold values is due to heavy update workload as can be observed in Figure 5 in which the update workload is close to 70% for AMM and 95% for pdr. Thus, under such settings, most of the system resources are devoted to process the updates from 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 of the location of a moving object will be high. Thus, the probability of providing incorrect information to mobile clients will also be high.

The better performance of AMM is due to the better monitoring scheme used in generating location updates, i.e., a smaller threshold is assigned to the moving object if its *begin time* is closer to the current time. Although the total number of updates generated is smaller under AMM than under pdr, the locations of mobile clients can be closely monitored. 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 updates. The smaller control message overhead in AMM can be observed in Figure 6.

In the second set of experiments, periodic transmission of query results is used with a window size of 50 seconds. The results are shown in Figures 7 to 10. Consistent to the results in the previous figures, the performance of AMM is much better than that of pdr. Comparing the results in Figure 7 with Figure 4, we can see that the performance of both methods is improved when the periodic transmission method is used. Although under the periodic transmission re-transmissions 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 Figure 10 with Figure 6. As shown in Figure 9, the better performance of AMM is also due to a smaller retransmission rate.

In the last set of experiments, we vary the upper threshold bound of AMM. The results are shown in Figures 11 to 13. 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 be 20, 30, 40, 50 and 75, respectively. As shown in Figure 11, the best performance is achieved when a medium upper threshold bound is chosen. If a smaller upper threshold bound is used, the update workload will be heavy as shown in Figure 12. When a large upper threshold bound is used, the uncertainty in the locations of the moving object will be larger although the update overhead and the control message overhead will be lower (as shown in Figure 12 and Figure 13). Increasing the upper update threshold results in higher number of updates. The graph displayed in Figure 11 for a large upper update threshold bound is interesting. The smallest value of IIR decreases and then increases when a larger upper threshold bound is used. The thresholds of the objects become tight when a smaller upper bound is used and IIR becomes greater, even larger than that of pdr. It is obvious that when the threshold is increased, the update frequency decreases. This explains the increase in IIR with the increase in the upper threshold bounds. However, the system performance with a larger upper threshold bound is better at a small lower threshold bound. With a small lower threshold bound, the thresholds for mobile objects are small enough to keep an effective update frequency. This explains the increase in system performance with small lower threshold bounds. These experiments also verify the results obtained with the setting of 30 to the upper threshold bound in the first three sets of experiments.

7 Conclusions

An important issue that should be considered in designing a mobile computing system is to provide support for processing of location-dependent continuous queries. Data values used for maintaining the locations of moving objects are highly dynamic and may possess real-time properties. Location-dependent queries from mobile clients may also be associated with timing constraints on their response times. In this paper, we have analyzed the issues related to the generation of location updates from mobile clients and the transmission of results of location dependent continuous queries from the server. A new method, called Adaptive Monitor Method (AMM) has been proposed with the aim to reduce the update workload and at the same time closely monitor the locations of moving objects. The objects, which are going to be included in query result soon, are monitored more closely by the proposed method. Extensive simulation experiments have been performed to investigate the effectiveness of the suggested method. The method makes use of two threshold bounds to calculate an update threshold, based on the begin time of query results, in generating updates for the location of moving objects.

Acknowledgement: The work described in this paper was partially supported by a grant from the Research Grants Council of Hong Knog SAR, China [Project No. 9040512] and a grant from CityU (Project No. 7001069).

References

[AFZ97] S. Acharya, M. Franklin, S. Zdonik, "Balancing Push and Pull for Data Broadcast", *Proceedings of ACM SIGMOD*, Tucson, Arizona, 1997.

[DK98] M.H. Dunham and V, Kumar, "Location dependent data and its management in mobile databases," in *Proceedings of International Workshop of Database and Expert Systems Applications*, pp. 414-419, 1998.

