AN EFFICIENT METHOD FOR GENERATING UPDATES IN PROCESSING LOCATION DEPENDENT CONTINUOUS QUERIES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

by

İlker Yoncacı

September, 2001

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Özgür Ulusoy (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. H. Altay Güvenir

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Uğur Güdükbay

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Science

ABSTRACT

An Efficient Method for Generating Location Updates in Processing Location Dependent Continuous Queries

İlker Yoncacı
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Özgür Ulusoy
September, 2001

Recent advances in wireless communication networks and portable computers have greatly increased the functionality of mobile information services and mobile computing applications. Most of those applications are expected to support location-dependent continuous queries (LDCQs) in the near future. A LDCQ is evaluated continuously in the system for a period of time and the changing results can be transmitted to the requesting client. The result of a LDCQ depends on both the locations of mobile objects on which the query has been issued and the location of the mobile client which has submitted the query. The result of a LDCQ can be provided to the requesting client as a set of tuples $< S, begin, end >$ indicating that object S satisfies the query from time *begin* to time *end*. The correctness and timeliness of query results of LDCQs have been studied by employing an adaptive monitoring method (AMM) that manages the locations of moving objects. In this thesis, we propose a Categorized Adaptive Monitoring Method (CAMM) which allows to specify various criticality levels for LDCQs. The aim of CAMM is to provide higher levels of correctness for query results as the criticality of the query increases, without significantly increasing the wireless bandwidth requirements. We provide a simulation model with multiple criticality levels and evaluate the performance of the mobile system and the proposed method with extensive simulation experiments.

Keywords: Mobile computing, location management, location update generation, location dependent continuous queries.

ÖZET

KONUMA DAYALI SÜREKLİ SORGULARIN İŞLENMESİNDE KULLANILAN ETKİN BİR KONUM GÜNCELLEME METODU

İlker Yoncacı
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Özgür Ulusoy
Eylül, 2001

Mobil haberleşme ağları ve taşınabilir bilgisayarlardaki son teknolojik gelişmeler mobil bilgi servislerinin ve mobil uygulamaların fonksiyonlarını önemli derecede arttırmıştır. Bu uygulamaların pek çoğunun yakın gelecekte konuma dayalı sürekli sorguları desteklemesi beklenmektedir. Bir konuma dayalı sürekli sorgu belli bir zaman dilimi içerisinde arka arkaya işlenir ve değişen sonuçlar sorguyu yapan mobil kullanıcıya iletilir. Mobil bir kullanıcı tarafından sisteme verilen konuma dayalı sorguların sonucu hem sorgulanan mobil nesnelerin konumuna hem de sorguyu yapan mobil kullanıcının konumuna bağlıdır. Konuma dayalı sürekli bir sorguda S nesnesi *başlangıç* zamanından *bitiş* zamanına kadar bu sorgunun yanıtına dahil ise, bunun gösterimi $< S,$ *başlangıç, bitiş* $>$ şeklinde olabilir. Sürekli sorguların doğruluk ve zamanlamaları hareketli nesnelerin konumunu yönlendiren Değişken İzleme Metodu (DİM) ile daha önce incelenmiştir. Bu tezde, mobil bilgi sistemlerinde sorgulara değişik kategoriler verilebilmesine olanak sağlayan Kategorize Edilmiş İzleme Metodu (KEİM) önerilmiştir. KEİM metodunun amacı, mobil sistemlerde kritiklik kategorisi arttıkça, sistemin yükünü çok arttırmadan sorgu sonuçlarının doğruluğunu arttırmaktır. Bu tezde çok sayıdaki kritiklik seviyesi olan bir simülasyon modeli kullanılmış ve bu modelle önerilen metodun geniş bir yelpazede deneyleri yapılmıştır.

Anahtar sözcükler: Mobil sistemler, konum yönetimi, konum güncelleme, konuma dayalı sürekli sorgu.

Türk Silahlı Kuvvetleri'ne

ve

Aileme.

ACKNOWLEDGMENTS

First of all, I would like to express my deepest gratitude to my advisor Assoc. Prof. Dr. Özgür Ulusoy for his great supervision, guidance and patience for the development of this thesis.

I would like to thank to committee members Prof.Dr. H.Altay Güvenir and Assist. Prof. Dr. Uğur Güdükbay for reading the thesis and for their constructive comments.

I also would like to thank to my wife Vahide and my son Alperen for their great patience, moral support and encouragement.

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

LDQ	: Location Dependent Query
LDCQ	: Location Dependent Continuous Query
AMM	: Adaptive Monitoring Method
CAMM	: Criticality-Based Adaptive Monitoring Method
DBMS	: Database Management System
MOST	: Moving Objects Spatio-Temporal Model
S	: Object which satisfies the LDCQ
begin	: Time in which the object S begins to satisfy the query
end	: Time in which the object S begins not to satisfy the query
AnswerCQ	: The set of all tuples in the answer set of the query
pdr	: Plain Dead-Reckoning Method
adr	: Adaptive Dead-Reckoning Method
dtdr	: Disconnection Detecting Dead-Reckoning Method
IT	: Immediate Transmission Method
DT	: Delayed Transmission Method
PT	: Periodic Transmission Method
APT	: Adaptive Periodic Transmission Method
MT	: Mixed Transmission Method
Selected Object Set	: The set of moving objects which AMM monitors closely
Unselected Object Set	: The set of moving objects which AMM neglects due to their far begin times
H	: Upper update threshold bound for generating location updates
L	: Lower update threshold bound for generating location updates
δ	: Time interval between the *begintime* of object x and the current time
C_{up}	: Criticality Category Upper Bound
C_{lw}	: Criticality Category Lower Bound

C_c	: Criticality Category of the query generated
C_l	: Criticality Level of the mobile client generating the query
IIR	: Incorrect Information Rate
RR	: Retransmission Rate

Chapter 1

Introduction

Recent advances in wireless communication networks and portable computers have greatly increased the functionality of mobile information services and mobile computing applications. The rapid development in mobile computing technology in recent years has made many expectations about wide usage of mobile computing systems a reality. A number of novel mobile information services, such as mobile shopping aids in a large shopping mall and stock exchange information distribution to users via mobile computers or mobile phones have already been implemented [LAC99]. As the users require instant and accurate access to information using their portable devices, different applications such as personal locator services and real-time stock information services will no doubt continue to emerge.

Although a typical mobile computing environment was described in [IB94, U98] with two distinct entities in Figure 1.1: *mobile computers* and *fixed (non-mobile) computers*, the unbelievable speed of recent advances in wireless communication technology has changed this definition in recent papers [LAC99, GU00, LCY00, LULCL01] as "a typical mobile computing system can be characterized by an information server and a number of mobile hosts connected to the server through a mobile network such as a cellular radio system" (Figure 1.2). Mobile computers which could be either information server or clients, can move while retaining their network connection through wireless links. Mobile hosts represent users equipped with mobile units that can communicate

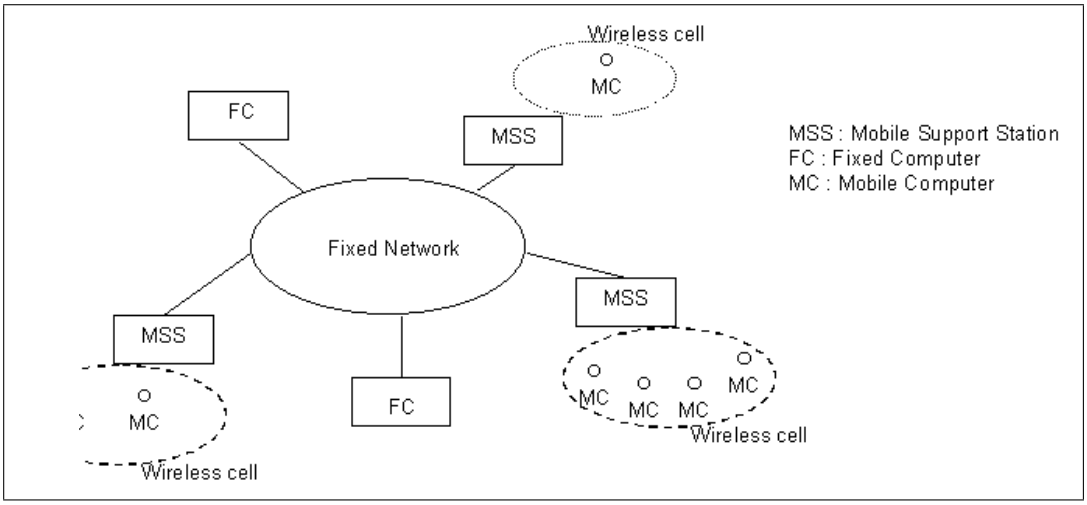

Figure 1.1: A traditional mobile computing system.

and generate queries on low bandwidth. The database server communicates with the moving mobile clients using a low bandwidth network such as the GSM network [MP92]. Figure 1.2 shows a system architecture of a mobile computing system that supports different types of queries including the so-called location-dependent queries (LDQs) on moving objects. The query submitted by a missile operator "retrieve the enemy tanks within 5 miles of my current position" in a battlefield might be an example of LDQs and the result of the query depends on both the locations of enemy tanks and the current location of the operator at time *t*. Besides, the operator may want to see the results continuously for a while. In this case, a LDQ becomes much more complex to process and this new type of LDQ might be called Location Dependent Continuous Query (LDCQ). A LDCQ is evaluated continuously in the system for a period of time and the changing results can be transmitted to the requesting client by employing various transmission techniques [GU00].

Although there exist several studies in the literature which explore LDCQs, none of them discusses the processing of LDCQs with different priorities or criticalness. In our work, we have considered categorization of LDCQs, according to their user defined criticalness. We believe that some of the mobile users in a mobile computing environment may need the answer of a LDCQ to be more accurate than those of other users, or the same user may want to categorize his/her LDCQs based on their required levels of accuracy. We have implemented a detailed simulation model of a mobile computing system that

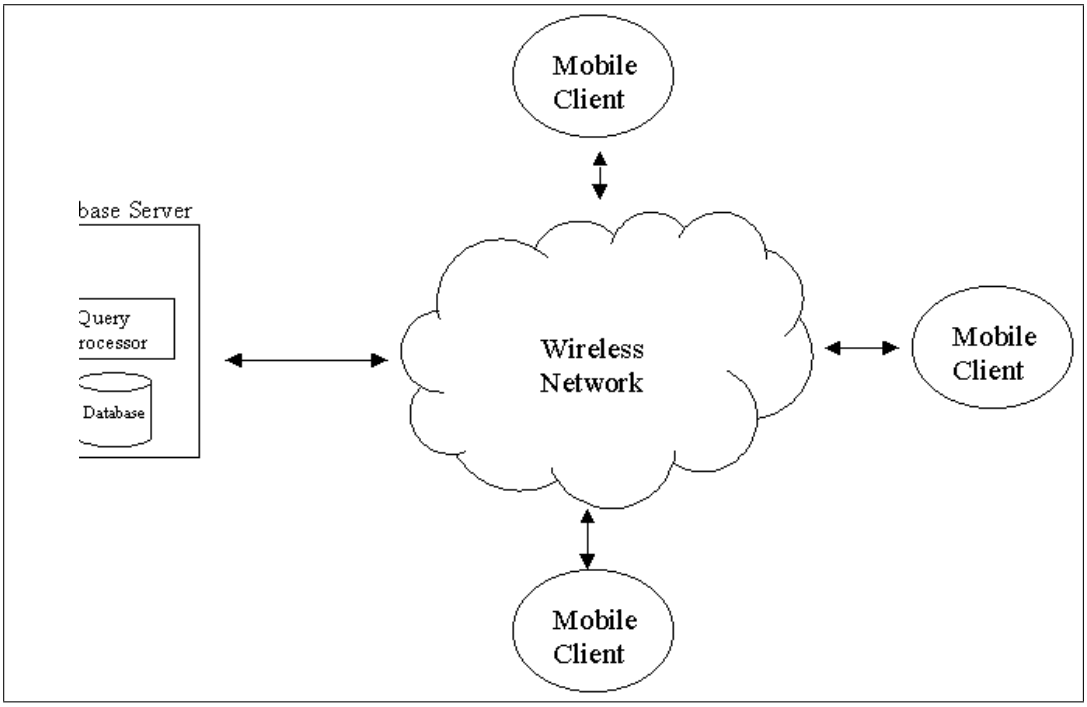

Figure 1.2: A wireless mobile computing system.

supports processing of LDCQs associated with different criticality categories. We have designed a new method to monitor the locations of moving objects based on the criticality category of LDCQs so that higher levels of accuracy can be achieved for the results returned to the requesting client for the LDCQs categorized with higher criticality.

