THE HONG KONG POLYTECHNIC UNIVERSITY

DEPARTMENT OF COMPUTING

Efficient Location Management Techniques for Moving Objects in Mobile Environments

ZHOU Jing

A thesis submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

November 2007

Certificate of Originality

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

ZHOU Jing (Name of student)

Acknowledgments

I would like to express my deepest gratitude to my research supervisor Dr. Hong Va LEONG and co-supervisor Dr. Qin LU for their invaluable support, advice, insight and guidance throughout this interesting and challenging research project. In addition, I would like to thank senior research students, Mr. Ken Lee and Mr. Teddy Chow for their suggestions, sharing experiences given to me.

Last but not the least, I also would like to thank my parents, and my friends for their endless love, support and encouragement. This thesis is dedicated to all of them.

Abstract

Mobile computing has become a reality as a result of the convergence of two emerging technologies: the appearance of powerful portable computers and the development of fast reliable wireless networks. In this new computing paradigm, computing entities like resources and users (e.g. humans, cars) are not required to remain in a fixed position in the network but possess the freedom of mobility and portability as well as the ability to issue queries regarding other objects of interest which could also wander around. They are therefore called *moving objects*. The novel characteristics and abilities of these moving objects under the new computing environment have enabled an entire new promising class of applications, LDIS (Location-Dependent Information Services), operating within a mobile environment. As continuous movement is the essential feature of moving objects, location management plays a fundamental role in supporting efficient LDIS applications. From the literature study in this thesis, it has been found that despite various efforts made and achievements gained previously, there is still much uncovered room for efficient techniques to improve location management system performance. This thesis aims at combating the limitations of previous work and proposing efficient location management techniques. The design goals these techniques seek to achieve can be summarized from the view points of both moving objects and system requirements as a whole, including *query awareness*, *movement awareness*, *cost optimization* and *error tolerance*. To achieve these goals, three new location management models are proposed and evaluated, namely, the query-aware model, basic cost-based model and extended cost-based model. The proposed models fulfill the design requirements in two ways. First, the models with their associated schemes have lower communication costs (i.e. fewer update messages from objects moving in the system are needed for position tracking), which leads to lower energy consumption. Second, from the system point of view, optimal resource utilization is achieved. On

the one hand, the models would lead to a lighter work load at the server side. On the other hand, they also improve the efficiency of query processing with more precise query results generated and produce a higher service satisfaction level of the system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

1.1.1 Moving Objects and Location-Dependent Information Services

A demanding requirement to the information acquisition is known as "anywhere anytime" which is one of the driving forces for the dramatic development of technologies in wireless communication and portable computing devices. As a result of the convergence of these two technologies, mobile computing with the aim of providing a ubiquitous computing environment for mobile users has become a reality. In this new computing paradigm, objects of interest (e.g. humans, cars, laptops, desktops, pets, wild animals, bicycles etc.) are not required to remain in a fixed position in the network but possess unrestricted mobility and portability. They are therefore called *moving objects* (MO).

Moving objects in a mobile environment can generally determine their locations with the help of personal locator technologies, such as global positioning systems or cellular telephone technologies. By enabling an upward link, the location data sent from the moving objects to the server creates an environment in which objects are

aware of the locations of surrounding objects as well as other related information. These new characteristics and abilities of the new mobile environment enable an entirely new class of applications, **L**ocation **D**ependent **I**nformation **S**ervices (LDIS). Such applications include *location-aware advertising*, *digital battlefield*, *local news*, *weather querying services*, *tourist services*, *completely automated traffic and vehicle navigation systems* and many other directory services.

1.1.2 Location Management for Moving Objects

1.1.2.1 Location Management Issues

As continuous movement is an essential feature exhibited by moving objects in a mobile environment, *location management* plays a central role for providing efficient LDIS applications. In order to provide LDIS, systems should possess the ability to process location-dependent queries through the use of location information. For example, in a *traffic information system*, a typical location-dependent query about moving objects is "report all taxicabs which are within $500m$ of my current position". Such queries may be issued from a user holding a mobile device walking in the street. The answer to this kind of location-dependent queries may depend on the location of the query issuer and querying objects, and in this case, the service user and taxicabs running in the nearby region. Obviously, the locations of these moving objects change continuously which is the most distinguishing feature of the objects of interest. As a result, to facilitate the support of location-dependent services, a crucial task is to maintain the up-to-date information about the locations of the moving objects. All these fundamental and central functionalities and problems are discussed and studied in the research field of *location management* for moving objects in mobile environments.

Location management has been extensively discussed and studied in personal communication networks (PCN) which commonly deploy the cellular architecture. Gen-

erally speaking, the problem involves two basic operations: *lookup* and *update*. A lookup operation is issued when a cellular user needs to be located for a call or a message. An update operation is issued when a user moves beyond the boundary of its current cell. The management problem addressed in the literature is on how to distribute, replicate, and cache the database of location records, such that the two types of operations can be executed as efficiently as possible.

Limitations of research work conducted in PCN are that they consider only the locations with cell resolution and the query to be processed is only a point searching query at the current time [61, 90]. In a general moving object environment, research work of location management deals with locations of finer resolution, process queries pertaining to the past or the future and provide sophisticated techniques for various types of more complicated queries [41, 88].

Overall, location management problems in moving object environments involve the interaction and data flow among several interrelated components, including the positioning technology, location modeling, the storing problem, location updating, and query processing.

Positioning technology is needed in the fundamental component to acquire the movement information of moving objects. Among several approaches, GPS is by far the dominant technology. *Location modeling* addresses the problem of how to construct position samples of each moving object acquired via technologies like GPS into useful and retrievable location information. The database may store various levels of location information and implement different location models for varied processing needs. The *storing problem* concerns the architecture of location databases. As objects are continuously moving in mobile environments, the main task in *location updating* is to update the location records stored in the database accordingly with the current positions. Query response time is the essential system performance metric for location-dependent services in a mobile environment containing a large number of moving objects. Therefore, *query processing*, with the support of efficient index-

ing methods on large volumes of location data and query optimization techniques also forms a crucial component in location management. The two key problems for location management for moving objects are **location updating** and **query processing**. In abstract terms, location management for moving objects involves two basic operations: *update* and *query*.

1.1.2.2 Common Location Management Models

In a mobile computing environment, there are two common location management models. The *client-server model* is applied when the system adopts a centralized data management approach where some computers are dedicated to serve the others, and they act as servers. The *peer-to-peer model* is applied with the distributed approach where every node is treated equally. Location updating and query processing activities in these two types of models function differently.

In the client-server model, the locations of moving objects are maintained by a location server, which supports location-dependent query processing. Moving objects may be passively tracked by the communication infrastructure or actively report their locations to the server. The main focus of this thesis is on the client-server model.

In the peer-to-peer model, moving objects play equal roles and exchange the information with one another freely. No centralized service center is provided. Queries are processed using distributed approaches.

These two common location management models are presented in detail in Chapter 3.

1.1.3 Uncertainty Problem in Location Management

Location management in abstract terms can be viewed as addressing the problem of providing uncertainty bounds for each moving object running within the whole working space covered by the service system.

In the PCN environment, the uncertainty bound of each mobile user is at the cell-granularity which is a sufficient resolution for calling and messaging. However, in LDIS, the uncertainty at the cell granularity is often insufficient. For example, a taxi-cab driver cannot know how to pick up a customer if he just knows the location of the passengers is within a cell whose coverage is 10 miles. To provide efficient uncertainty management in a much finer resolution is therefore a fundamental research problem for location management in general moving environments.

Uncertainty is an inherent property of location data. Consider a location management system which keeps the up-to-date location information about moving objects and processes location-dependent queries issued from the service user. All the moving objects have to continuously send updates of their positions to the system via a wireless communication link. Because of continuous motion and measurement/digitization errors, the location record is not always identical to the actual location of the object regardless of the methods used to update and track the moving object. In most cases, the location of an object is known with certainty only at the update time and the uncertainty increases until the next update. The uncertainty management has various implications for all the location management components and issues, especially for the two key management components, i.e. the location updating and query processing.

The focus of much research in the uncertainty management is on how to locate continuously moving objects efficiently without updating the location information whenever it moves. Obviously, the naive updating approach (e.g. sending an update report whenever the moving object changes its location) can remove the uncertainty but the excessive communication cost of the system and the energy cost of the moving objects make it unfeasible. Therefore, updating strategies are needed which are not only aware of the induced location uncertainty but also resource efficient.

As the stored location recorded in the database can vary from the real position, query processing mechanisms should take account of the uncertainty that may be involved in the results returned to the user. There is a risk that the uncertainty is unac-

ceptable and the results cannot meet service requirements. As a result, there is a need for strategies to handle the uncertainty and ensure precise query results precision.

Despite the kinds of techniques applied for updating and querying, the uncertainty in the database location always exists. Lowering this uncertainty would come at a cost. Consider an example where an update is issued when the location distance from the last update exceeds a threshold, say d. Lowering the value of d could decrease the uncertainty in the database location but the increase resource consumption such as bandwidth utilization and energy in moving object. Increasing d on the other hand enlarges uncertainty. This increases the risk of precise query processing but makes the resource consumption more efficient. The *tradeoff* between updating resource consumption and the uncertainty penalty should be appropriately quantified.

Preliminary studies of the uncertainty management seek to provide bounds on location uncertainty while minimizing the update overheads as much as possible. Extended studies make efforts to quantify the uncertainty and optimize the tradeoff. An example is the cost-based model [89] which determines the amount of the uncertainty for optimizing the resource consumption and query imprecision.

1.2 Limitations of Previous Work

Location updating and query processing are two key issues in location management. The goal for a good location updating design is to produce less network traffic and gaining more precise location information, whereas for query processing, the goal is efficient processing and precise results. To build a good location management scheme, both goals form the performance targets. Recall earlier that the *uncertainty problem* affects both updating and querying issues. To handle the uncertainty and build efficient and good updating and querying schemes, there are three approaches proposed in previous work. However, it has been found that all of the approaches have their own limitations and problems. The analysis and discussions are conducted in this section.

1.2.1 Approach I: Precision Assumption

Previous studies based on this approach to handle the uncertainty problem focus their attention on efficiently evaluating queries while simplifying the updating issues by assuming that the location information is always *accurate enough* for precise query results. They assume the straightforward time-based updating approach which updates the location information to the database periodically. The assumptions are made that the object location record stored at the database is 100% correct; indexing or other proposed methods are based on these "correct" location records. Under this assumption, the accuracy issue of the location information is completely a burden of the object performing the location update.

Drawbacks:

We know that in practical applications, when the location information is not accurate, the returned results for the queries issued by the system user may deviate from the actual values. This approach ignores the fact that updating methods should be aware of the uncertainty problem and make efforts to handle it.

1.2.2 Approach II: Quantifying the Uncertainty with Probability

Previous work adopting this approach provides probabilistic answers to queries [11]. The probability actually quantifies the uncertainty that may be involved in the location information and thus the query results. This approach aims at informing the service users of the existence of the errors that may affect their query results.