[GU00] Hüseyin Gökmen Gök and Özgür Ulusoy, "Transmission of Continuous Query Results in Mobile Computing Systems", *Information Sciences*, vol. 125, no. 1-4, pages 37-63, 2000.

[SWCD97] A.P.Sistla, O.Wolfson, S.Chamberlain, S. Dao, "Modeling and querying moving objects", Proceedings of the 13th International Conference on Data Engineering, pages 422-432, Birmingham, UK, April 1997.

[SWCD98] A.P.Sistla, O.Wolfson, S.Chamberlain, S. Dao, "Querying the Uncertain Position of Moving Objects", Temporal Database: Research and Practice, pages 310-337, Lecture Notes in Computer Science (Springer Verlag), 1998.

[WCDJ97] O. Wolfson, Sam Chamberlain, Son Dao, Liqin Jiang " Loation Management in Moving Objects Databases", WOSBIS'97, pages 7-13, Budapes, Hungary, October 1997.

[WSCY99] O. Wolfson, P. Sistla, S. Chamberlain, Y. Yesha, "Updating and Querying Databases that Track Mobile Units", *Distributed and Parallel Databases,* vol. 7, no. 3, pp. 257-287, 1999.

[WXCJ98] O. Wolfson, Bo Xu, Sam Chamberlain, Liqin Jiang, "Moving Objects Databases: Issues and Solutions", 10th International Conference on Scientific & Statistical Database, pages 111-122, Italy, July 1998.

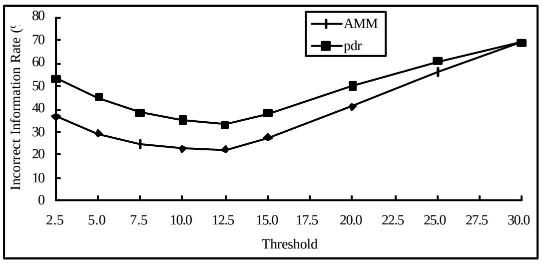

Figure 4. The impact of Lower Threshold Limit on Incorrect Information Rate under Delayed Transmit

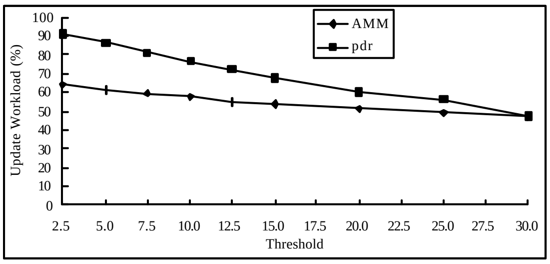

Figure 5. The impact of Lower Threshold Limit on Update Workload under Delayed Transmit

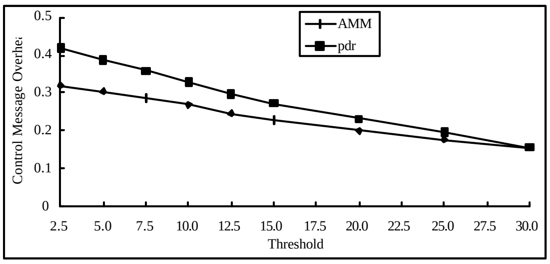

Figure 6. The impact of Lower Threshold Limit on Control Message Overhead under Delayed Transmit

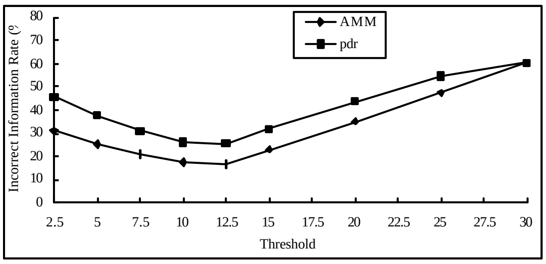

Figure 7. The impact of Lower Threshold Limit on Incorrect Information Rate under Periodic Transmit with window size of 50s