The remainder of this thesis is organized as follows. In Chapter 2, we provide a background and review the related work. In Chapter 3, we describe the location update generation method Adaptive Monitoring Method (AMM) for LDCQs. We introduce our criticality-based location update generation method (CAMM) in Chapter 4. Chapter 5 provides the system model and performance evaluation results of the proposed method. Finally, concluding remarks are provided in Chapter 6.

Chapter 2

Background and Related Work

In this chapter, we provide an overview of the basic ideas behind mobile computing, data dissemination and location dependent query processing.

2.1 Mobile Computing Systems

A mobile computing system is a dynamic type of traditional distributed systems where the links between mobile computers and the mobile support stations that provide the wireless interface to the mobile computers can change dynamically [KU99]. The mobile computing system that we assume to support location dependent continuous queries (LDCQs), consists of a database server and a number of mobile hosts. The database server communicates with the moving hosts using a low bandwidth wireless network. Mobile hosts (or mobile clients) represent users equipped with mobile units.

The new challenges in mobile computing and the impact of mobile computing on data management are discussed in [IB94, B99]. In [B99], asymmetry in the communications, frequent disconnections, power limitations and small screen size are addressed as the characteristic features that make the mobile systems unique. On the other hand, prototyping, bandwidth utilization, transactional properties and optimization of location dependent query processing are pointed as the challenges still remaining to be investigated.

In many mobile computing systems, data items maintained at an information server store status information such as the last traded stock prices, current traffic situation of a road, news updates and weather conditions. Because of the natural dynamic features of the stored data items, it may be necessary to refresh the database records frequently. Otherwise, the values of data items may be outdated and the results of continuous queries may not be accurate. In a mobile computing environment, the need for a real-time database management (DBMS) is strong, because one of the basic requirements in a mobile data management is to provide real-time response to the transactions of the underlying application. Recent advances in mobile computing systems enforced the support of mobile computing systems with real time DBMSs. One of the aspects of this support, timing requirements of mobile computing systems are discussed in [U98]. Mobility of hosts makes the location information dynamic and this feature causes a considerable increase in the cost of search and updates on the mobile hosts location. The resource constraints of mobile computing systems, such as low bandwidth, unreliable wireless links and frequent disconnections between mobile hosts, make it difficult to satisfy timing requirements of supported applications. In [KU99], a mobile transaction execution model is proposed to obtain a reasonable real-time performance in mobile computing systems.

2.2 Data Dissemination

Mobile computing systems are characterized by a number of intrinsic constraints such as high transmission error rates, low bandwidth, limited power supply and unreliable communication. Due to these basic constraints of mobile computing systems, efficient data dissemination to transactions from mobile clients becomes the most important issue. Although different data dissemination methods have been proposed to meet the different data requirements of the mobile clients, basically, there are two approaches to disseminate data items: on-demand and data broadcast [LAC99]. In the former approach, the data items from the information server are sent on request and in the latter,

the data items from the information server are sent continuously and periodically to the mobile clients. Although the on-demand approach is simple, if there are a large number of transactions waiting for different data items, this approach does not scale well. On the other hand, the data broadcast approach can satisfy multiple requests for the same data item simultaneously. Thus, the data broadcast approach is suitable for disseminating large amount of information to a large amount of mobile clients where bandwidth efficiency is a major concern [LAC99]. In [DCKV97], the authors mention that the most likely mode of retrieval in mobile environment is going to be mixed, i.e., the most frequently accessed items are broadcast and other items are provided on demand. They propose adaptive broadcast protocols that dynamically alter the broadcast content depending upon client demand patterns to support such a mixed technique.

Data dissemination problem is addressed in [LCY00] in terms of completion deadline and validity issues. New broadcast approaches are proposed in which the temporal properties of the data items are considered in selecting the data items for broadcast. These methods are evaluated in terms of the client cache hit probability and access delay for the data items.

2.3 Moving Objects Spatio-Temporal (MOST) Model

A new data model in location management, MOST (Moving Objects Spatio-Temporal) is proposed in [SWCD98, SWCD97, WXCJ98] for databases with continuously changing attributes as a function of time. Consider a database that represents information about the current location of objects in a battlefield. A typical query for that database may be: "retrieve the friendly tanks that are closer to headquarters than 5 miles". Furthermore, the queries may originate from the moving objects, or from stationary users [WXCJ98]. In such databases, an answer to a query depends on both database content and the time at which the query is entered. In order to represent moving objects (e.g., tanks in a battlefield) in a database, the moving object's position has to be

continuously updated. In addition to the database records, the answer sets to queries also have to be updated. It is obvious that position updates of moving objects would impose a serious performance and wireless-bandwidth overhead and the answer sets would be outdated. In order to overcome these problems, in MOST model, some of the attributes of a moving data item are stored as dynamic attributes. For instance, the position of a moving object (e.g., tank) is given as a function of its motion vector (e.g., east, at 20 miles/hour).

In MOST data model, while a static attribute changes only when an explicit update of the database occurs, a dynamic attribute may change over time according to a function given independent from explicit updates. A dynamic attribute, X is represented with three sub-attributes *X.value*, *X.updatetime*, and *X.function*. The value of X at time *X.begintime*+t_o is given by;

$$X.value + X.function \times t_o$$

as a function of time. In other words, even though the database has not been updated, the answer of a query that includes object X may be different for the time interval from *X.begintime* to t_o.

A query is a function that takes as input a database trace and a time value, and outputs a set of values [SWCD98]. In MOST data model, three types of queries are supported: *instantaneous*, *continuous* and *persistent*:

- an instantaneous query at time t uses the set of current database states,
- a continuous query at time t is a sequence of instantaneous queries at each time $t' \geq t$, and
- a persistent query at time t is a sequence of instantaneous queries, all at time t. The queries are evaluated at each time $t' \geq t$ when the database is updated.

The same query may be entered as *instantaneous*, *continuous* and *persistent*, and in each case the query produces different results. An instantaneous query uses the set of current database states at time t, i.e., the time when the query is entered. For example, "display the friendly tanks within 5 miles of

headquarters" is an instantaneous query and it returns a list of friendly tanks, presented to the user immediately after the query is evaluated. Assume that no friendly tank is found as a result of evaluating this query. In that case, the headquarters may wish to make the query continuous, to be evaluated continuosly at each time instance or in other words requesting the system to respond as an instantaneous query being continuously reissued at each clock tick until canceled: "display the friendly tanks within 5 miles of headquarters for 10 minutes". While executing this continuous query, the server computes the result set and returns a set of tuples, each in the form of $< S, begin, end >$ which indicates that object S satisfies the continuous query from the time *begin* to the time *end* (Figure 2.1). *AnswerCQ* denotes the set of all tuples in the answer set of the query.

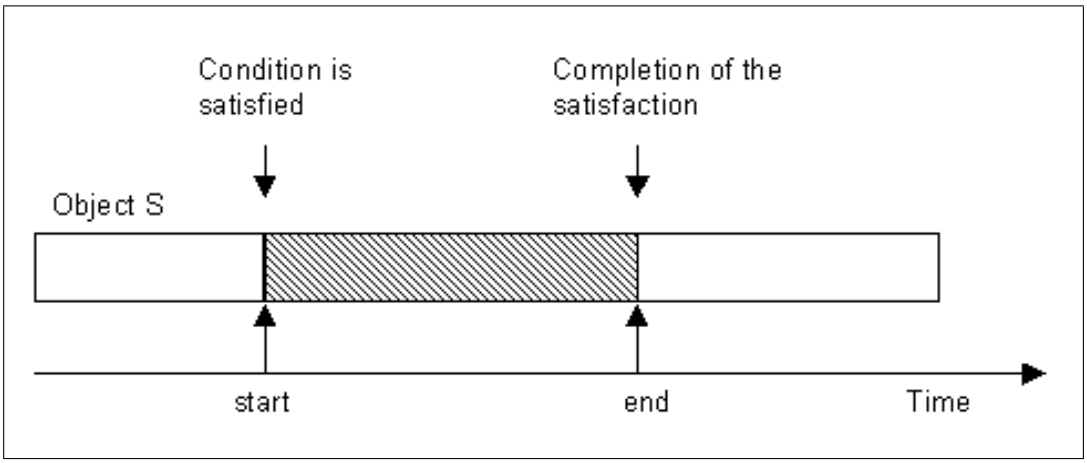

Figure 2.1: Query representation.

As the third type, a persistent query at time t is a sequence of instantaneous queries, at each future time $t' \geq t$, consisting of two arguments *(i)* a view of the database at time t', and *(ii)* the time value t. In fact, a persistent query consists of a series of instantaneous queries having the same starting point and initial values. As the time t' increases, the values of dynamic attributes of each object change with respect to the initial values at time t. For example, the headquarters may wish to find the enemy tanks whose speed doubles in the battlefield within the next 30 minutes : "display the enemy tanks whose speed in the direction of east doubles within 30 minutes". Suppose that the query is entered as persistent at time t. Whenever there is a change that doubles the speed of any enemy tank, this tank is included in the answer set of the query.

2.4 Methods for Generating Location Updates

As we have stated earlier, frequent position-updating is very expensive in terms of search and update costs on the mobile host location and wireless-bandwidth overhead. In order to reduce the search and update costs, the current position of a moving object is modeled by *Wolfson et al.* [SWCD97] as the distance from its starting point, along a given route. So, the DBMS computes the current position of moving object along the given route according to a given speed. On the other hand, the actual position of moving object may deviate from the position computed by the DBMS, due to the fact that moving object does not travel exactly on the given route with a constant speed. To handle this situation, *plain dead-reckoning (pdr)* method is proposed in [WCDJ97] as an update policy. In *plain dead-reckoning (pdr)* method, the moving object has a predefined threshold *th*, and the object updates the current position of itself whenever the deviation (i.e., the difference between the actual position and the database position) exceeds *th*. Defining a fixed threshold to update the position of a moving object arises some problems. As the update cost varies over time depending on the demand for bandwidth, *plain dead-reckoning (pdr)* method becomes unsatisfactory. To address this problem, *adaptive dead-reckoning method (adr)* is proposed in [WCDJ97] which provides at each update a new threshold *th* that is computed using a cost based approach.