Drawbacks:

This kind of answer does not provide a perfect solution but a notice which reminds the query issuer of the error that may be involved in the query results. As a result, no control is provided to protect the user from the uncertainty problem in terms of both updating and querying issues.

1.2.3 Approach III: Balancing Tradeoff Introduced by Uncertainty

Intuitively, the tradeoff between the updating resource consumption and the uncertainty penalty always exists when setting the uncertainty bound. The smaller the bound is, the more update messages should be sent and the lower the position error bound is generated and vice versa. Bearing this fact in mind, the research work in this category studies how to establish the update frequency to balance the update communication cost and positioning error cost in query answering.

There are two methods to balance the tradeoff. The first method is to "sign" an agreement on the uncertainty bound between the service user and provider [73]. The bound is assumed to be a reasonable performance level accepted by the service user. The updating activity is then designed to ensure the bound which should be guaranteed in the location information provided. Queries are processed accordingly based on the uncertainty bound. As the agreement of the uncertainty bound is created beforehand, the service user is expected to be satisfied with the service provided.

The other method to balance the tradeoff is a cost-based method whose aim is to optimize the quantified tradeoff itself and achieve the optimal uncertainty bound [84]. In this approach, the optimal uncertainty bound is said to be achieved when the cost defined in the research model is minimized.

Drawbacks:

The assumption made by the first tradeoff handling method is that the service user can be satisfied by the mutually agreed uncertainty bound. As a result, the query precision is actually not taken into consideration. However, the query performance gain concerning communication cost can also be feasible when relating the querying process to the updating activity because tight monitoring for those seldom queried objects can be ignored by which large amount of resources saving can be achieved.

The second tradeoff handling method is a cost-based approach. It seems to be the best approach among all the previous work handling uncertainty problems. However,

previous work has several drawbacks.

First, much previous work considers querying and updating as two separate procedures for an application/system in a moving object environment, in which the uncertainty and deviation impose a cost or penalty in terms of incorrect decision making. Based on this, the information cost function integrates both update cost and penalty for the uncertainty in order to derive the minimal cost of a trip. Previous work assigns the optimal values to the object deviation threshold and then considers the query issued in a next step. However, in many practical systems, the only reason to provide the up-to-date location information is to provide precise answers to queries concerning these objects. Therefore, if no query is issued for the whole trip of the moving object, it is not necessary to produce the information for the non-existent "consumer".

Second, they suffer from the dependency on some predicted functions. The cost is adjusted to the current motion pattern, whose changes need to be reflected by parameter changes on the predicted function and this is hard to achieve.

Third, these policies are only applicable when the destination and motion plan of the moving objects are known. The route would be fixed and known to both the moving object and the server, and the update policy is used to revise the time when the moving object is expected to be at various locations along the route. However, the future route of a tracked object is not always known.

1.2.4 Discussion

Two design goals for a location management scheme are the generation of less network traffic (efficient location updating) and the provision of precise query processing. The drawbacks of previous approaches have impact on these goals. Although querying issues and updating protocols are closely related to each other, no effort for interaction has been made to improve the inaccuracy that may be introduced. This means that the location updates are always query blind though improvements can be made by making

use of the close relationship between the two key factors in location management.

The two goals can be fitted better when we notice that query processing is related to object updating and the performance of query processing depends on the location information the object tracking mechanism provides. Thus, separating query processing from object updating is not appropriate. The motivation of this research work is based on this observation. Combining updating and querying can lead to a novel model that on the one hand, the query precision requirement is met and on the other hand, the network traffic is reduced as much as possible.

1.3 Solution Strategies

1.3.1 Design Goals

This thesis aims at removing the limitations of previous work and proposing efficient location management techniques. The design goals these techniques seek to achieve can be summarized from the points of view of both moving objects and the system as a whole.

From the point of view of the moving objects, the proposed models can reduce the communication cost (i.e. fewer updates are needed for position tracking) and this reduction leads to lower the energy cost of each single moving object. Also, queries issued from these moving objects can be processed more efficiently and faster. Furthermore, the query results are expected to be more precise as a result of the application of those techniques.

From the system point of view, optimal resource utilization is expected to be achieved. The resource consumption for the system to conduct communication activities (e.g. bandwidth consumption) and handle a large number of continuously moving objects is reduced. This leads to a lower work load for a location management system as a whole. In one word, because of the efficient location management, the service satis-

faction level of the system could become higher.

1.3.2 Design Methodologies

This thesis studies several key research issues. First of all, the analysis on how location updating and query processing can affect each other and how the relationship can be used for more efficient location management is performed. Based on the analysis, the ultimate goal to optimize the resource utilization as well as maintaining the query processing service level is set up and several features that should be involved in the location management design in order to achieve the final goal are identified. These features are:

- **Query Awareness:** Query processing on moving objects is the main purpose of moving object location tracking with respect to many practical applications. Efficient location management schemes should consider the possible influences of query patterns to the location management schemes.

- **Movement Awareness:** If the current or future location of each moving object can be efficiently tracked based on some information either provided by the moving object itself or from the historical data, then continuously updating can be eliminated and a location management scheme consuming less resources and with smaller uncertainty risks can be designed.

- **Cost Optimization:** The best way to handle the uncertainty tradeoff problem when it is inherently generated from the two competitively leading factors is to optimize it. As the cost-based approach is the most natural choice for optimization, the uncertainty problem can be well addressed when the total cost is optimized.

- **Error tolerance:** Error tolerance is a desirable feature when the response time is also an important resource for query issuers. Under certain circumstances, query

issuers need replies from the service system as soon as possible without further probing into the most updated object location information, which is much more time consuming. When the error tolerance feature is provided, this situation can be well handled without sacrificing the precision of the query result.

1.4 Contribution

In this thesis, previous work in the field of location management for moving objects has been analyzed. The limitations of previous work have been removed by integrating the desirable features into the design of the efficient location management models. As a result, three models which are equipped with the features have been proposed. The proposed models are based on the client-server location management model which is the main focus in this thesis.

1.4.1 Query-aware Model

The target of the query-aware model is to take querying information into account in the model design. As query processing requires the location tracking in many practical applications, the query-aware location management model is designed to address the limitations of previous work which ignores the mutual impact between updating and querying.

1.4.2 Cost-based Model

Although the query-aware model improves the updating and querying performance for the system as a whole, the main problem (i.e., the uncertainty tradeoff) for location management still remains unsolved. To handle this limitation and maintain the benefits of the query-aware approach, a cost-based model is proposed.

Apparently, the best way to handle tradeoff is to optimize it and through the opti-

mization procedure, it is the natural choice to quantify tradeoff in the system as several kinds of costs due to updating and querying activities. In the cost-based model, the CUP (Cost for Updating and query Processing) scheme is proposed. The CUP scheme provides a target cost function and several adaptive optimization algorithms to achieve the minimal cost point. These algorithms can adapt objects management activities according to not only the changing movement pattern of moving objects but also the changing querying situation in the system.

1.4.3 Cost-based with Error Tolerance Model

To make the CUP scheme more general, further extensions to the basic version is proposed and the extended scheme has the ability to handle the error tolerance required by the system user. The error tolerance ability is useful when the response time of a query is also one of the important performance metrics the users take into account. The extended cost-based model provides mechanism to allow the user to control the errors that may be involved in the query results.

1.5 Organization of the Thesis

In this chapter, the background of research studies in a moving object environment has been briefly described. The limitations in previous work have also been analyzed and the motivations as well as contributions of this thesis have been discussed.

To have a clear view of the whole thesis, the organization of the rest of this thesis is outlined as follows:

- Chapter 2 reviews important work related to the location management for moving objects. A brief introduction to the research work of several general data management issues in mobile computing environments is given. Previous work of the location management problem in both cellular networks and moving ob-

ject environments is reviewed. The focus is on the fields of updating and querying problems as well as uncertainty problem handling.

- Chapter 3 introduces two common general location management models and their special characteristics. Then various location updating and query processing models are presented in detail. Discussions about desirable features which have not been applied in previous location management schemes and motivate the proposed techniques are given.

- Chapter 4 introduces the query-aware location management model. It gives background scenarios and examples which motivate the design of the query-aware model and shows how the query-aware model can help to improve system performance followed by introduction of query-aware updating and querying models. Based on the models, the proposed scheme *Aqua* is presented in detail, with simulation studies showing some possible benefits and performance improvements.

- Chapter 5 proposes a basic cost-based model. It first presents the overview of the optimization approach. Then the detailed updating and querying models are addressed. Based on the models, the basic cost-based scheme, CUP (Cost for Updating and query Processing) is presented, followed by an introduction of several adaptive optimization algorithms. Adaptive algorithms are designed based on the CUP scheme, as efficient tools for handling the dynamic environment and achieving the cost optimization. Simulation studies are conducted to examine the system performance. At last, limitations of CUP are discussed and possible extensions are explored.

- Chapter 6 extends the basic cost-based model. To simplify the definition of the cost functions, assumptions are made in the basic model. The relaxation to the assumptions and extension to the basic CUP scheme are conducted. As a result

of the relaxation and extension, the cost functions and the adaptive optimization algorithms are reviewed. Simulation studies are also conducted to evaluate the performance of the new algorithms.

- Chapter 7 concludes all the work in this thesis and outlines some potential future work.

Chapter 2

Related Works

In this chapter, previous work related to location management techniques in mobile environments is presented. First of all, the research on several general data management issues is briefly introduced. Previous work of the location management in both cellular networks and moving object environments is then presented. Among several issuers included in the location management problem for moving objects, updating and querying problems are the main focus in this thesis. These two fields of studies are reviewed in detail after the brief introduction. Finally, the research work about handling the relationship between updating and querying is presented.

2.1 Data Management

The data management in mobile computing environments [27] has been an active research topic in the mobile computing area for the last decade. There are many discussions about the new challenges met in this new situation and new research problems that need innovative solutions. The research survey [5] makes an excellent analysis on the impact on the mobile computing to the data management. According to it, the new environment is distinguished from the conventional situation in the following aspects:

- Asymmetry in the communication;
- Frequent disconnection;
- Power limitation;
- Small screen size.

All these characteristics pose a significant impact on the design and implementation of the databases for mobile computers and also create new problems and opportunities for the research world. Generally speaking, the new research problems faced by researchers include the management of location-dependent data, information services to the mobile users, wireless data broadcasting, disconnection management and energy efficient data access [28, 29]. The challenges include prototyping, transactional properties, bandwidth utilization, optimization of location-dependent query processing, data visualization etc [5]. One of the mentioned problems that attract many research efforts recently is the management issue for location-dependent data. Since the locations of the users change as they move, the issue of mobility is a natural and unavoidable problem that should be faced in mobile computing. The research work of the location management is reviewed in the following sections.

2.2 Location Management in Mobile Environments

2.2.1 Location Management in Cellular Networks

According to [45], in cellular network environments, the location-dependent data management and information access face several challenges, including mobile environment constraints, spatial data processing and user movements. New research issues are raised [45] when dealing with the location-dependent data by using the traditional three information access methods, namely, on-demand access, broadcast, and data caching.