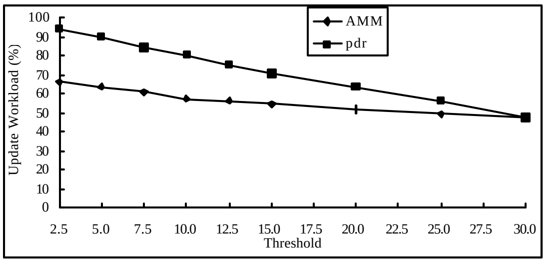

Figure 8. The impact of Lower Threshold Limit on Update Workload under Periodic Transmit with window size of 50s

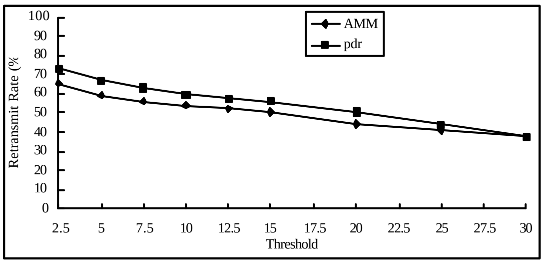

Figure 9. The impact of Lower Threshold Limit on Retransmit Rate of LDCQ results under Periodic Transmit with window size of 50s

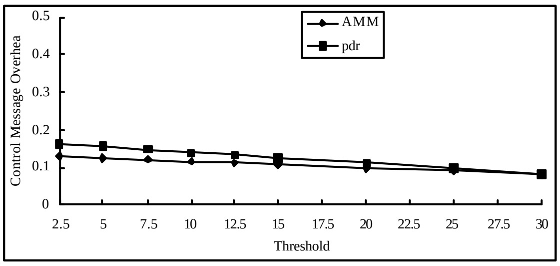

Figure 10. The impact of Lower Threshold Limit on Control Message Overhead under Periodic Transmit with window size of 50s

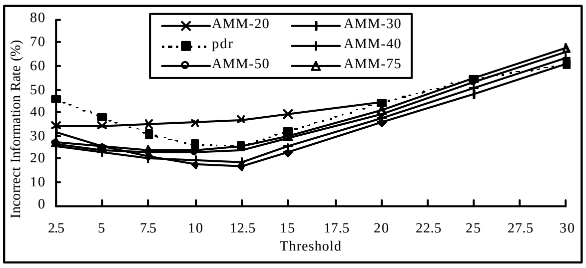

Figure 11. The impact of Upper Threshold Limit on Incorrect Information Rate under Periodic Transmit with window size of 50s

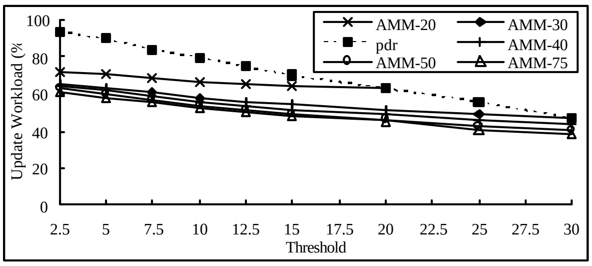

Figure 12. The impact of Upper Threshold Limit on Update Workload under Periodic Transmit with window size of 50s

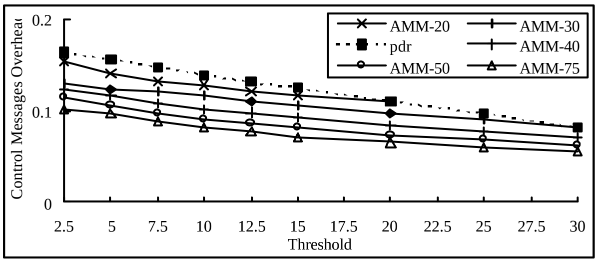

Figure 13. The impact of Upper Threshold Limit on Control Message Overhead under Periodic Transmit with window size of 50s