A common problem to both *pdr* and *adr* is that the moving object may be disconnected for a while. In other words, although the DBMS thinks that updates are not generated since the deviation does not exceed the update threshold, in fact the actual reason is that the moving object is disconnected. In [WSCY99], a new method called *disconnection detecting dead-reckoning (dtdr)* is introduced to cope with this problem. Instead of regular checking process for disconnection, the threshold continuously decreases; i.e., *th* for the first time unit, *th/2* for the second, and so on. Threshold *th* minimizes the total information cost including update cost, deviation cost and uncertainty cost.

In the next chapter, we provide a detailed description of another method, called Adaptive Monitoring Method (AMM), for generating location updates while processing location dependent continuous queries [LULCL01]. Unlike the

other methods that have been proposed with the aim of optimizing utilization of the limited wireless bandwidth, the aim of *AMM* is to maintain the correctness of the results of query evaluation with an insignificant increase in the wireless bandwidth requirement.

2.5 Query Result Transmission Methods

Consider a DBMS and many mobile clients connected to this DBMS with a wireless network. Whenever a client issues a continuous query *CQ*, the DBMS computes the result set *AnswerCQ* and sends this set to the requesting client. The result set consists of tuples $< S,\ begin,\ end >$ which indicates that object S is the answer of the generated *CQ* from time *begin* to time *end*. There are two basic ways which the DBMS can choose to transmit the result set *AnswerCQ* to the mobile client in order to reduce the communication overhead : *immediate approach* and *delayed approach* [SWCD97]. In the *immediate transmission (IT)* approach, the whole result set is transmitted immediately after being computed. This approach minimizes the control message overhead because all tuples are sent in a single message. In the *delayed transmission (DT)* approach, each tuple is transmitted to the mobile client at time *begin*. This approach minimizes retransmission overhead due to the explicit updates. In addition to these two methods, three new methods are proposed in [GU00]. *Periodic Transmission (PT)* tranmits the tuples in the answer set periodically. At each w time units, this method transmits all the tuples $< S,\ begin,\ end >$ satisfying the condition $t \leq begintime < t + w$ where t is the current time and w is the size of the time window. The second method, *Adaptive Periodic Transmission (APT)*, transmits the tuples in the answer set with a dynamicly adjusted period w. Finally, *Mixed Transmission (MT)* method partitions data objects into two categories as *hot* and *cold* objects. *Mixed Transmission* method transmits the *hot* tuples as in *APT* and *cold* tuples as in *IT*.

Chapter 3

Adaptive Monitoring Method (AMM)

In this chapter, we present the details of a location update generation method which our work is based on. The method proposed for processing LDCQs is called Adaptive Monitoring Method (AMM).

3.1 Aim

In [LULCL01], a new method, called *adaptive monitoring method (AMM)*, is proposed for managing the locations of moving objects. AMM aims to maintain the correctness of the results of query evaluation without significantly increasing the wireless bandwidth requirements by closely monitoring the location status of moving objects.

In AMM, moving objects generate updates to report their current locations to the database server with a time-stamp that specifies the time when the current value will become valid. In our model, some of the mobile objects (that we call mobile clients) may generate location-dependent continuous queries on other objects. Each of these queries is submitted with a *start time* and an *end time* in the form of *LDCQ (start_time, end_time)*. The information server reevaluates each query when there is any change in the database state during

the period of time between *start-time* and *end-time*. After evaluating the query, the information server lists the tuples of the answer set by their *begin* times, which indicate the beginning of the time period for which the mobile object in the tuple satisfies the condition of the query. Once the results are ready, the information server may send the tuples to the requesting mobile client.

Although *adaptive monitoring method (AMM)* achieves better performance relative to *adr* and *pdr* methods (see Section 2.4), it uses the same rule to generate update thresholds. In fact, as we stated earlier, AMM aims 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 [LULCL01]. If a result set of a query provides false information such as incorrect *begin/end times* or incorrect values, we assume this result to be incorrect.

3.2 Similarity Bound

It is almost impossible to have a real-time database that contains accurate location information of mobile clients. However, in practice, it is usually accepted that the recorded location of a moving client is considered to be the "same" as its current location if the deviation is very small [LULCL01], i.e., smaller than a pre-defined bound. This pre-defined bound, denoted by a *similarity bound*, is a system or user-specific parameter to specify the accuracy of query result. Both in AMM and in our model, it is assumed that each query is associated with a *similarity bound*.

3.3 Selected and Unselected Sets

AMM has three main parts: (i) partitioning all moving objects into *selected* and *unselected sets*; (ii) determining the update generation threshold for mobile clients that have submitted LDCQ's; and (iii) determination of the update generation threshold for other mobile objects. While determining the update threshold values for all moving objects, AMM considers the deviations of the

locations of moving objects. If the deviation is greater than the update threshold, a location update is generated. Instead of using a threshold value, AMM uses threshold bounds, called *upper* and *lower threshold bounds* to determine the right update threshold value for each object. The *lower threshold bound* can be a tight (small) value and *upper threshold bound* can be loose (large) value.

In *AMM*, all moving objects are divided into two sub-sets as *selected object set* and *unselected object set* for each query. A moving object is in the *selected object set* of a query if;

(i) it satisfies the condition of the query currently, or

(ii) it will satisfy the condition in the near future according to the predictions done by using the MOST data model function *A.function* (see Section 2.3).

Otherwise, a mobile object is classified into the *unselected object set* of the query. We are especially interested in the *selected object set* for the query and mobile objects in that set should update their location information more often than the mobile objects in the *unselected object set*. So, we need to assign smaller update threshold values to the mobile objects in the *selected object set* to monitor their locations closely. We can assign bigger update threshold values (e.g., the upper threshold bound) for the mobile objects in the *unselected object set*.

Although all of the movements done by all mobile objects affect the location update process, the effect of the mobility of mobile clients that have submitted LDCQs on the location update process and the correctness of query results is much higher. Whenever there is a change in the location of such a moving client, if the change differs from the prediction, all the selected tuples of its query have to be updated [LULCL01]. So, the requesting clients should be monitored very closely to get precise answers to their LDCQs. In order to monitor the status of the requesting clients we should assign smaller update threshold values (lower threshold bound) to these mobile clients so they can generate updates more frequently.

The third part of *AMM* is the generation of right update threshold value of mobile objects excluding the requesting client. Once we divide all moving objects into two sub-sets as *selected object set* and *unselected object set*, we can simply assign upper threshold bound to the objects in the *unselected object set* since we are not interested in these objects for a while. On the other hand, we have to compute a threshold value for each mobile object in the *selected object set*. According to the *MOST* data model, each object in the *selected object set* is a tuple with the format $< S, begin, end >$. 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. According to *AMM*, the moving object with the closer begin time to the current time gets smaller update threshold value. In this way, the moving objects can be monitored closely if they will satisfy the conditions of the LDCQ in the near future. Similarly, the moving objects whose *begin times* are far in the future get higher update threshold values close to the upper threshold bound.

3.4 Determination of Update Threshold

The update threshold of a mobile object is determined based on the begin time of the object, by using the following formula;

$$\text{Update threshold of object} \quad x = H - (H - L) \times e^{-\delta t}, \tag{3.1}$$

where

H : Upper Update Threshold bound,

L : Lower Update Threshold bound, and

δt : *begin time* of object $x - currenttime$.

As an example, suppose that a mobile client MC_1 has submitted a LDCQ, and in the first evaluation of the query at current time 20, the following tuples have been generated:

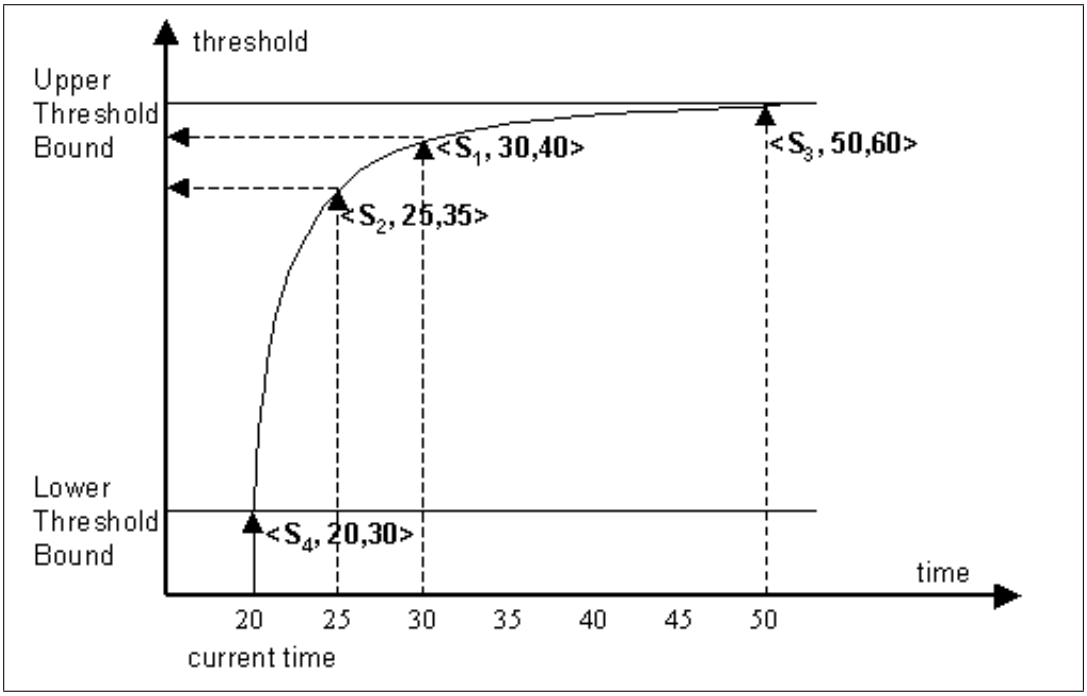

Figure 3.1: Generation of update thresholds in AMM.

$< S_1, 30, 40 >$

$< S_2, 25, 35 >$

$< S_3, 50, 60 >$

$< S_4, 20, 30 >$

In Figure 3.1, the update threshold values of mobile objects S_1, S_2, S_3 and S_4 determined by using Equation 3.1 are shown.

The computed threshold values are sent to the mobile clients and the clients generate their location updates according to these thresholds.

Chapter 4

A Criticality-Based Method for Generating Location Updates

In this chapter, we present the details of the location update generation method that we propose, called Criticality-Based Adaptive Monitoring Method (CAMM), together with discussion of our motivation.

4.1 Incorrect Information

As we stated in Section 2.3, the requesting mobile client may wish to make the location dependent query continuous, to be evaluated continuosly at each time instance. Obviously, the main purpose of submitting a query as a location dependent continuous query 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 [LULCL01]. In both AMM and our model, main objective is to ensure the correctness of the query results by just bounding the uncertainty in locations of selected objects as the result of the query. Obviously, for those objects which are not selected for any query or selected but have far *begin times*, the uncertainty in their locations will not affect the correctness of query results significantly.