Besides information access problems, the researchers [61] present a comprehensive survey of various approaches to other related problems in the field of location management. These problems include storing, querying, and updating the location of objects in mobile computing environments. Concerning the storing issues, four types of architectures of location databases are introduced, including a two-tier scheme, a hierarchical scheme, a non-tree hierarchy and a centralized DBMS (Database Management System). For different location database architectures, the strategies of caching, replication and forwarding pointers vary significantly.

The updating and querying issues are also discussed and reviewed [61], but a more detailed survey focusing on the location management in PCN (Personal Communication Network) can be found [90]. The location management problem is divided into the design of two basic operations [90], namely lookup/paging and update. A lookup/paging operation is invoked each time when there is a need to locate a mobile object. Updates of the stored location of a mobile object are initiated when the object moves to a new network location. Basically, these two operations are considered and conducted separately. According to most research work in this field [28, 61, 90], the fundamental tradeoff in the location management problem is between searching and informing. Some questions may be raised. Examples are if one wants to establish the location of a moving object o, should the search be conducted in the whole network or constrained around some predefined locations? Should o inform anybody about its moves? Basically, the more the updating cost is, the less the paging/searching cost should be paid. Various selective paging strategies and updating methods are discussed [61, 90].

The research issue of how to accurately maintain the current location of a large number of mobile objects while minimizing the number of updates is not trivial given that the location of a moving object changes continuously but the database cannot be updated continuously. Basically, three strategies are proposed to address the update frequency problem in PCN [4], namely, the time-based strategy, the movement-based

strategy and the distance-based strategy. In the time-based update strategy [66], a predefined time period is set. Each moving object performs location updates periodically according to this predefined time period, say every T units of time. In the distance-based location update [50], a location update is performed when a mobile object moves to a location that the distance between the last registered cell and the current location exceeds a predefined threshold. The movement-based strategy [2] asks a moving object to update its location after it has performed a predefined number of movements. Among these three strategies, the distance-based location update is shown to be the best according to the analytical performance results [2, 4, 67].

All the three basic location update strategies belong to the group of static update algorithms. In a static algorithm, the location update is triggered based on the topology of the network. The other group of algorithms is dynamic strategies whose location update is based on the user's call and mobility patterns. Examples include the selective LA update and profile-based location update scheme. LA stands for location area. Selective LA update [72] allows mobile object not to perform location updates in every LA. The update LA is selected based on the user's movements and the time periods it stays in that particular cell. The profile-based [61, 62] location update scheme is also designed to take advantages of user's movement patterns and all the information is maintained by the network as a profile for each user. Variations of the three basic location update strategies are proposed and most of them belong to the dynamic group of update algorithms. Examples can be found in [38, 50, 53]. All the variation schemes make adaptive decisions to the threshold value which is predefined in the static schemes. The adaptation is mainly done to make the update behavior adjustable to the mobility patterns, system model and call patterns.

To compare the performances of various update schemes, much research makes assumptions of certain topology and mobility models for the mobile environment. Modeling techniques are then applied for the performance analysis of location updates and terminal paging. The analysis work based on several common modeling can be found

in [2, 3, 66, 67, 98].

2.2.2 Location Management Issues for Moving Objects

The location management problem for moving objects environments involves several interrelated components listed below.

- **Positioning Technology** acquires the position and movement information of moving objects.
- **Location Modeling** addresses the problem of how to construct the position information of each moving object acquired via position technology into useful and retrievable location information.
- **Storing Problem** concerns the architecture of location databases.
- **Location Updating Technique** addresses the problem of how to continuously maintain in a database the current locations of a large amount of moving objects.
- **Query Processing Technique** addresses the problem of answering queries issued from the system user and returning the results back.

2.2.2.1 Positioning Technology

There are three main technologies to determine a given location: triangulation, proximity and scene analysis. The implementations of locating system generally use one or more of these techniques to locate any moving object. A survey of these location systems is provided in [23]. Examples include the global positioning system (GPS), Active Badges, Active Bats MotionStar, MSR RADAR, Cricket and so on. These systems differ in location sensing technologies used, accuracy possessed, sensing scale reached. Among all the location systems, GPS is by far the dominant one, and it is getting better and cheaper [26, 12].

2.2.2.2 Location Modeling

The database may implement different location models for varied processing needs. Compared to location modeling in the cellular architecture which uses network locations, moving objects modeling has higher resolutions.

Point and trajectory modeling are two commonly used modeling techniques in the modeling component. Compared to point modeling which is the most straightforward method and has several limitations, the novel trajectory model for the moving object database has more beneficial features [85]. The management system receives samples of the position of each moving object, which enable them to construct a trajectory for each object that represents the object's movement. Trajectories are also termed poly-lines and consist of connected line segments.

However, manipulating and querying the trajectory representations of movements in space and time is inherently challenging because the amount of collected data is proportional to the elapsed time. To conquer the challenge, moving objects have been modeled as abstract spatio-temporal data types in the context of spatio-temporal databases. Spatial-temporal databases [51, 71, 60, 65] deal with spatial objects whose positions and regions change over time. A data model [17, 21, 18] for moving object databases has been proposed in the same context. The data model includes evolving spatial structures and is implemented as a collection of data types and operations.

A new data model MOST is proposed [73, 74]. MOST models locations of moving objects as one of the *dynamic attributes* whose values can change continuously with time passing. The location information stored in the MOST model has some potential uncertainty and deviates from the actual locations of moving objects. In [84] and [86], the MOST model is extended to deal with situations where moving objects move on pre-specified routes.

Trajectory modeling can also be improved by modeling moving object trajectories as piecewise linear functions of time, and process these less frequently changing

functions instead of more frequently changing object positions. This kind of representations of movements is very useful for efficiently indexing the moving object positions [37, 69, 88].

2.2.2.3 Storing Problem

The centralized architecture is commonly assumed in moving object environments [61]. The storing problem concerns how to represent and index the positions of moving objects in a database management system and how to efficiently retrieve them for query processing.

Early work in spatial databases assumed a static dataset and focused on efficient access methods and query evaluation algorithms. The most famous example is R-tree [22]. The related query evaluation algorithms are introduced in [24, 68, 47]. In order to represent and index the position data of moving objects, the most straightforward approach is to extend the commonly accepted spatial index (e.g. R-tree [22]) with desirable novel features.

There are two ways to extend R-tree. Historical R-tree (HR-tree) [54, 6, 70], multi-version R-tree [79, 39] and 3D R-tree [83] introduce timestamps for R-trees and are designed to index the historical moving object location data. In order to reduce the space consumption, HR-tree applies partially persistent structures to allow R-tree to share common nodes at consecutive timestamps [70]. As HR-tree still involves considerable data redundancy, multi-version R-tree is developed to further reduce the space needed for storing [79, 39]. The 3D R-tree was also invented for indexing historical data. It treats time as an extra dimension [83].

Quadtree [80] and B^+-Tree [32, 34] indexing methods are extended to index predictive location data. They generate periodically the index to support queries about the future. When assuming object movement trajectories are known, Time-parameterized R-tree (TPR-tree) for indexing moving objects is proposed [69]. In a TPR-tree, the location of the moving object is represented as a linear function of time. TPR^*-tree

is an extended version of the TPR-tree and optimizes the performance of the TPR-tree [76]. Corresponding query processing algorithms based on the TPR-tree are introduced later. Examples are query evaluation algorithms for NN and reverse NN search developed in [7].

Alternative indexing methods besides the TPR-tree for predictive spatio-temporal querying include the grid model and dual transform. In [13], the authors invent a grid model for indexing moving objects and make use of the index for range and kNN queries. STRIPES is introduced in [59] which is a novel index structure using the dual transform.

Besides positioning technology, location modeling and the storing problem, the other two components of the location management problem for moving objects left, namely location updating technique and query processing technique are introduced in detail in the following two sections.

2.3 Location Updating Techniques

Conventionally, moving objects equipped with some positioning tool (e.g. GPS) propagate their location updates to a location server where the queries are handled. To keep the precision of query results, frequent updates are needed given that the location of a moving object changes continuously [4]. To do location updating more efficiently and maintain fairly accurate location information of moving objects for the query purpose, a number of techniques have been proposed in previous research work.

2.3.1 Basic Updating Methods

The most straightforward way to do updating is to update the location information every time the object changes its position. This naive approach is surely neither feasible nor acceptable because it may generate excessive communication cost and can

quickly exhaust the energy of the moving objects. As the main problem of monitoring continuously moving objects is the cost constraint, in order to reduce the update costs, most recent research work rather than using the naive updating method prefers to use a linear function $f(t)$ for expressing the movements of objects. The linear function can be used to estimate the position of the objects at different times [73]. Therefore, the location information of moving object needs to be updated only when the parameters of the function change. This basic updating method avoids excessive location updates because no explicit update is required unless the parameters of $f(t)$ change [88].

2.3.2 Threshold Strategies and Safe Region Methods

In many real applications, however, it is hard to find a good function to describe object movements. When the simple linear function cannot describe a complex movement, a lot of update overheads are needed for the changing parameters. Threshold techniques are introduced to handle the complex situation. Threshold techniques have the advantage of preventing numerous updates and the disadvantage of making the location information data inaccurate.

The performance of the deviation-based policy with a predefined threshold setting has been studied in [89]. Their work assumes that the threshold setting in most practical systems is up to the choice of the user and targets at reducing the communication cost as much as possible while keeping the agreed level of the location uncertainty. The deviation-based policy is shown via simulation experiments to be up to 43% more efficient than the distance-based policy which is commonly used in terms of messaging cost. However, query precision issues were not addressed there. In [89], the threshold is set for linear deviation. Angular deviation can also be used as the threshold, for example in object tracking via the dead-reckoning policy [20]. This policy monitors both linear and angular deviation and issues a location update whenever any one deviation exceeds their predefined threshold. Objects in this model are assumed to travel

on predefined routes; no further uncertainty control is mentioned.

Other than threshold techniques, the safe region method was also proposed to address the updating problem. In order to reduce the location update cost, an object that remains within its prescribed boundary can be included in an answer, satisfying a given error bound. This prescribed boundary is called *safe region* in which the object is assumed to stay without any location update.

In [1], moving objects are bounded by the safe region within which updates to the database can be eliminated. The region can be adjusted. The adjustment is made according to the movement patterns of the objects. Different methods to define the safe region areas are also proposed and studied [46].

2.3.3 Movement Prediction and Group-based Updating

The technique of movement prediction which provides the estimation of the location information can be applied to reduce the number of updates [14, 20, 92]. This leads to a better system performance. Three update policies were proposed: a point policy, a vector policy, and a segment-based policy. Different prediction approaches are applied in these three policies and the segment-based policy is regarded as the most important because it enables a wider range of services.

Moving objects can also be clustered into groups so that the group leader can send location updates on behalf of the whole group [9], thereby reducing the expensive uplink updates from moving objects to the system location server. In GBL [40], moving objects are clustered into groups dynamically, in which group leaders report the object locations in a collective manner to the system server, thereby reducing the expensive uplink reporting traffic. Group maintenance procedures are defined to keep the groups alive.