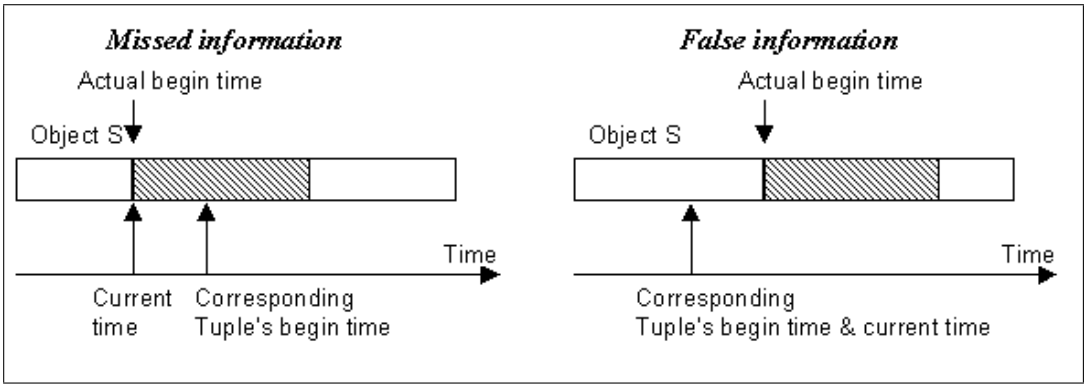

Figure 4.1: Missed and false information.

It becomes so important to ensure the correctness of the query results returned to the requesting mobile clients since they may make incorrect actions based on the received incorrect results from the database server. If the mobile computing system is related directly to human being, then correctness requirements of this model become vital.

If a query result provides false information, either in begin/end times or in value, the query result is accepted as *incorrect*. In [LULCL01], the *actual begin time* of an object is defined as the time when a moving object starts to satisfy the conditions of a query. It is expected that the database server should send to the requesting client the same *begin time* value as the *begin time* of the corresponding tuple before the *actual begin time*. Otherwise, a mobile client observes incorrect results as:

(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* (this problem is called *missed information*),

(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 (this problem is called *false information*).

In the first case, the mobile object satisfies the query but the requesting client does not aware of that situation (Figure 4.1). In the second case, the requesting mobile client is informed that an object has met the condition of its query but actually it does not. Transmission delay and out-dated database

state are the major causes for those incorrect results.

4.2 Motivation

In *AMM*, all moving objects in the coverage area of the information server are divided into two sub-sets. For each selected mobile object, a tuple is generated to be placed in the answer set *AnswerCQ* of the continuous query *CQ* and those objects which are in the *unselected object set* are neglected in *AnswerCQ* until they move to the *selected object set*. *AMM* assigns upper threshold bound value to mobile objects in the *unselected object set* for the sake of decreasing the update cost by increasing the cost of missed information. Besides, *AMM* generates different update threshold values for each mobile object in the *selected object set* according to their *begin times* but all mobile objects update their location information using the same method.

However, some critical systems which are related to health care, human beings, military and national security systems (e.g., ambulatory service of a hospital, battlefield monitoring systems) etc., must monitor the possible tuples in the answer set closely. In order to cope with the emergency as soon as possible, these systems need instantaneous and accurate information to make correct decisions about the situation they involved.

For example, the location dependent continuous query (LDCQ) "check for a free ambulance within 5 miles (to pick up a patient) of the museum" should be more critical than the LDCQ "check for a free taxi-cab within 10 miles (to pick up a tourist)". Another example to critical queries might be "retrieve the enemy tanks within 2 miles of the headquarters". In these critical LDCQs, locations of all possible ambulances (or tanks) should be monitored more closely.

In order to get more precise and timely results to some LDCQs, which have high critical levels, we propose a new location update generation method: *Categorized Adaptive Monitoring Method (CAMM)*. Besides minimizing the location update cost, *CAMM* aims to maximize the correctness of the results

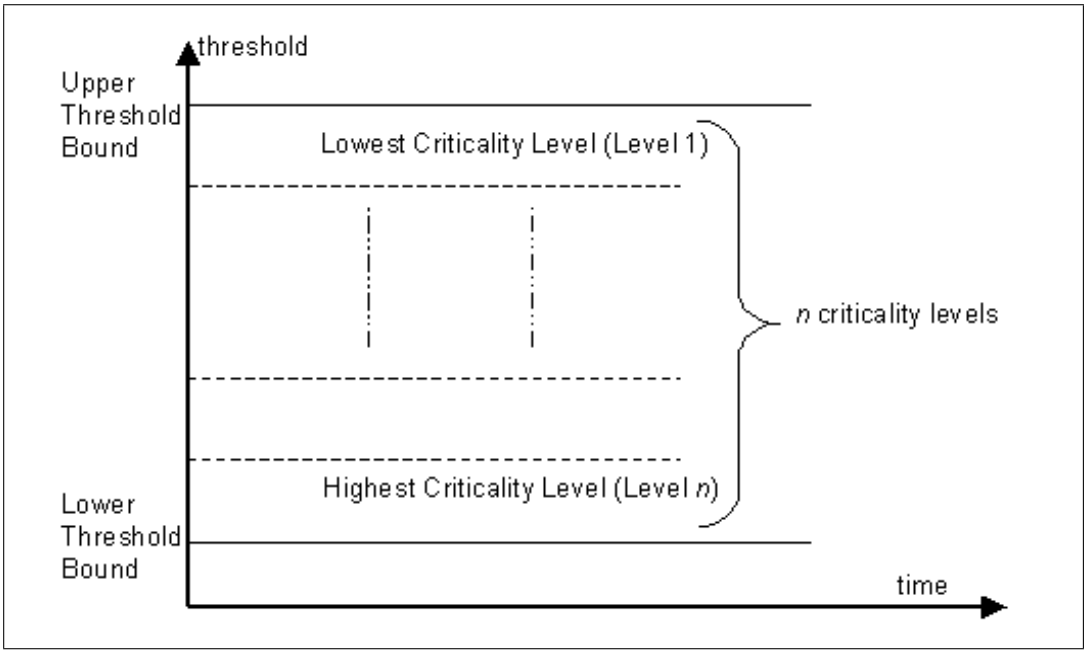

Figure 4.2: Associations of criticality categories for location dependent continuous queries

returned to the requesting client for any LDCQ categorized with higher criticality. We defined n criticality levels for the clients generating the LDCQs at a mobile computing system (Figure 4.2). Any mobile client in that system may generate LDCQs with one of the criticality categories of its own criticality level. For example, ambulatory service of a hospital may operate in a mobile environment with criticality level n=4 and may generate a LDCQ with a criticality category 1, 2, 3 or 4 in that level. Update threshold values of the objects in the answer set of a query are determined by the criticality category of the query. Our emphasis has been on providing relatively timely and correct results to LDCQs considering their criticality categories. The *MOST* data model [SWCD97] is used to specify the location information for the moving objects.

4.3 Determination of Location Update Threshold with Criticality Levels

In implementing *CAMM*, the mobile system described in Chapter 1 is assumed. The system consists of a database server and a number of moving objects. The

database contains the data items (including location information) issued with the moving objects. Moving objects generate updates to report their current locations to the database server. Each mobile client may generate location dependent queries with different criticality categories. We defined n=4 levels of criticality for the mobile clients generating LDCQ's. A client in criticality level 1 receives the same answer set to its LDCQ with the answer set generated with *AMM*. A moving client with criticality level 2 may generate a LDCQ either with a criticality category 1 or criticality category 2. In this case, LDCQ with criticality category 2 generates more accurate and timely results than the LDCQ with criticality category 1.

In *AMM*, update threshold bounds are defined for each moving object and if the deviation is greater than the update threshold, a location update is generated within these bounds. The update threshold of an object is determined based on the *begin time* of the object by using Equation 3.1.

In *CAMM*, the same update threshold bounds are used for each object. If the deviation is greater than the update threshold, a location update is generated within these bounds according to the criticality category of the LDCQ by using Equations 4.1 and 4.2.

$$C_{up} = H - (C_c - 1)\left(\frac{H - L}{C_l}\right), \tag{4.1}$$

$$C_{lw} = L, \tag{4.2}$$

Update threshold of a mobile object

$$x = [C_{up}] - \left[(C_{up} - C_{lw}) \times e^{-\delta t}\right], \tag{4.3}$$

where

C_{up}: Criticality Category Upper Bound,

C_{lw}: Criticality Category Lower Bound,

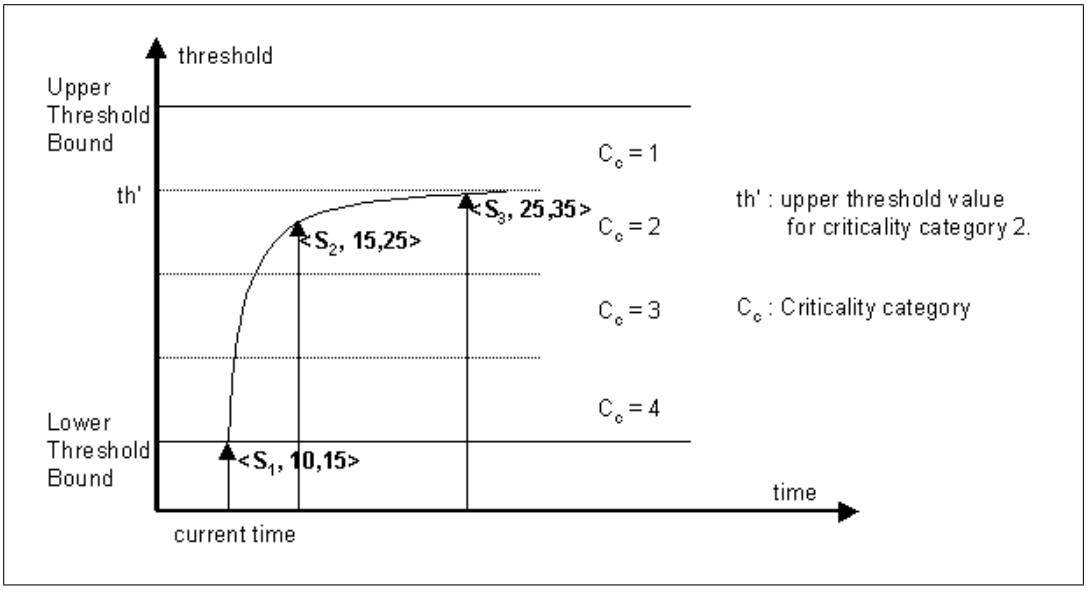

Figure 4.3: An example to CAMM.

H : Upper Update Threshold Bound,

L : Lower Update Threshold Bound,

C_c : Criticality Category of the query generated,

C_l : Criticality Level of the mobile client generating the query, and

δt : *begin time* of object $x - currenttime$.

As an example, suppose that client MC_1 with criticality level 4 has submitted a LDCQ with criticality category $C_c = 2$. In the first evaluation of the query at time 10, the tuples and update thresholds generated for mobile objects S_1, S_2 and S_3 are shown in Figure 4.3. According to the specified criticality category, MC_1 will receive an answer set by using threshold bounds L and th' corresponding to C_{lw} and C_{up}, respectively.

Chapter 5

Performance Experiments

In this chapter, we present the performance results for the Category-based Adaptive Monitoring Method (CAMM) that we proposed for generating location updates in processing LDCQs. We have designed and implemented a detailed simulation model to study the performance of the proposed method, CAMM, as compared to AMM. The simulation program was written in CSIM18 [S96]. The simulation model and the performance results are provided in the following sections.

5.1 Simulation Model

Our simulation model is based on the performance models proposed in the previous related work [GU00, LULCL01]. These models have been extended to support modeling of processing LDCQs with different criticality categories. The system consists of an information server and a number of moving objects. Mobile objects and the information server communicate with each other via a low bandwidth network such as the GSM network [MP92]. The information server maintains a database which contains data items for recording the locations of the moving objects in the mobile system. Mobile clients may generate LDCQs on other moving objects. Generated queries are transmitted to the information server via wireless network. LDCQs are submitted each with

a start time and an end time as mentioned in Section 3.1. The information server evaluates these queries when there is any change in the database state during the period of time between *start_time* and *end_time*, and transmits the computed results to the requesting client according to a query result transmission approach mentioned in Section 2.5. The results are in the form of a set of tuples, each indicating the beginning of the time period for which the object specified in the tuple satisfies the condition of the query. Each mobile object generate updates to report their current locations to the information server. The information server receives the updates, and updates the location of the sending mobile object in the database using the new value of location data.

5.2 Performance Criteria

A number of simulation experiments have been conducted to study the performance as a function of incorrect information rate (IIR) which is the number of occurrences of false and missed information over the total number of tuples generated. Experiments were designed to evaluate the performance of CAMM, in terms of the accuracy of query results and communication overhead imposed on the wireless communication. As our main performance measure, IIR indicates the capability of the system in providing correct information to the LDCQs generated by the mobile clients. In addition to IIR, we also consider the performance measures of update workload in terms of the CPU utilization and the retransmission rate (RR). RR is defined as the total number of tuple retransmission over the total number of tuples transmitted [LULCL01]. During the processing of a continuous query, a tuple may need to be retransmitted due to the changes in the data values. Obviously, as the RR increases, the communication overhead increases. Finally, we study the CPU Utilization which is defined as the system load introduced by location updates of moving objects and processing of continuous queries. The number of mobile objects in the system, maximum query duration, the number of mobile clients are among the input parameters for the performance experiments (Table 5.1).

Figure 1.2 depicts the architecture of the mobile computing system assumed in our experiments. All of the experiments were performed on SunSparc

Table 5.1: Model parameters

SYSTEM PARAMETERS	
Exec_time	Execution time of the simulation
NumCPU	Number of CPUs
Cpu_sche	CPU scheduling option
SERVER PARAMETERS	
Comp_time	Time for processing a tuple
MinULTh	Minimum threshold limit for location updates
MaxULTh	Maximum threshold limit for location updates
WIRELESS COMMUNICATION NETWORK MANAGER	
Comm_time	Communication delay for sending a tuple
MOBILE OBJECT PARAMETERS	
TotalObject	Total number of mobile objects
ReqMobObject	Total number of mobile objects which may generate LDCQ
MaxCQLife	Maximum life of a LDCQ
MinCQLife	Minimum life of a LDCQ
WaitingTime	Minimum time interval between two consequent LDCQs generated by the same mobile object
SpeedofObject	The speed of a mobile object

Workstations running SunOS, using the CSIM18 [S96] simulation package.

The simulation model consists of three basic components:

* Server Model,

* Wireless Communication Network Manager, and

* Mobile Client Model.

In the following sub-sections, we describe each component in detail.

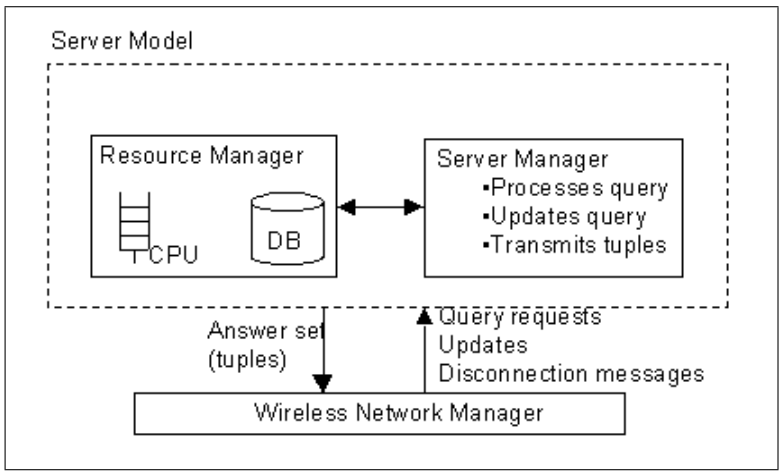

Figure 5.1: Server model

5.3 Server Model

The server model in our simulation has two modules as shown in Figure 5.1: A server manager and a resource manager. The server manager processes LDCQs received from mobile clients through the Wireless Communication Network Manager. The server manager also updates the LDCQs whenever a change occurs both in the locations of selected mobile objects for that query and in the location of the mobile client which has submitted the query. The resource manager controls the access to CPU and the database in the server model.

The system has *NumCPU* CPUs using *Cpu_sche* scheduling option. The database is modeled as a collection of *TotalObject* objects. The CPU time for processing a tuple is specified by *Comp_time*. Threshold limits for generating location updates are specified by using parameters *MinULTh* and *MaxULTh*.

5.4 Wireless Communication Network

The wireless communication network module carries all the traffic between server model and mobile object model. Communication delay in Wireless Communication Network Manager is specified by *Comm_time*. When a LDCQ is submitted from a mobile client, wireless communication network manager transmits this request to the information server in *Comm_time* seconds. The

information server computes the answer set and sends all of the tuples to the requesting mobile client via wireless communication network manager.

5.5 Mobile Client Model

The third part of our simulation model is the Mobile Client model which consists of three components as shown in Figure 5.2: a Resource Manager, a Query Generator and an Update Generator. These three sub-parts communicate with the server via wireless communication network manager. The resource manager models the CPU at the client machine, processes LDCQs and transmits the query to the server. The resource manager also receives the tuples in the answer set of the query and presents these tuples to the mobile user. The query generator generates LDCQs. We assume that there are *Total_Object* mobile objects and *ReqMobObject* of those objects, which are called mobile clients, can generate LDCQs. All of the LDCQs generated by the query generator are sent to the information server through the resource manager. The information server evaluates each query and sends the tuples in the result set through the wireless communication network manager. The resource manager receives these tuples and presents the answer set to the mobile user. The lifetime of a LDCQ is chosen randomly between *MinCQlife* and *MaxCQlife* [LCLUL01]. Once the lifetime of a LDCQ is chosen, this query is continuously evaluated throughout its life time in the information server. It is assumed that each mobile client can generate a new LDCQ after a waiting time following the completion of its previous LDCQ. The waiting time is denoted by *WaitingTime*.

In our simulation model, each mobile object is assigned a default speed denoted by *SpeedofObject*. The update generator generates update for an object when either:

(1) the deviation in the location of the object is greater than its update threshold, or

(2) the current time becomes equal to the last update time + current update period,

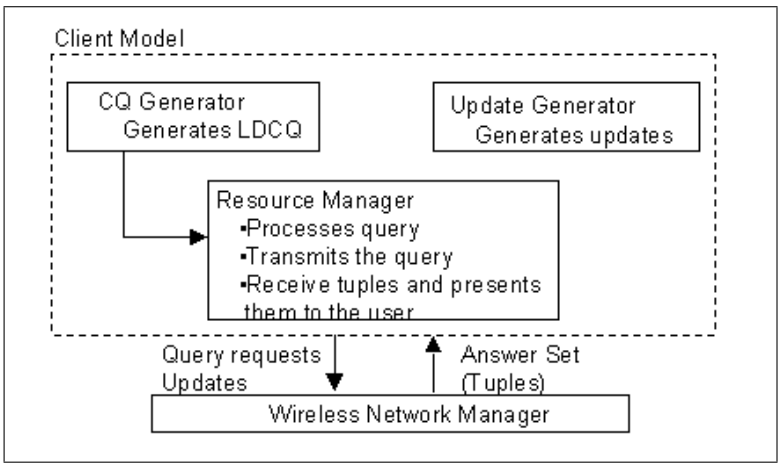

Figure 5.2: Mobile client model

whichever is earlier. In the former case, each mobile object calculates the deviation of itself in location since the last update. Whenever the deviation exceeds the system defined update location threshold, mobile object generates a location update to the information server. In the latter case, the value of the current (say, i^{th}) update period P_i of a moving object is determined by the following formula:

$$P_i = P_{i-1} \times U_i / U_{i-1}, \tag{5.1}$$

where U_i is the i^{th} update threshold. At each update period, the new update threshold U_i and update period P_i are sent to the moving object so that the object can calculate the next update time.

5.6 Experiments and Results

The values of the simulation parameters were choosen so as to be comparable to the related simulation studies [LULCL01, GU00]. The first set of experiments compare the performance of AMM with CAMM when different values are used for the lower threshold bound for both methods. The second set of experiments examine the effect of the number of mobile objects on both methods. In the third set, the minimum continuous query life is reduced to 60 seconds instead of 240 seconds which was the default value. On the contrary, in the forth set

Table 5.2: Model parameter values

SYSTEM PARAMETERS	
Exec_time	7 hours
NumCPU	1
Cpu_sche	EDF
SERVER PARAMETERS	
Comp_time	0.05-0.1sec.
MinULTh	5 sec.
MaxULTh	40 sec.
WIRELESS COMMUNICATION NETWORK MANAGER	
Comm_time	0.1-0.2sec
MOBILE OBJECT PARAMETERS	
TotalObject	200
ReqMobObject	50
MaxCQLife	360sec
MinCQLife	240sec
WaitingTime	1 sec
SpeedofObjcet	40 km/h.

of experiments, the maximum continuous query life is increased to 600 seconds instead of the default 360 seconds. In the fifth set, we examine various cases of ratio of mobile objects which may generate continuous queries by varying this ratio from 5% to 75% of all mobile objects. Since there is no real data available for modeling the tuples in the answer set of a CQ, we are concerned here with performance trends rather than exact performance predictions.

Table 5.2 provides the values of simulation parameters which are common to all experiments. The length of each simulation is 7 hours. The computation time, i.e., the time for processing a tuple, is selected from the range between 0.05 seconds to 0.1 seconds. The lower update threshold is varied from 5 seconds to 40 seconds in order to examine how lower threshold bound affects the performance of both methods. The communication time, i.e., the communication delay for sending a tuple, is chosen from the range between 0.1 seconds and 0.2 seconds. The number of mobile objects, is set to 200 objects for all experiments except where otherwise specified. In the second set of experiments, the total number of mobile objects is varied from 100 objects to 600 objects in order to examine the effect of the mobile object number on the performance.