2.3.4 Other Approaches

There is some research work addressing the update problem in some specific applications. For example, the researchers [82] address the problem of updating moving objects databases using the real-time traffic information and researchers [48, 15] address the updating protocols when the road network scenario is examined. Lam et al. [41] propose an adaptive monitoring method for location-dependent continuous queries. Their approach also belongs to the dead-reckoning method. To achieve the target of increasing the correctness of the query results, their work sets a smaller update threshold for those objects that fall into a query region and assigns larger threshold values to those who are outside the query region.

A range of other special techniques are proposed for further performance improvement. Examples include representing object positions as more complicated and comprehensive functions, taking movement constraints into the consideration of the movement model, predicting the future movement based on application semantics [33].

A comparison work of updating methods is made in [49]. The paper classifies the updating methods according to the update issuers and three main protocols are identified.

2.4 Query Processing Technique

The query response is the essential system performance metric for location-dependent services in mobile environments and query processing is a crucial component in the location management problem. In this section, the literature review on the relationship between indexing and query processing is conducted. Previous work that processes the continuous query using efficient techniques is also reviewed. Finally, distributed approaches for query processing are introduced.

2.4.1 Index Technique for Query Processing

Due to the dynamic nature of moving object environments, the volumes of the data stored are necessarily large and the data must be assumed to be disk resident. To obtain efficient query performances, some forms of indexing should be employed. As a matter of fact, a lot of previous research work on moving object querying issues makes their focus on index designing and related algorithm improvements. These indexing techniques include introducing timestamps for R-tree to index the historical moving object location data (e.g. HR-tree [54, 6, 70], multi-version R-tree [79, 39] and 3D R-tree [83]), extending R-tree to index predictive location data (e.g. Quadtree [80], B^+-Tree [32, 34] and TPR-tree [69]), indexing location information with novel structures (e.g. grid [13] and STRIPES [59]). These indexing methods have been reviewed in the storing problem. Note that all the work which designs the index structure introduces corresponding algorithms for efficient query processing.

2.4.2 Continuous Query Processing

Unlike ad-hoc queries, the continuous query aims to continuously tracking the changes of the results until certain conditions are fulfilled. This type of queries is more challenging as well as useful in many application scenarios. This section reviews the current efficient techniques proposed for processing continuous queries.

2.4.2.1 Validity Region, Grid and Linear Modeling

The concept of validity region method is applied in [93, 97]. A validity region is a boundary which is around the query issuer's location within which the result remains the same. The technique returns to a moving query issuer the current result as well as its validity region where the result remains the same. The query is reevaluated only when the query issuer exits the validity region.

Grid-based in-memory structures for object and query indexes can also be used to

speed up the reevaluation process of range and kNN query. Previous research work [35, 96] implements the structure and propose corresponding query processing algorithms.

Assuming a linear movement of objects, algorithms of kNN queries are examined in [30, 64]. The extended version of [30] proposes algorithms for the distance semi-joins for two linearly moving datasets for moving objects [31].

2.4.2.2 Scalable Algorithms and Generic Scheme

An alternative approach for continuous query processing is query indexing. A R-tree like structure called Q-index [63] is used to index queries. At each evaluation step, only those objects that have moved since the previous evaluation step are reevaluated against the Q-index. This novel technique targets at efficient and scalable processing of continuous queries.

Another scalable algorithm called SINA is also proposed in [52]. This is an incremental hash-based algorithm for both range and kNN queries. SINA indexes both queries and objects and achieves scalability by employing shared execution and incremental evaluation of continuous queries. Similarly, the authors of [91] apply the incremental evaluation and shared execution to achieve scalable query processing. Their work focuses on processing continuous k-nearest neighbor queries only.

A generic framework [25] is introduced to deal with most types of continuous queries for moving objects. The distinguishing aspect of this framework is that it can adapt query patterns to reduce the traffic overheads introduced by continuous object location changing.

2.4.2.3 Dynamic Queries Processing

Unlike conventional queries over moving objects, dynamic queries are continuously changing. For example, the querying region is moving over time. Algorithms to pro-

cess moving queries over stationary objects are proposed in [78, 77]. The situation for both dynamic queries and objects is more challenging and addressed in [44].

2.4.3 Distributed Approaches

The approaches introduced above ignore the underlying mobile communication system and the capabilities of moving objects. In contrast, the essence of the distributed approach is to make use of the computation capabilities of the moving objects and release part of or all of the query processing tasks to them.

Previous work towards this direction [8, 19] addresses the processing of continuous range queries by utilizing distributed location monitoring techniques. The Domain Tree [8] was invented to index static continuous range queries in the server. In the system, each moving object should find its own resident region according to the domain tree. While moving around its resident domain, the moving object keeps monitoring those queries whose ranges overlap that sub-domain and does reporting if it is within any query range. A distributed real-time location monitoring system, called MobiEyes is introduced in [19]. In the system, the server mainly acts as a mediator between moving objects. MobiEyes introduces a much more complex model for processing continuous moving queries over moving objects. Most design efforts have been directed to monitor locations of moving queries better and efficiently determine the set of queries that the moving objects should keep on evaluating.

2.5 Uncertainty Problem in Location Management

Uncertainty is the main problem that should be appropriately handled in a location management system. One way to handle the uncertainty is to propose probabilistic answers to queries by accepting the existing of the uncertainty. This way quantifies the uncertainty. The other common practice is to make efforts to control the uncertainty and balance the tradeoff between the resource consumption and the uncertainty

penalty. The previous research work to handle the uncertainty problem is reviewed in this section.

2.5.1 Qualitative Solution and Quantifying Uncertainty with Probability

In the context of moving object environments, the inherent uncertainty property in the stored object location information has attracted extensive research efforts. The first research work addressing this issue is DOMINO [87], in which both qualitative and quantitative solutions are proposed [73, 88]. The May and Must semantic are introduced in the qualitative approach. The May semantic provide an answer to the query issuers so that all objects that have the chance to be within the answer set will be returned as a result. While under the MUST semantic, the answer is only those objects that are surely in the results. Unlike the qualitative approach, the quantitative solution provides concrete measures to the uncertainty. The answer not only includes the possible objects, but also the probability that these objects may be part of the real answer to that query.

The research work from [11, 10] also quantifies the uncertainty for the location management problem. The fundamental argument is that answers of location-dependent queries can be augmented with probabilistic estimates of the validity of the answer. Unlike other works which trade the accuracy for the performance or vice versa, their work designs algorithms for providing probabilistic answers for several query types. Specifically, the execution of the probabilistic range and NN queries is studied.

2.5.2 Balancing Tradeoff Introduced By Uncertainty

The preliminary work to handle the inherent tradeoff problem introduced by the uncertainty is to bound the uncertainty with a signed agreement between the service user and provider.

Two update policies are proposed [84], namely they are immediate linear and delayed linear update policies. In both policies, the location uncertainty is measured by the deviation which is estimated by a linear function of time. Both policies are deviation-based and apply the MOST data model for moving objects. MOST is a data model proposed in [73] and models the location of moving objects as one of the dynamic attributes whose values can change continuously as time passes. Both update policies show that it is possible to track the moving object with desired uncertainty bound, and it is also possible to find the optimal time for the updating behavior. However, the uncertainty bound is assumed to be a reasonable performance level accepted by the service user. Queries are accordingly processed based on the uncertainty bound. As the uncertainty bound agreement is created beforehand, the user is expected to be satisfied with the service provided.

An alternative approach to handle the tradeoff problem is a cost-based method whose aim is to optimize the quantified tradeoff itself and achieve the optimal uncertainty bound. This approach is designed to conquer the limitations of the previous work.

There are three dead reckoning update policies which can perfectly make the bounded uncertainty error known to the system and query issuers [86], including the plain dead-reckoning (pdr) (i.e. the threshold value is predefined and the same for every update), the adaptive dead-reckoning (adr) (i.e. an extension of pdr by computing a new threshold with each update) and the disconnected detection dead-reckoning (dtdr) (i.e. further extended the pdr to deal with the problem of network disconnection). The design goal of dead-reckoning approaches is to balance the tradeoff between the performance and tracking cost and find the cost-optimal updating point. These policies still make use of a deviation threshold as in the previous linear methods. They propose an information cost model that captures the uncertainty, deviation and communication to determine when the location of a moving object in the database should be updated. All of the three dead-reckoning policies can find a deviation threshold which is the

maximum distance the moving object can deviate from its database recorded location by optimizing the information cost. In the dead-reckoning policies, a cost function should be pre-defined for the whole system. Based on the cost function, a cost-optimal update can occur and thus the optimal threshold value can be computed to yield a minimum information cost for every single update. By this means, it is said that the uncertainty is controlled and the optimization is achieved. The performance of those dead-reckoning policies is studied and experiments on the simulated data are also conducted to compare the three dead-reckoning policies as well as the delayed linear and immediate linear policies. The results show that the dead-reckoning policies are superior to the two linear ones.

2.5.3 Tradeoff Handling in Other Research Fields

It has been found that the tradeoff problem exists in many other research contexts. The methodologies and strategies introduced in these contexts may also be applicable to deal with the particular problem met in the moving object environment. Therefore, a literature review of the tradeoff problem in other contexts has been conducted.

For spatial queries processing, Xingbo Yu et al. conduct the tradeoff study between the precision and performance [95]. Rather than quantifying the tradeoff, they study the correlation between the data quality and precision requirements given in the queries issued by users. They assume that the accuracy/uncertainty of the query result has already been required by the user and what the system is expected to perform is to fulfill the user's specific certainty requirement. Under this situation, they present a novel technique to set the data precision constraints for the data collecting process so that guarantees on the uncertainty in answers to the queries could be provided. However, their work addresses aggregated queries only.

Another work related to the tradeoff handling is from Chris Olston and Jennifer Widom [58, 56]. Their work is a study in the environment of "stale replication". In the

so called TRAPP (Tradeoff in Replication Precision and Performance) system, they aim at balancing the precision and computation speed (performance) by combining cached and master data. The key methodology in their work is that once the user's requirement on data "refreshment" is confirmed, the system then can automatically select the sources to get back the data needed. The sources include cache and master data. Retrieving data from the cache can speed up the computation while from the master data can meet the tight freshness requirement. Their work focuses on queries with aggregation.

Similar to [58], the EASE scheme from Jianliang Xu et. al. [94] also addresses the tradeoff between precision and performance in the research field of tracking sensor networks. Their approach trades the precision for the energy efficiency and tries to derive the optimal approximation setting for the imprecise data when the mobility pattern is known.

All the work dealing with the tradeoff problem introduced above provides some hints to us when we look for a better strategy to handle the tradeoff problem in the moving object context.

Chapter 3

Location Management Model for Moving Objects

Mobile computing has become a reality as a result of the appearance of powerful portable computers and the development of fast reliable wireless networks. In this new computing paradigm, objects of interest (e.g. humans, cars) would possess unrestricted mobility and portability and thus are referred to as *moving objects*. The novel characteristics and abilities of moving objects within the mobile computing environment have enabled the provision of LDIS applications (Location-Dependent Information Services). In this context, the location management plays a central role for supporting efficient LDIS services.

A good location management scheme should have the ability to provide efficient query processing and location updating. Updates occur when a moving object changes its location and queries are generated when the system user wants to receive some LDIS services which involve certain moving objects whose locations are unknown to the system user. A general location management model can be implemented as different location management schemes which apply different updating and querying models.

In this chapter, two common general location management models and their spe-

cial characteristics are introduced. Following the introduction of the general model, various location updating and query processing models are then presented in detail. At the end of this chapter, some features which have not been applied in previous location management schemes are discussed. These features actually motivate the novel techniques proposed in this thesis.

3.1 Common Location Management Models

In mobile computing environments, there are two different data management approaches, namely, the centralized and distributed approach. In the centralized approach, some server computers are dedicated to serve the others. In the distributed approach, every node is treated equally. Location management activities in these two types of approaches operate differently. In this section, two common types of models are introduced. The *client-server model* implements the centralized approach and the *peer-to-peer model* follows the distributed approach.

In this thesis, discussions are focused on the client-server location management model. Compared to the peer-to-peer location management model, the client-server model is more commonly applied for realistic applications. Therefore, it is the main focus of this thesis. Both models are introduced in this section and the discussion of all following sections and chapters are based on the client-server model.

3.1.1 Client-server Location Management Model

The most common approach for the location management in a mobile environment is based on the client-server architecture in which there is a location server managing locations of moving objects and serving system users, who are regarded as clients. Generally, the location server is equipped with a centralized DBMS and communicates with some query processors whose function is to provide location querying service to system users.

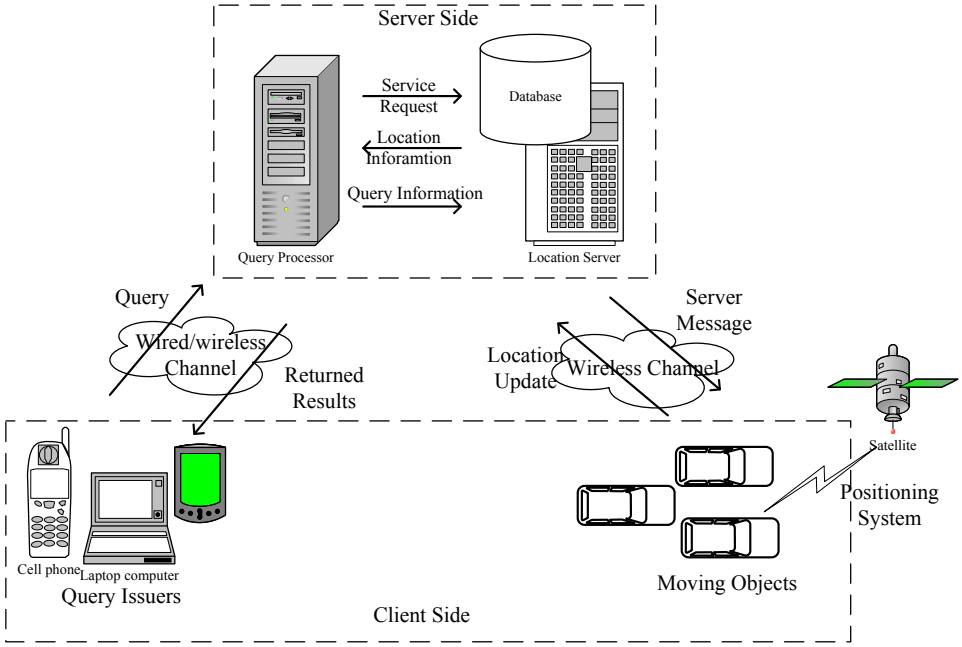


Figure 3.1: Client-server Location Management Model

Figure 3.1 depicts the client-server location management model that supports the location management of moving objects and the corresponding query processing. There are two main components in this model, namely, the server-side component and the client-side component. Between these two components is a comparatively low bandwidth wireless network.

3.1.1.1 Server-side Component

There are two main entities residing at the server-side, including a *location server* and a *query processor*. The server-side component is the service center whose main function is to provide location-dependent services to system users.

Location server

In the client-server location management model, moving object locations are managed by a location server. The location server is normally equipped with some type of DBMS such as Oracle, DB2. The research in this area often makes use of the notion of MOD (i.e., Moving Object Database) which has extended capabilities. The data stored

in MOD includes the updated location records of moving objects which can be queried by the system user. Management tasks running in the location server include keeping location records up-to-date, providing location query services, improving the service efficiency by making use of an appropriate data index etc. There are two types of communication channels between the location server and other system components. One type of the communication channels connects the server with client-side components via a low bandwidth network. The information flowing through this channel includes location updating messages submitted by the moving object to the server and probing messages from the server to the moving object. The other communication channel connects together different server-side components.

Query processor

The main function of the query processor is to process queries issued from the system user. It communicates with the query issuers through a wireless or wired channel. To process location-dependent queries, the query processor needs to cooperate with the location server which provides the location information service for query answering.

3.1.1.2 Client-side Component

The components that reside at the client-side are *query issuers* which include the system users and *moving objects*. The moving objects are monitoring targets of the system. These two types of entities sometimes can be identical, implying that some moving objects whose locations are of interest to others also issue location-dependent queries to the system.

Query issuers

Query issuers are system users who receive location-dependent services provided by the system server. They communicate with the server-side component via a wired or wireless channel. The query issuer can be any entity or device which has the ability to send and receive messages. The query issuer sends location-dependent queries using

uplink channels to the server-side component and gets back the results. They can also employ some mechanisms to filter the query results or resend follow-up queries to obtain better results to fit their particular requirements.

Moving objects

Moving objects are the targets queried by the system. Their properties especially the location information in the location-dependent information system are of special interests to system users. Examples of moving objects include public buses, taxis, automobiles, cellular phone users, air planes etc. These moving objects possess several common features as listed below:

- They change their positions continuously with time;
- They are equipped with a positioning tool such as a GPS receiver and have the ability to locate themselves in a mobile environment;
- They have the ability to propagate their location updates through a wireless channel to the location server;
- They have limited resources such as short battery life and therefore need to run with resource-preserved applications.

3.1.1.3 Cooperation between Client and Server

To efficiently manage the locations of moving objects, cooperations between the client-side component and the server-side component and among entities in each side are essential. Client- and server-side operations and the cooperation between them are summarized in Table 3.1 and Table 3.2.

3.1.2 Peer-to-Peer Location Management Model

The peer-to-peer location management model is the proper solution when no centralized server is available. Entities with similar capabilities play equal roles and exchange

Query issuer	Moving object
1: Send location-dependent queries to the query processor	1: Collect its own location and current time via its positioning device
2: Receive query results from the query processor	2: Communicate with the Location Server based on its updating protocol
	3: Update its location voluntarily
	4: Send feedback to the server upon server's request

Table 3.1: Client-side Component

Query processor	Location server
1: Receive query message from users	1: Provide location information to the query processor
2: Request location service to the location server	2: Update location records by receiving moving objects updates
3: Feedback query information to the server	3: Send messages to moving objects when needed
	4: Get useful query information from the query processor

Table 3.2: Server-side Component

information with and provide services to one another. This kind of architecture differs from the client-server model, in which the server-side component is dedicated to serve the client-side component [75, 16, 36].

Figure 3.2 depicts the peer-to-peer location management model. In this model, moving objects continuously change their physical location and establish peer relationships among one another. In order to provide end-to-end communication throughout the network, moving objects must cooperate to support general networking functions such as ad hoc message routing. There are some features of peer-to-peer systems.

- **Self-organizing**: Whenever a moving object moves, it re-discovers the set of reachable moving objects. It sends "ping" messages around and listens for the corresponding "pong" messages.

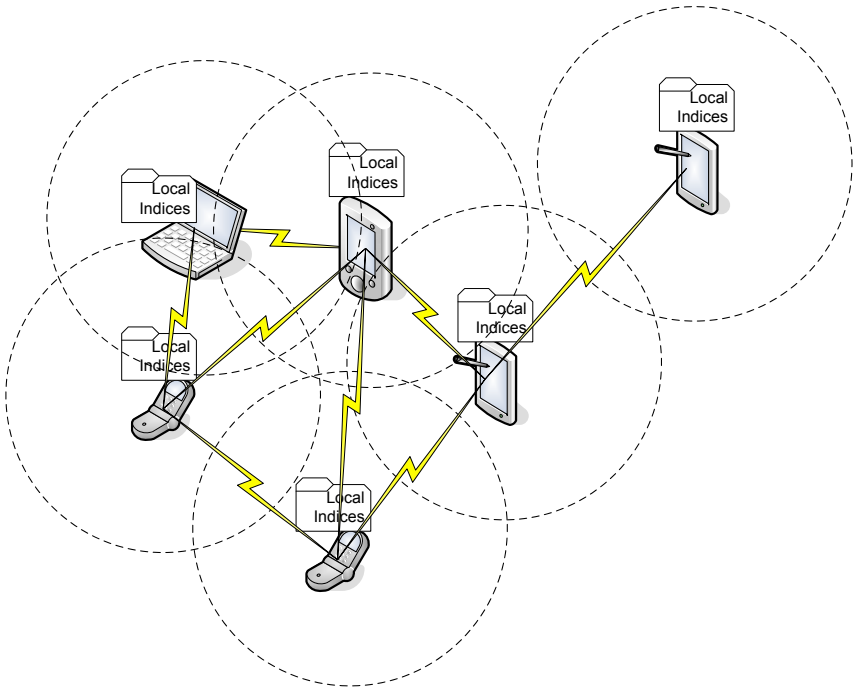


Figure 3.2: Peer-to-peer Location Management Model

- **Fully decentralized**: No central server exists in a peer-to-peer environment. Therefore, every moving object is equally important within the network.
- **Highly dynamic**: The topology of mobile peer-to-peer systems can change very rapidly.

Similar to that in the centralized approach, the location management problem in the peer-to-peer model consists of two sub problems: how to efficiently perform location updates and how to efficiently answer location queries.

The absence of any centralized, dedicated server to maintain the location information of the moving objects poses a big challenge to the peer-to-peer model. The **Local Indices** technique is often adopted for location information updating and querying. The local index of a moving object is a data structure that records the location information of all its adjacent moving objects. Normally, a moving object records all the adjacent objects within r hops away from itself. The number r is known as the radius of the index. With the location indices, the following sections show that how location

updating and querying can be handled in a peer-to-peer model.

3.1.2.1 Location Updating Activities

As the moving objects are moving constantly, the location indices which provide the location information about other moving objects should be updated at regular intervals and under certain scenarios.

Scenario 1: A moving object joins a network.

When a moving object joins a network, it scans its neighbors within a depth of r hops from it and forms the Local Indices by requesting for the neighbors' identities. During the process of formation of the Local Indices, each moving object that receives the scan will update its own Local Indices with a record of the moving object that recently joined the network.