The number of mobile objects which may generate LDCQ is 50 objects which is 25% of all the mobile objects. In the fifth set of experiments the percentage of the number of mobile objects which may generate LDCQs is varied from 5% to 75%. The continuous query life is chosen randomly between 240 seconds and 360 seconds for all experiments except the third and fourth sets. In the third set, the minimum life time of a LDCQ is decreased to 60 seconds, and in the fourth set of experiments, the maximum life time of a LDCQ is increased from 360 seconds to 600 seconds. The information server has a single CPU.

5.7 The Base Experiment

We first examine the performance results of the proposed criticality-based adaptive monitoring method under varying values of the lower update thresholds by changing the corresponding parameter value from 5 seconds to 40 seconds. Figure 5.3 depicts the incorrect information rate (IIR) obtained when different values are used for the lower threshold bound for both AMM and CAMM while upper threshold bound is fixed at 40 seconds. It can be seen that the performance of CAMM is much better than AMM, i.e., the IIR for various criticality levels tried with CAMM is smaller than that of AMM. All of the curves are in V-shape and that's what we expected. Although the smallest IIR value obtained with criticality level 3 is with minimum threshold of 7.5 seconds, the best performance is achieved with criticality level 2 when the threshold value is greater than 10 seconds. The poor performance obtained with very small threshold values is due to the heavy update workload as can be observed in Figure 5.4 in which the update workload is around 60% for small thresholds for both methods. Obviously, under such heavy workload, most of the system resources are devoted to process the updates from mobile objects due to smaller threshold values and the system will not be able to generate timely responses to the continuous queries from mobile clients. Consistent with our expectation, this situation causes an increase in incorrect information rate which is defined in Section 4.1 as missed and false information. On the other hand, when a large threshold is used, mobile objects update their location with greater time intervals and this causes the probability of providing

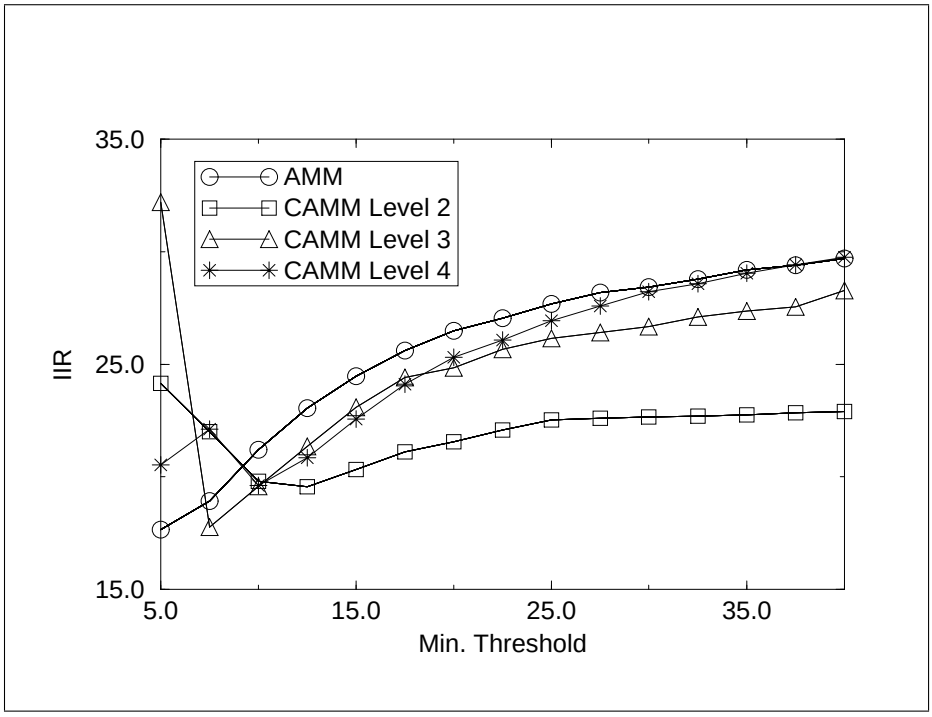

Figure 5.3: IIR for base experiment

incorrect information to mobile clients to increase. Figure 5.4 presents the CPU utilization results. As the lower threshold limit is increased, the curves start to move downward due to decreasing update workload. CAMM with criticality level 2 outperforms both AMM and all other criticality levels experimented with CAMM. Figure 5.5 shows the retransmission rate (RR) results that are consistent with those obtained for IIR and CPU utilization. The high RR values with small threshold values is again due to heavy update workload. With increasing threshold values, we obtain better results in terms of RR.

The RR values obtained with all criticality levels of CAMM are worse than those of AMM. The reader may notice from Figure 5.5 that RR obtained with criticality level 2 of CAMM is the highest among the others. This may seem contradictory to the results presented in Figure 5.3 and Figure 5.4. However, as the level of criticality in a mobile computing system increases, the threshold values for location updates decrease and this lowers the rate of retransmissions. On the other hand, while increasing criticality level and thus decreasing the location update threshold values, at the same time we increase the false information which is one of the factors considered in determining IIR. Due to this situation, AMM produces less number of retransmissions compared

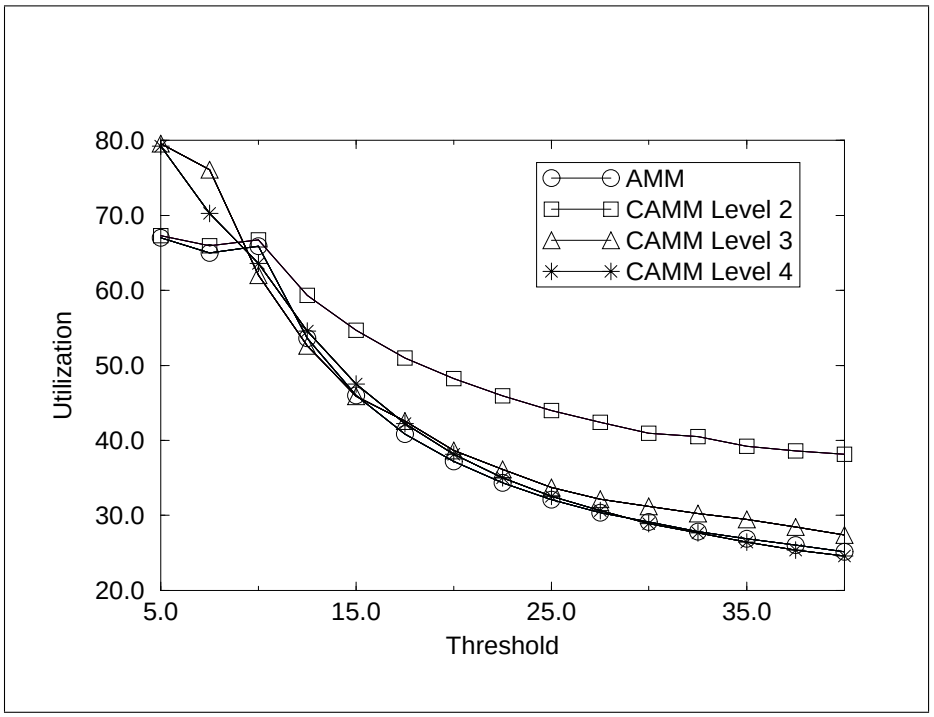

Figure 5.4: CPU utilization for base experiment

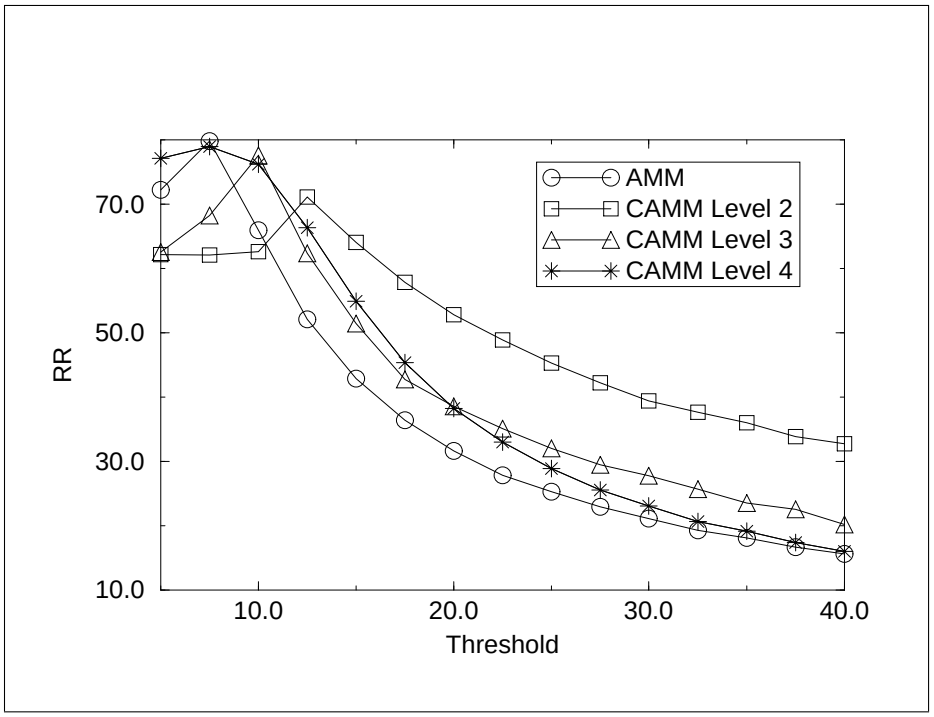

Figure 5.5: RR for base experiment

to all criticality levels of CAMM.

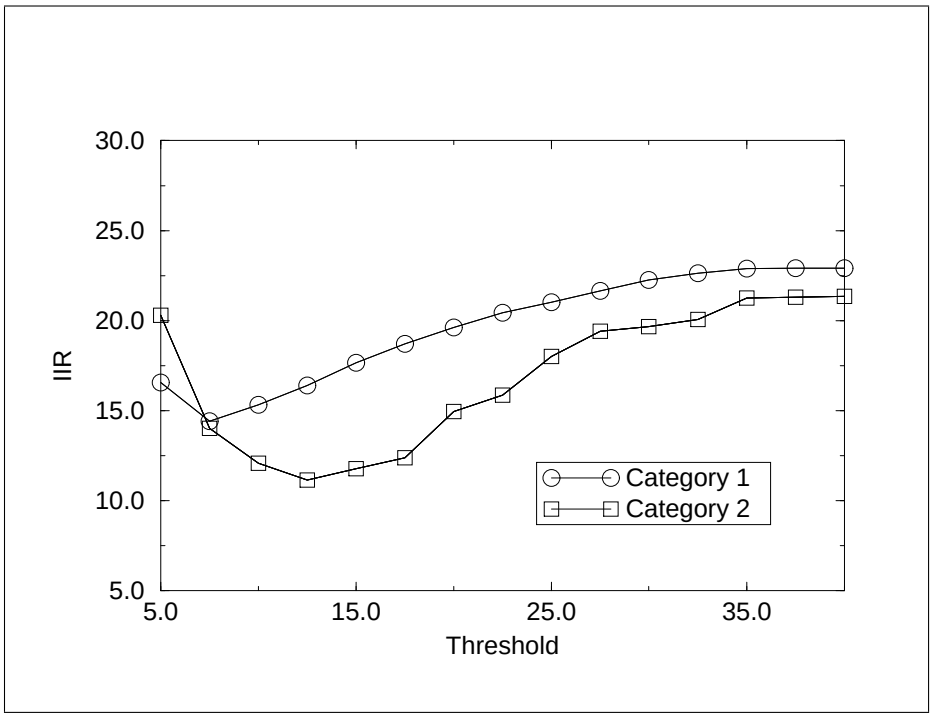

Figure 5.6: IIR for criticality level 2

The results we have discussed so far show that CAMM with criticality level 2 provides better performance, in general, than all other levels of CAMM and AMM. In Figure 5.6, IIR evaluation of criticality level 2 is provided to show the difference between the performance of category 1 queries and category 2 queries (which are more critical). Obviously, as expected, criticality category 1, which has higher upper threshold limit, produces higher IIR values than those of criticality category 2. So, more accurate results are provided to more critical category 2 queries.