Scenario 2: Normal updating work.

Periodically, every moving object sends a *ping* message to all mobile hosts in its Local Indices. In response, every moving object that receives a *ping* message responds with a *pong* message, indicating its existence.

Scenario 3: A moving object leaves a network.

The leaving object does not do anything to indicate its departure. Other moving objects will find out the fact when no *pong* message is received from this departed moving object after a fixed interval of time. Then these objects assume that the object is no longer within their radius or is currently disconnected. A corresponding deletion to the Local Indices of these adjacent moving objects is made.

3.1.2.2 Query Processing Activities

When an object receives a location-dependent query, it checks the Local Indices to determine if it has the location information of the queried objects. If it is the case, the object would return the information to the querying object; otherwise, the Local

Indices does not contain any useful record and the object would forward the query to other objects within r hops away from it. By using the Location Indices, the query issuer can query a small number of moving objects and obtain the location information of many moving objects. The procedure of maintaining moving objects in the peer-to-peer location management model is summarized in Table 3.3.

Updating	**Querying**
1: Scan the neighborhood when joining a new network	1: Receive a query from neighbor n_i
2: Build up the local indices with a depth of r	2: Search the local indices
3: Periodically *ping* neighbors	3: If location information is recorded then
4: Update the location indices	4: Return results to n_i
	5: Else pass the query to other neighbors

Table 3.3: Moving Object Activities in Peer-to-peer Model

3.2 Location Updating Model

The location management for moving objects involves two basic operations: *update* and *query*. The location server is equipped with a database maintaining the reported object locations for the purpose of query processing. With stored object locations, the queries can be promptly answered. However, some stored locations could deviate significantly from the actual object location and some do not. In the client-server location management model, we generally witness a large number of user-interested objects moving continuously in the system. Equipped with some positioning tools (e.g. GPS), moving objects have the ability to determine their locations. The updating activity therefore occurs between the location server and every moving object.

An updating protocol should be executed at the location server as well as in the moving object as a mutual agreement between the server and the moving object. This agreement decides when and how often an update is to be sent from the moving object

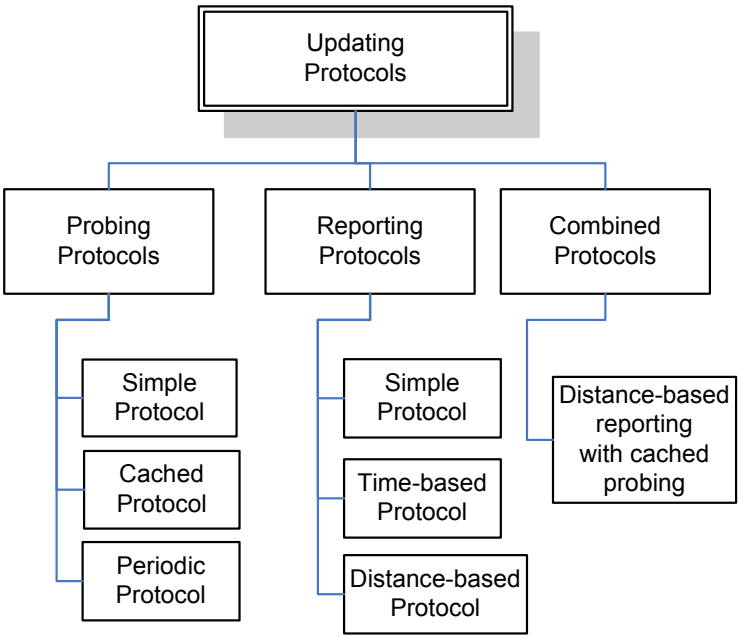


Figure 3.3: Updating Protocols

to the server. In most applications, the query processor requires the location server to provide the location information with ensured qualities. Therefore, various updating protocols are designed to fit various querying situations.

In this section, several updating protocols and their classification are introduced. Among these protocols, the combined distance-based protocol with cached probing possesses unique features and performs better when compared to other protocols. This kind protocol is presented in detail in the subsequent sections.

3.2.1 Updating Protocols

Generally speaking, three main classes of updating protocols are identified [49]. Figure 3.3 shows the classification. Based on the deciding part of the updating activity, updating protocols can fall into *probing protocols*, *reporting protocols* or *combined protocols*. Under these three main categories, a finer classification is described in detail below.

3.2.1.1 Probing Protocol

If the updating activity is decided by the location server, the updating protocol is called probing protocol. There are three types of this kind of protocols.

Simple

The updating activity in the simple protocol occurs when the server probes the moving objects for their current location information and the server probe occurs only when some queries are generated for the specific object's locations. This is called simple protocol and actually no stale location information is stored at the server-side. The obvious drawback is that a large number of updating messages should be caused if some particular objects are queried often. However, the advantage is that no stale location information is returned in the query result.

Cached

To relieve the drawback of the simple protocol, a cached probing protocol stores the last updated location information at the server and makes use of the information for answering queries rather than probing the moving objects every time the locations of these object are queried. To control the degree of imprecision of the results in query processing, the protocol estimates the accuracy of the stored location information in server. Whenever a query is issued, the server will return results based on the stored location information if it is estimated to be accurate enough; otherwise, a location probe from the server to the moving object is initiated.

Periodic

As its name suggests, a periodic probing protocol is one that the server probes the location information from moving objects every T time units. This protocol has the same characteristic with the time-based reporting protocol introduced in the next section.

3.2.1.2 Reporting Protocol

If the updating activity is decided by the moving objects, the updating protocol is called a reporting protocol. In reporting protocols, the server never bothers to probe for more accurate location information from moving objects and makes query processing relying totally on the stored record in the database. Three types of reporting protocols are summarized as follows.

Simple

A moving object sends an update report whenever its location has changed. Obviously, this kind of protocols can yield excessive traffic overheads for moving objects.

Time-based

A moving object sends an update report to the server every T time units. The update rate is fixed and the degree of the temporal uncertainty is controlled. The drawback of this protocol is that it cannot provide any spatial guarantee for the querying precision.

Distance-based

A moving object sends an update report to the server when the distance between its real position and the location stored in the database exceeds the pre-determined threshold. The distance-based protocol relieves the drawback of the time-based protocol. However, some spatial uncertainty is caused.

3.2.1.3 Combined Protocol

The drawback of the plain probing protocol is that it cannot adapt to different mobility patterns of moving objects. The drawback of the plain reporting protocol is that it does not consider the query factors in terms of the arrival rate and precision requirements. A combined protocol aims to integrate the advantages of both the probing protocol and the reporting protocol and makes the updating activity to be both query-aware and movement-aware.

One example of the combined protocol is the distance-based updating protocol with

cached probing. This protocol is an integration of cached probing and distance-based reporting. In this protocol, the server can implement movement prediction mechanisms and the moving objects compute the distance between their real position and recorded position in the location server. When the distance exceeds the predefined threshold, an update report needs to be sent to the server from the moving object. Queries are issued and processed at the server. Unlike the plain distance-based protocol where query results can only be generated based on the stored location information, the server is able to probe more accurate location information if the query precision requirements cannot be fulfilled by the stored location records.

3.2.2 Distance-based Updating Protocols with Cached Probing

In this thesis, discussions are focused on the combined distance-based updating protocol. Compared to other updating protocols introduced earlier, this kind protocol possesses several distinct advantages.

- The combined protocols relieve the drawbacks of both plain probing and reporting protocols.
- Distance-based updating protocols provide certain spatial guarantees to queries. This is especially beneficial for location-dependent queries.
- Cached probing protocols make updating protocols more flexible under a variety of querying requirements.

There are three variants of combined distance-based location update protocols, namely the *basic protocol*, the *prediction protocol* and the *safe region protocol*. These three protocols are detailed in the following sections.

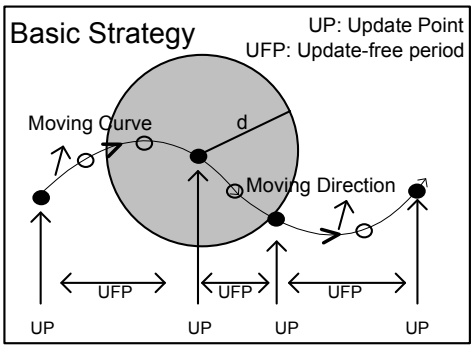


Figure 3.4: Basic Distance-based Protocol

3.2.2.1 Basic Protocol

The main idea of the basic protocol is that the server simply keeps the latest reported object location. As long as the distance that an object is located away from the reported location does not exceed the threshold, no update will be generated from the client-side. Server-side probing occurs based on the query requirement as in the normal case of the cached probing protocol.

Figure 3.4 illustrates the reporting activity of this protocol. The figure considers an object moving along the directed line in the space. Between two update points (UP), there is an update-free period (UFP) during which no update is needed and the object keeps monitoring its own locations. The gray circle with radius d represents how far the object can move without updates.

3.2.2.2 Prediction Protocol

The prediction protocol optimizes the basic protocol by applying movement prediction techniques. Similar to the basic one, moving objects update their location position whenever the distance between the real position and the record that stored in the database exceeds some threshold d. However, the database record in the prediction protocol dose not contain the last updated location information; rather, it is an estimated value which is computed by both the server and the object itself. The

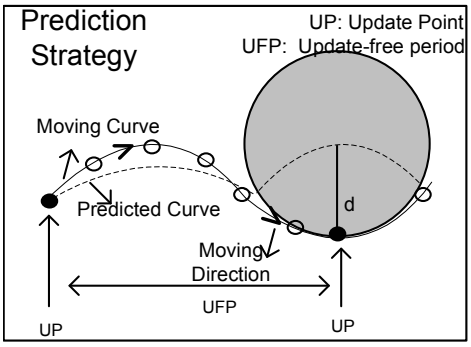


Figure 3.5: Prediction Distance-based Protocol

computation is based on the object's old location, its speed and the direction of its movement. The dead-reckoning strategy performs well when the prediction value is relatively precise.

Normally, the server in this protocol captures the movements of objects as functions of time. Given a time and a function, the location of an object can be calculated. This function is known by both the server and the respective object. The object always compares its current location and the calculated position based on the function during its update-free period (UFP). If the difference is more than a threshold, a report is generated to the server at the update point (UP). Figure 3.5 shows an example of the prediction protocol. The dotted line represents the predicted object movement. The gray circle represents how far the current location of the object can deviate from the predicted one. If the real curve is within the circle, no update is needed. Otherwise, an update is sent to the server.

3.2.2.3 Safe Region Protocol

The safe region protocol uses a region to represent a set of possible locations of a moving object. The region is called **safe region** of this object which means that the object can move safely without bothering to update its location within the region. The choice of the shape and size of the safe region should be mutually agreed between the location server and the object beforehand. It could be a circle, a rectangle, or a

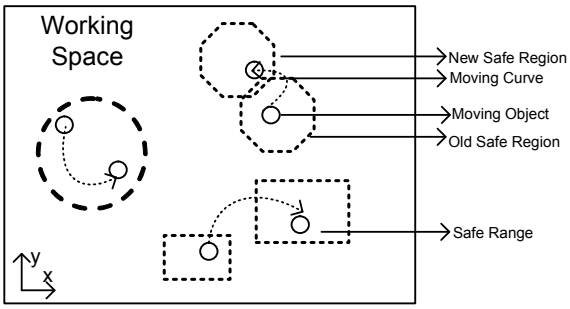


Figure 3.6: Safe Region Distance-based Protocol

polygon. Once the agreement is set up, the object needs only to report to the server whenever it exits the region. If the object stays inside the region, all updates are saved. On the other hand, at the server-side, the location server answers queries using the location information provided by the safe region. Probing is triggered only when the query requirement cannot be fulfilled by safe region location information. An example for safe region and its updating activity is illustrated in Figure 3.6.

3.3 Query Processing Model

Besides object location updating, query processing is the other important activity in the client-server location management model. Query issuers send location-dependent queries to the query processor at the server-side through a wireless/wired communication channel. The query processor examines the queries received, sends service requests to the location server and gets back object location information to compute the results which are to be returned to the issuer. Figure 3.7 depicts the main procedure involved in query processing. In this section, several common query types in the location-dependent information system are introduced first. Classification and common strategies for handling these queries are presented. Two query processing protocols which can control the query precision are finally described.

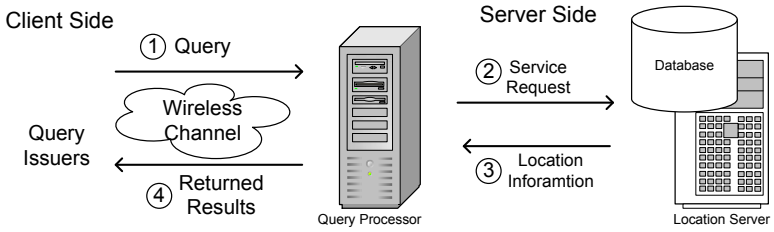


Figure 3.7: Query Processing Procedure

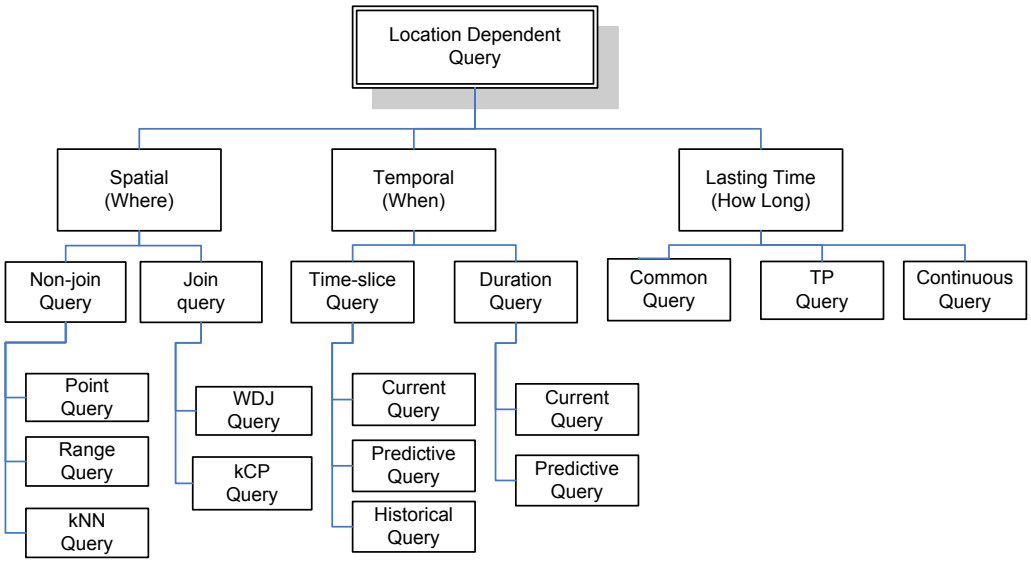


Figure 3.8: Query Classification

3.3.1 Query Classification

Location-dependent queries are concerned with both spatial and temporal aspects. Therefore, they can be categorized according to two orthogonal dimensions, namely spatial dimension and temporal dimension. Also, depending on the duration of a particular query, it can be classified as a typical query or a time-parameterized query or even a continuous query. Figure 3.8 shows the categorization for location-dependent queries. In short, for a typical location-dependent query, three questions should be asked: *when*, *where* and *how long*. Common query types are then presented.

When is concerned with the temporal aspect of the queried objects. It can be a time point or time duration. For both point and duration, there is a special querying

point: NOW. Before NOW, the queries are concerned with the historical data while after NOW, the queries are regarded as predictive ones which are asking for situations that have not yet happened but should be predicted.

Where is concerned with the spatial relationship among the queried objects. The categories under this dimension follow the traditional spatial database practice, ranging from a simple point query that asks for the position of one single object to a very complicated join query that questions the relationship between two sets of objects stored in the moving object environments.

How long is concerned with the lasting time of the query itself. If the query is issued once, it is a typical common query. Some special queries are registered with the query processor first and then the query processor keeps performing that query for a period of time.

3.3.1.1 Common Query Types

Point Query

This is the most typical and basic location-dependent query among all types of location-dependent queries. It asks the spatial position of a particular object at time T. T can be a time point such as NOW, 7:00 AM this morning (past time), 3:00 PM tomorrow or can be a time duration such as from NOW to 3:00 AM tomorrow, or from 5:00 PM to 6:00 PM last Sunday.

Range Query

Range query is one of the most common query types addressed by a lot of research work. A certain range query specifies a query region R and a time interval T. Its aim is to find all the objects whose locations are within R during T. According to the specification of T, a range query can be further categorized to time-slice query in which T is a time point and window query (WQ) in which T is a time duration.

kNN Query

A k nearest neighbor (kNN) query specifies a query point Q and time interval T. Its aim is to find k objects whose distance to Q during T are the smallest among all the objects stored in moving object environment.

Location-Dependent Joins

While the above queries involve only one dataset, the location-dependent join query involves two datasets. In other words, it is a kind of query that deals with one set of objects combined with another set of objects in a moving object environment. Several common subtypes are listed below.

WDJ: Within-Distance Join query. Given two datasets S_1, S_2, a WDJ reports all the object pairs $\langle o_1, o_2 \rangle$ in the Cartesian product $S_1 \times S_2$, such that the distance between o_1 and o_2 during a query time interval T is smaller than a certain threshold d.

kCP: k Closest Pair query. Given two datasets S_1, S_2, a kCP reports k object pairs $\langle o_1, o_2 \rangle$ such that the distance between o_1 and o_2 during T is the smallest, among all the pairs in $S_1 \times S_2$. Note that this kind of queries can be defined in both historical and predictive context according to the definition of T.

3.3.1.2 Time-evolving Queries

Time-parameterized (TP) queries and continuous queries are not under any type of the above location-dependent query categories. However, all types of location-dependent queries discussed above can be extended as a TP or continuous query. Compared to the traditional query types, TP and continuous query introduce special solutions which aim at addressing the novel dynamic nature of the moving object environment.

The reason to introduce the *How Long* dimension is because the processing approach of traditional queries (i.e., range query, kNN, WDJ etc.) is inadequate. The results returned from the algorithms that process traditional queries may change due to the movements of the objects or queries. To overcome this problem, the time-parameterized (TP) query is introduced to return in addition to the traditional results

for any traditional query, with an expiry time T for the results and the set of changes of the results after T.

Continuous queries further extend the TP query and its aim is to continuously track the result changes until certain conditions are fulfilled. It can also be applied to any traditional query type such as range query and kNN.

3.3.2 Query Processing Protocols

Every location-dependent query issued from the query issuer to the server-side query processor is expected to be associated with certain *query precision requirement*. The query precision requirement is the requirement for the quality of query results returned from the server to the issuer. This precision requirement is assumed to be provided at the time a query is issued, and the query precision expectation is subject to the needs of the application. For different query types, the definition of query precision may vary. For example, for a point query, the precision requirement is normally regarding the distance deviation between the real position and the reported position while in a range query, the precision requirement is regarding the ratio between the number of correctly reported objects that are really residing in the queried region to the number of all reported objects.

Quantifying query precision has several benefits.

- Make sure that the system meets the service level that a particular application requires;
- Provide another important metric besides resource usage to examine system performance;
- Provide the freedom for query issuers to control the result accuracy.

The most challenging aspect of providing precise query results to the issuer is the dynamic nature stemming from the continuous location changing of the objects. This

is because location information stored at the location server cannot be updated continuously due to resource limitation and that stale location information may be used to answer queries, resulting in imprecise results being returned to the query issuer. If the precision of the results is too low and beyond the issuer's expectation, further refining work should be done by server-side components. Based on the action taken by the server to deal with imprecise results, two query processing protocols are proposed, namely, the *Eager-probing Protocol* and the *Lazy-probing Protocol*. Figure 3.9 shows the flow chart of query processing. The flow chart summarizes how the two protocols function. The following sections explain the protocols in detail.

3.3.2.1 Eager-probing Protocols

In eager-probing protocols, both query issuer and server-side processor make an agreement that only query results whose precision meets the requirement can be returned. This agreement ensures that the stale location information can be updated immediately whenever needed.

The instantaneous location information updating activity is realized by server probing. The location server identifies the relevant objects that affect the query results and probes these objects through a wireless channel. At the client-side, upon receiving a server probing message, the moving object would send a location update message reporting its current position.

3.3.2.2 Lazy-probing Protocol

As the name suggests, the lazy-probing protocol involves no instantaneous probing action from the location server to moving objects. In this type of protocol, the query issuer and server-side processor agree that imprecise query results can be tolerated. Results with possible errors could be returned to the issuer without immediate probing.