5.8 Evaluation of the Impact of Number of Mobile Objects

In this section, we investigate the impact of the number of mobile objects in the mobile computing system on performance metrics IIR, RR and CPU Utilization. The number of mobile objects in the system is varied from 50 to

600. As expected, the higher the number of mobile object, the higher IIR and the system workload. On the other hand, we observe that RR decreases as the number of mobile objects increase. As the system moves into a heavy workload caused by increasing number of mobile objects, most of the system resources are devoted to process the updates from mobile objects, and the system becomes not able to generate timely responses to the continuous queries from mobile clients.

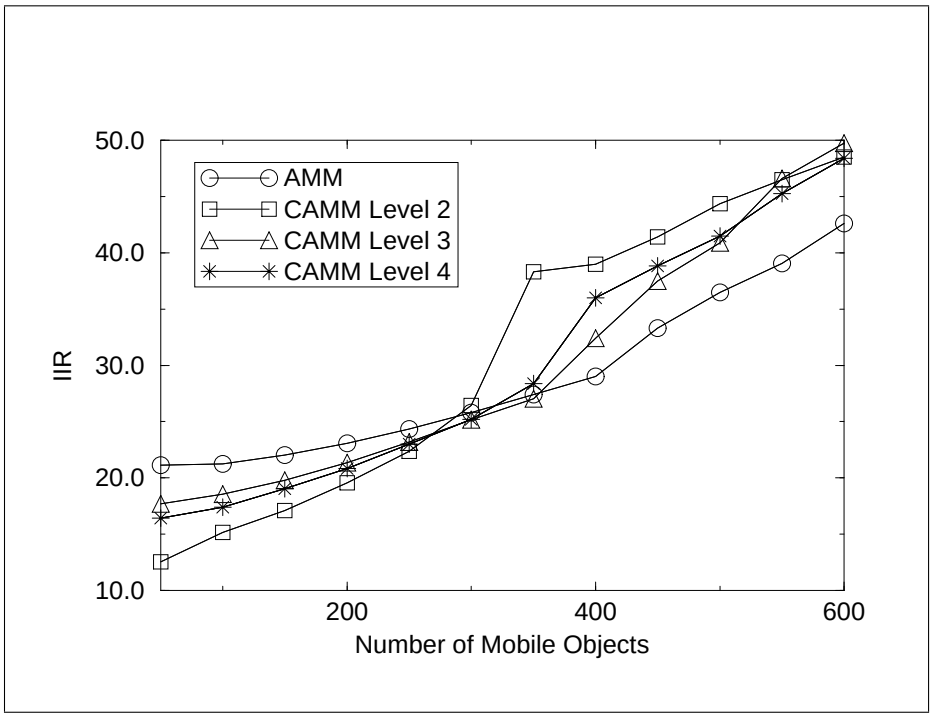

Figure 5.7: IIR for different number of mobile objects

The system workload increases up to a certain point as the number of mobile objects increases, as shown in Figure 5.8. After that particular point, the system workload remains at maximum. Obviously, after that point, in Figure 5.7, IIR begins to increase, and in Figure 5.9, RR begins to decrease dramatically.

5.9 Evaluation of the Impact of CQ Life Time

In this section, we examine the performance impact of the CQ life time as the minimum update threshold is changed. We performed two experiments, and

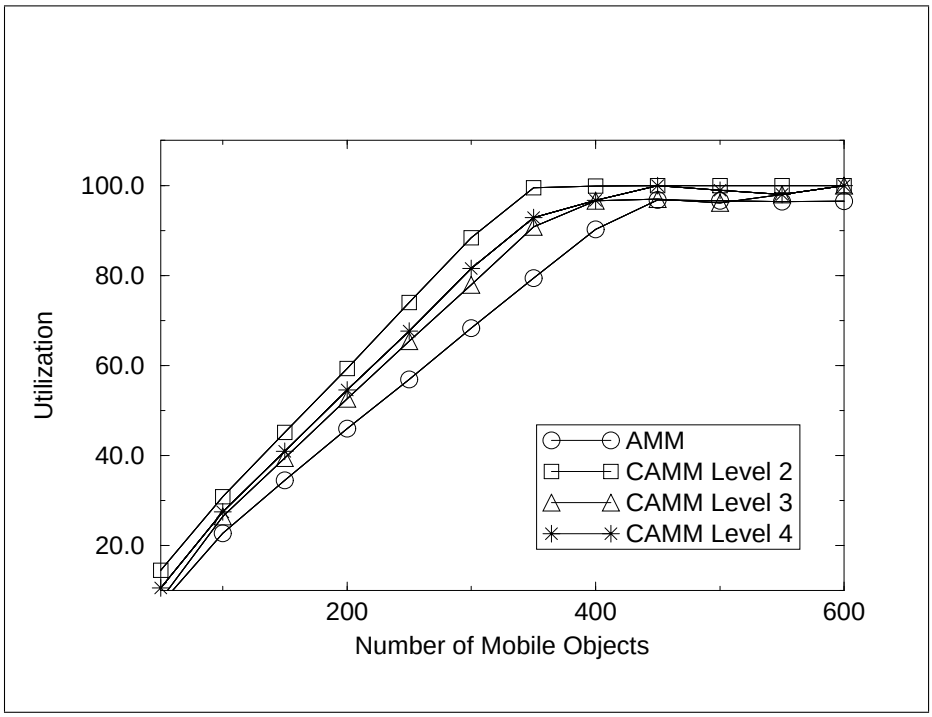

Figure 5.8: CPU utilization for different number of mobile objects

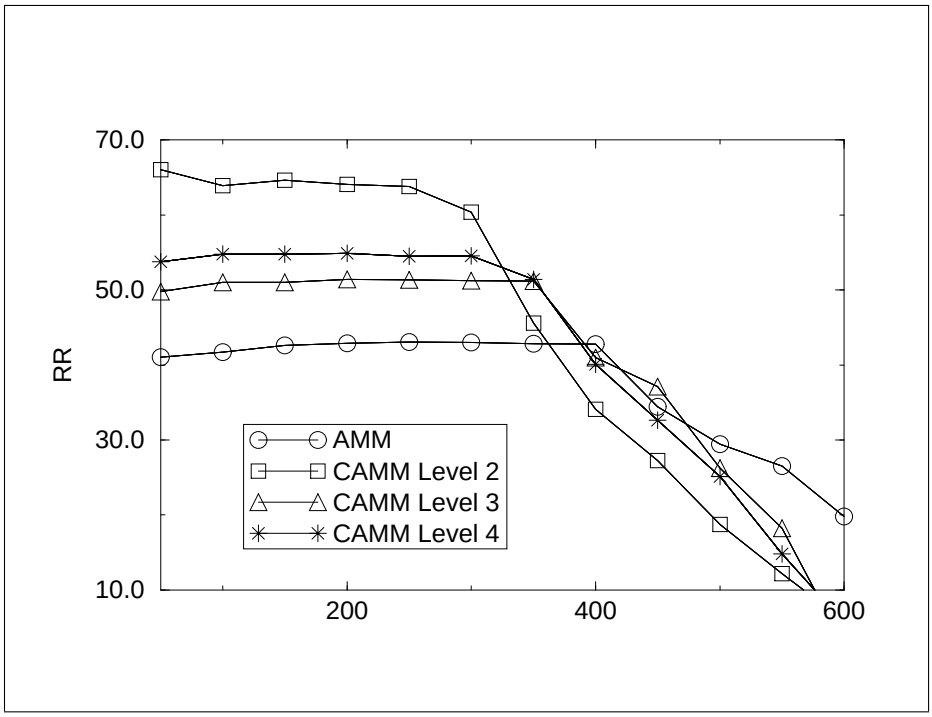

Figure 5.9: RR for different number of mobile objects

Figure 5.10 shows the performance results.

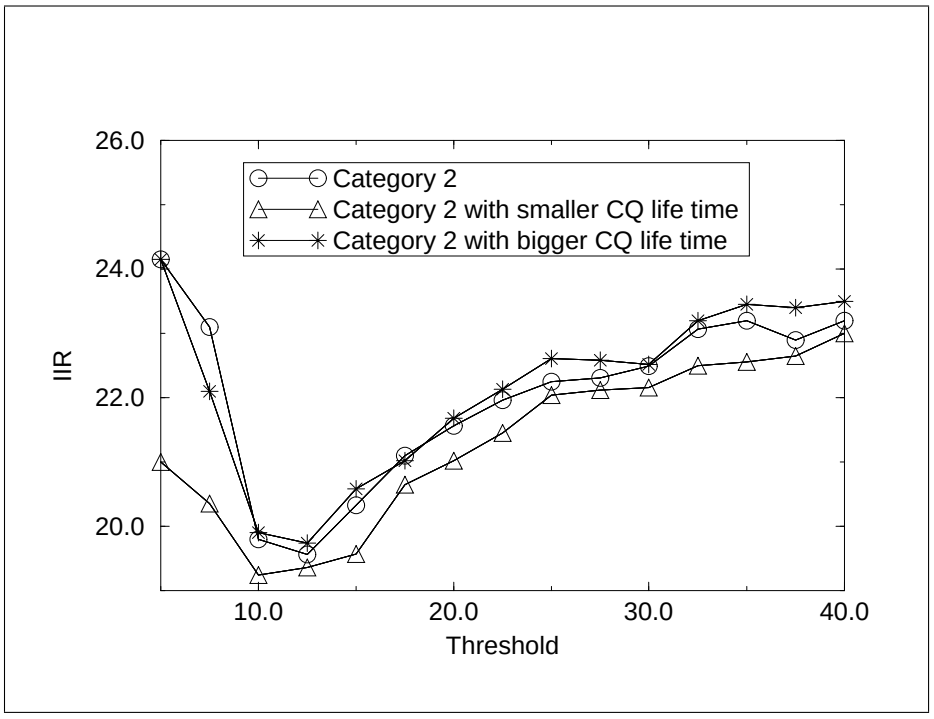

Figure 5.10: IIR with changing CQ life time

In the first experiment, we have reduced the CQ life time by setting lower and upper bounds to 60-240 seconds. In the second experiment, we increased these bounds to 240-600 seconds. The performance patterns are very similar to that of Figure 5.3 as shown in Figure 5.10. The lower IIR obtained with shorter CQ life time is due to reduced system workload. As the system workload decreases, system resources are devoted to generate more timely responses to the continuous queries. In the contrary, as the system workload increases with longer CQ life times, IIR increases. The results of in terms of the other metrics i.e., RR and CPU Utilization, are not displayed, as they lead to similar observations with IIR results.

5.10 Evaluation of the Impact of Number of Mobile Clients

We have also investigated the performance impact of the number of mobile clients which may generate LDCQs. Figure 5.11 shows the IIR results obtained when the minimum location update threshold is fixed at 12.5 seconds and the percentage of the number of mobile clients within all mobile objects is varied from 5% to 75%.

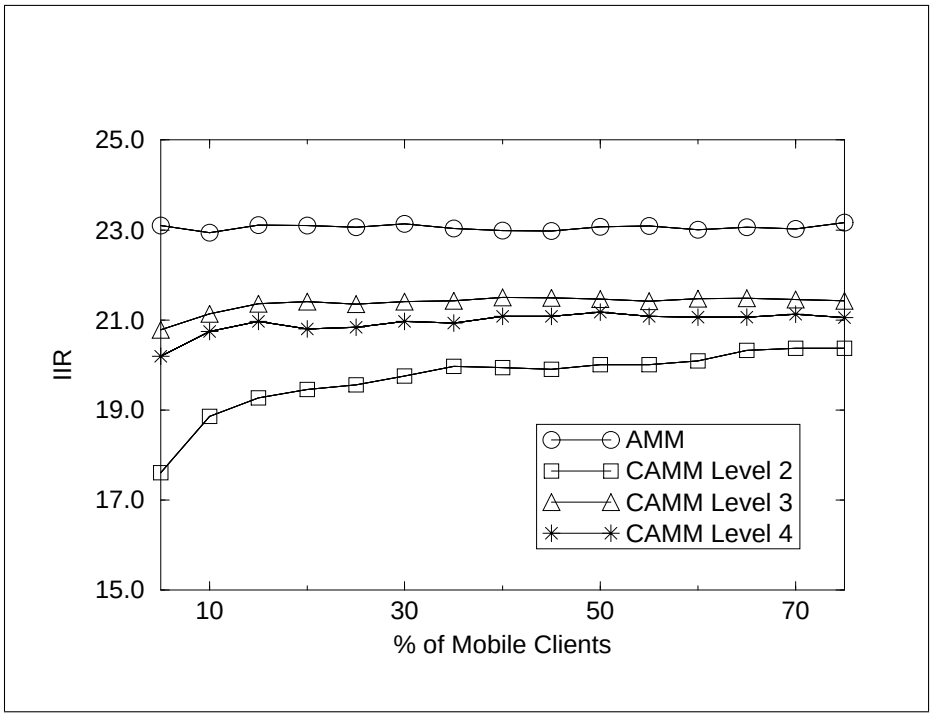

Figure 5.11: IIR for different number of mobile clients

Our observations with the previous experiments are also valid in this experiment. The best results in terms of IIR are obtained with the criticality level 2 of CAMM. All the criticality levels of CAMM produce higher IIR with larger number of mobile clients. The base experiment was conducted with 25% mobile clients. The relative performance of the methods and different criticality levels does not change with changing percentage of mobile clients.

Figure 5.12 presents the system workload results as a function of the increase in percentage of mobile clients. As expected, CPU Utilization values

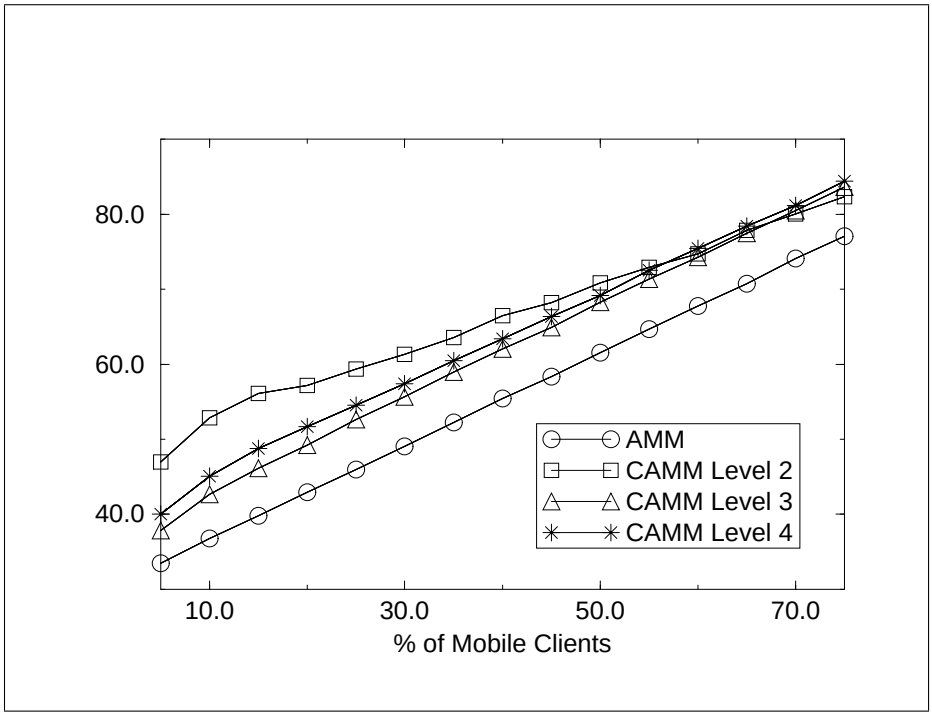

Figure 5.12: CPU utilization for different number of mobile clients

for all criticality levels and AMM increase as the ratio of mobile clients over the total number of mobile objects increases. We observe that CAMM with any level leads to higher CPU utilization compared to AMM. Criticality level 2 of CAMM performs worse than the other levels until 60% mobile clients, and after that point all levels of CAMM perform very similar to each other. This is consistent with the results shown in Figure 5.11 where level 2 gets closer to levels 3 and 4 after 60%.

Chapter 6

Conclusion and Future Work

In a mobile computing system, mobile hosts can access information via submitting queries to the information server over the wireless network. Some of these queries are called *location-dependent queries* as the answer to such kind of queries depends on the current location of the user who issued the query [SWCD97, SWCD98,WXCJ98]. An important issue that should be considered in designing a mobile computing system is to provide timely and accurate support for processing of location-dependent continuous queries. Because of the natural dynamic features of the stored data items (e.g., position of objects) in a mobile computing system, the database records should be refreshed frequently. Otherwise, the values of data items may become outdated and the results of location dependent continuous queries may not be accurate.

In order to reduce the cost of frequent position updates, in Moving Objects Spatio-Temporal (MOST) data model, the current position of a moving object is modeled as the distance from its starting point, along a given route [SWCD97]. However, the actual position of a moving object may deviate from the position computed by the DBMS, due to the fact that the moving object does not travel exactly on the given route with a constant speed. *Adaptive Monitoring Method* (AMM) was proposed with the aim to reduce the update workload and at the same time closely monitor the locations of moving objects [LULCL01]. In AMM, moving objects generate updates to report their current locations to the database server with a time-stamp which specifies the

time when the current value will become valid. The database server assigns different location update threshold to each mobile object.

In this thesis, we have presented a new method for the generation of updates in processing location dependent continuous queries. It is obvious that the query submitted by an ambulance to pick up a patient must have higher priority compared to a query submitted by a taxi driver. Additionally, the same user may want to get more accurate and timely results for some of his/her queries. Unlike AMM which our work is based on, in CAMM the mobile user may categorize his/her queries submitted as LDCQ to the information server. The categorization of queries enables the system to provide more accurate and timely results as the criticalness of queries increases.

We have provided a detailed simulation model with multiple criticality levels and evaluated the performance of the mobile system and the proposed method. In particular, we have examined the effects of varying the update threshold bound, the number of mobile objects, the length of continuous query life time, and the number of mobile clients in terms of incorrect information rate, retransmission rate and update workload.

The overall performance results indicate that, although the message retransmission rate and update workload increases a little bit when CAMM with any criticality level, rather than AMM, is employed; a considerable improvement in the performance in terms of incorrect information rate is possible with CAMM. The best results, i.e., most accurate query results, are obtained when only two levels of criticality are considered with CAMM.

There are some related issues that could be explored as future work. One possible direction of future research is to extend our simulation model with various transmission methods for query results in order to reduce the message retransmission cost and update workload that we faced.

Bibliography

[B99] D.Barbara, Mobile Computing and Databases - A Survey, IEEE Transactions on Knowledge and Data Engineering, Vol.11, No.1, pp.108-117, 1999.

[DCKV97] A.Datta, A.Celik, J.Kim, D.E.VanderMeer, Adaptive Broadcast Protocols to Support Power Conservant Retrieval by Mobile Users, Proceedings of 13th International Conference on Data Engineering, Birmingham, UK, 1997.

[DK98] M.H.Dunham, V.Kumar, Location Dependent Data and its Management in Mobile Databases, Proceedings of International Workshop of Database and Expert Systems Applications, pp.414-419), 1998.

[GU00] H.G.Gök, O.Ulusoy, Transmission of continuous query results in mobile computing systems, Information Sciences, Vol.125, No.1-4, pp.37-63, 2000.

[IB94] T.Imielinski, B.R.Badrinath, Mobile Wireless Computing: Challenges in Data Management, Communications of the ACM, Vol.37, No.10, 1994.

[KU99] E.Kayan, O.Ulusoy, An Evaluation of Real Time Transaction Management Issues in Mobile Database Systems, The Computer Journal, Vol.42, No.6, pp.501-510, 1999.

[LAC99] K.Lam, M.Au, E.Chan, Broadcast of Consistent Data to Read-Only Transactions from Mobile Clients, Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, Louisiana, 1999.

[LCY00] K.Lam, E.Chan, J.C.Yuen, Approaches for Broadcasting Temporal Data in Mobile Computing Systems, Journal of Systems and Software, Vol.51, No.3, pp.175-189, May 2000.

[LULCL01] K.Lam, O.Ulusoy, T.S.H.Lee, E.Chan, G.Li, An Efficiend Method for Generating Location Updates for Processing of Location-Dependent Continuous Queries, Proceedings of 7th International Conference on Database Systems for Advanced Applications(DASFAA'01), Hong Kong, April 2001.

[MP92] M.Mouly, M.B.Pautet, The GSM System for Mobile Communication, Cell and Sys, 1992.

[S96] H.D.Schwetman, CSIM18- The Simulation Engine, Proceedings of the 1996 Winter Simulation Conference, 1996.

[SWCD98] A.P.Sistla, O.Wolfson, S.Chamberlain, S.Dao, Querying the Uncertain Position of Moving Objects, Temporal Database: Research and Practice, pp.310-337, 1998.

[U98] O.Ulusoy, Real Time Data Management for Mobile Computing, Integrated Design and Process Technology, Vol.2, pp.233-240, July 1998.

[WCDJ97] O.Wolfson, S.Chamberlain, S.Dao, L.Jiang, Location Management in Moving Objects Databases, Proceedings of WOSBIS'97, Budapest, Hungary, pp.7-13, 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, B.Xu, S.Chamberlain, L.Jiang, Moving Objects Databases: Issues and Solutions, Proceedings of the 10th International Conference of Science and Statistical Database Management, pp.111-122, Capri, Italy, 1998.