At the client-side, the query issuer evaluates the returned results. Two actions may

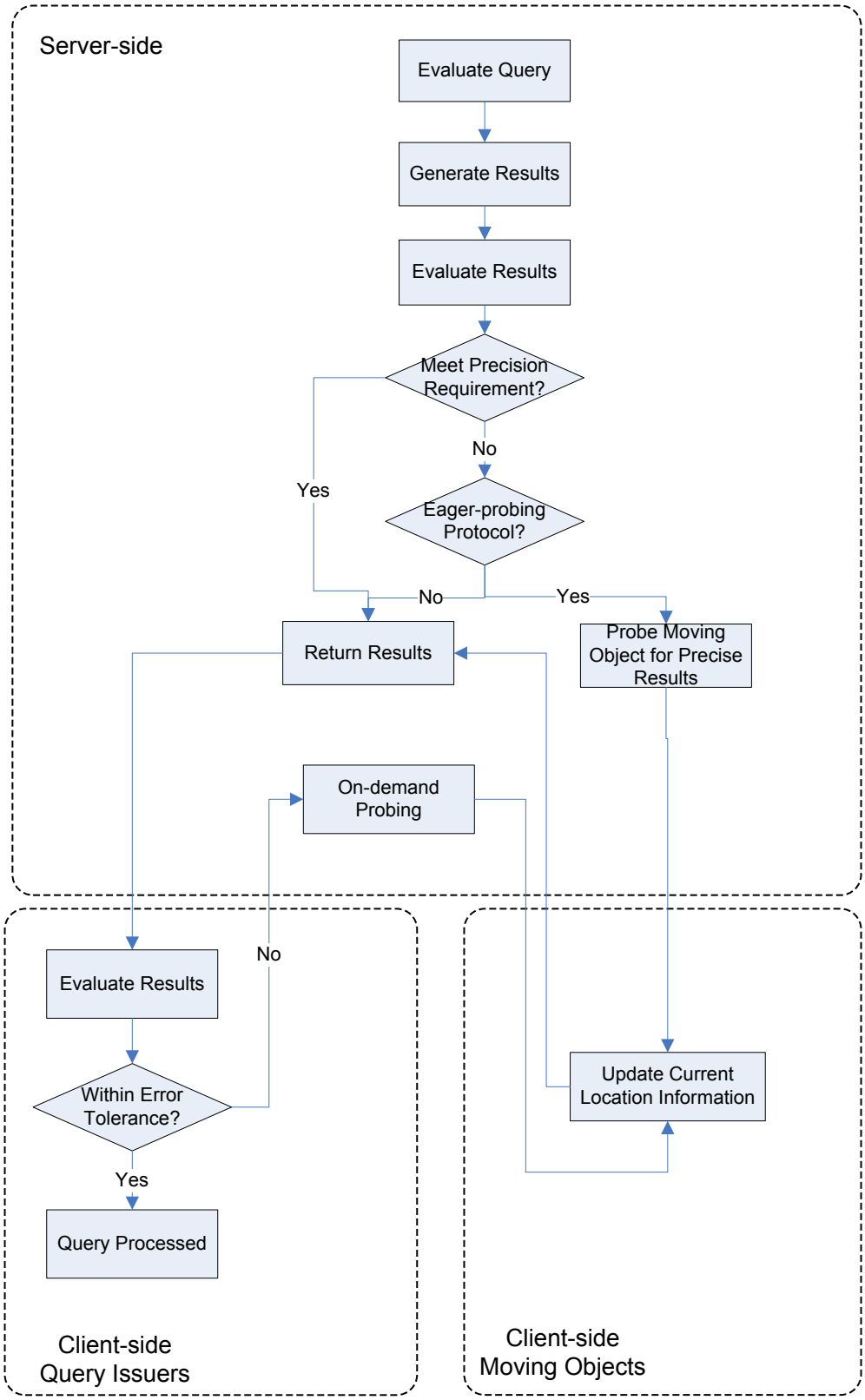


Figure 3.9: Query Processing Flow Chart

be taken. First, although the results contain errors and are not precise enough, they are within the error tolerance that the issuer can bear. In this case, no further action should be taken and the query is successfully processed. Second, errors are out of the tolerance level and the issuer sends back the query for re-evaluation. The queries sent for re-evaluation may require on-demand probing to make sure that satisfactory query results are returned.

3.4 Tradeoff Problem between Updating and Querying

Given that objects keep moving all the time, monitoring a large number of moving objects is not a trivial task for any client-server location management system. The main performance concern for location monitoring is resource consumption. Without resource consumption consideration, the location server can maintain precise location information for every object at anytime by forcing the object to update its new position whenever there is a change. However, resource limitation is an inherent characteristic of mobile environments in which the network connectivity is weak and objects have short battery life.

Fortunately, strictly exact location information is not always required by location-dependent applications. Rather, most applications often permit imprecise estimate of the actual location. Instead of exact values, these applications are able to tolerate some bounded errors in the returned data values. As a result, location management can be viewed as addressing the problem of providing uncertainty bounds for each moving object running in the whole working space covered by the service system. The settings of the uncertainty bounds for moving objects would affect both location updating and query processing and vice versa.

As stated previously, the two main issues of location management are: efficiently updating the continuously changing position information of moving objects to the location server and providing fairly precise results for queries issued to the server. As the

uncertainty bounds setting is the key for these two issues, there is a tradeoff between updating and querying when setting the bound, a problem that needs to be addressed properly.

The uncertainty bound which reveals the tradeoff problem can be expressed in various forms according to different location management application scenarios. Take the distance threshold as an example. Although the exact x-axis and y-axis of an object o cannot be known in a digital map, o cannot be far away from its database stored value by more than a distance of d. Here, d is an uncertainty bound agreed upon by the application semantics for any query concerning the current location of o.

For a certain location management scheme, the d value is the uncertainty bound which is provided by this particular location management scheme to the query issuer. This uncertainty bound leads to efficient approximation in object location updating and querying but also raises the issue of tradeoff between resource usage and query precision. On the one hand, a larger uncertainty bound leads to efficient resource consumption for the whole system, but it causes answers to become imprecise. On the other hand, a smaller uncertainty bound could decrease the uncertainty in the database location with more precise query answers, at the expense of increased resource consumption such as bandwidth and energy consumption in the moving object.

How to leverage the amount of uncertainty and to arrive at good system performance becomes an important issue for updating and querying moving objects in a mobile environment. The tradeoff between these two factors is a core issue that should be appropriately addressed for an efficient location management scheme.

Table 3.4: Features of Proposed Models

Proposed Model	Query Awareness	Movement Awareness	Cost Optimization	Error Tolerance
Query-aware	√	√		
Cost-based	√	√	√	
Extended Cost-based	√	√	√	√

3.5 Desirable Features for Location Management Models

Despite most previous research efforts, to design a good location management scheme handling the tradeoff problem well is still an open issue that remains unaddressed. Much previous research work considers location updating and querying as two separate activities which are handled by the location server and the query processor individually. The interdependency between queries and location updates has not been explored well by previous research work.

The query processor can provide up-to-date querying information to the location server for performance improvement. However, this part is ignored by much previous work. Making use of the bi-directional communication between the query processor and the location server would be a desirable feature of a novel location management scheme. A query-aware scheme is designed. The word "query-aware" means that the updating scheme in the location management model is aware of the querying situation and adapts to this situation for the best resource usage.

Table 3.4 lists all the desirable features. These features are desired for more efficient location management schemes but have not been studied in previous work. In this thesis, several novel location management models are proposed to implement these features as listed in Table 3.4.

3.5.1 Feature: Query Awareness

The main reason why a location management model should address the uncertainty problem is that the old location record stored in the location server may produce incorrect results when used to answer queries. In other words, query processing on moving objects is the main purpose of moving object location tracking with respect to most practical applications. Therefore, how the queries are issued may have impacts on location updating and uncertainty management. Unfortunately, none of the previous approaches addressing the uncertainty problem consider the possible influences of query patterns to the location management models.

The limitations of the previous research which does not take query patterns into consideration is that all the efforts the previous research made can only set a good balance between resource consumption and the uncertainty, but the system performance as a whole still cannot be improved by leveraging the tradeoff only.

3.5.2 Feature: Movement Awareness

No matter how efficiently the system can address the tradeoff, managing the moving objects still needs a large number of updates. The updating activity can be handled by the movement-aware feature exhibited by the moving object. For example, moving objects with varying moving speeds may be set with different updating frequencies. Movement prediction is also included in movement-aware feature. If the current or future location of each moving object can be efficiently predicted based on some information either provided by the moving object itself or historical data, continuously updating can be eliminated and a location management model with lower resource consumption and smaller uncertainty risks can be achieved.

3.5.3 Feature: Cost Optimization

Apparently, the best way to handle a tradeoff when it is inherently generated from two competitively leading factors is to optimize it. The very first step towards the optimization is to quantify the abstract term, namely the tradeoff between the object update cost and processing cost of user initiated queries. Intuitively, the cost based approach is the most natural choice for the quantification procedure because the system performance is measured by resource consumption, which is subject to some kinds of costs.

Without loss of generality, location update cost and query processing cost are considered as the two major system costs which are important to system performance. The location update cost is basically the cost of locating the moving object. It includes battery power consumption (at client-side), location maintenance overheads (at server-side), and communication costs between the client and the server. The query processing cost refers to the cost induced when the less precise locations are used to compute for query answers. This kind of cost mainly results in the need of probing the objects for their current location.

As cost consumption is the performance metric, cost optimization is a desirable feature of novel location management models.

3.5.4 Feature: Error Tolerance

Figure 3.9 shows that there are two query processing protocols. The lazy-probing protocol can tolerate errors in the query results returned to the query issuer. Error tolerance is also a desirable feature when the query issuer needs replies from the server-side components as soon as possible without probing. This is because probing may lead to unpredictable waiting time. As most previous work assumes probing as a common practice and does not take the response time into account, the proposed scheme in this thesis aims to remove this limitation by implementing the lazy-probing protocol with

the error tolerance feature.

Chapter 4

Query-aware Model

In this chapter, the query-aware location management model is introduced. A distinguished feature of this kind of model compared to previous work is that in this model, the query information is taken into account in location monitoring. In Figure 3.1, the query information is passed from the query processor to the location server and this part of mutual communication between the query processor and the location server is only applied in the proposed query-aware model. In this chapter, background scenarios and examples that motivate the design of the query-aware model are presented first, followed by the analysis on how this model can help to improve the system performance. Two important components of the model are then introduced, namely, the query-aware updating model and the query-aware querying model. Based on these models, the query-aware scheme *Aqua* is presented in detail and finally simulation studies are conducted to show some possible benefits and performance improvements provided by Aqua.

4.1 Background

The query-aware model is motivated by the observation that query processing on moving objects is the major subscriber to moving object location updating. We are more

interested in the positions of the moving objects returned for the queries than those objects which are seldom queried.

There is an observation on daily life scenarios. Consider an *Intelligent Transportation Systems* launched in a city for answering queries from drivers, passengers, police and other interested parties. Typical queries include "*What is the congestion level on I-10 near downtown Los Angeles?*", "*What are the nearest taxicabs to me when I am at the junction of State Street and Second Street?*" and "*Which street is the bus that I'm waiting for at now?*". It can be expected that the querying pattern exhibits a strong temporal and spatial property. More queries would be issued during the peak hours in early morning or late afternoon and against downtown area or against a partial set of so called *hot* objects. Generally speaking, most queries either concern certain hot-spots or hot objects. In the last query example, the downtown area is a hot-spot area and the objects with type "bus" are hot objects. Both *hot area* and *hot objects* are popular for queries issued by the user in the whole system.

One should distinguish moving objects that are seldom queried (like those objects which are far away from downtown or personal cars) from those frequently queried objects (those which reside in the downtown area or public transportation vehicles). Intuitively, in the former case, it is not necessary to update the object location since almost no one is interested in it. Very infrequent update or probing/paging technique can be adopted to return the object location. In the latter case, a higher update frequency should be made, despite the relatively slower movement of the object due to traffic jam, thereby reducing the uncertainty involved in the stored location information and hence enhancing the precision of the query result. This is the central idea behind the query-aware model.

Incidentally, this observation has not been exploited in previous location management models. Most previous work regards querying and updating as two separate procedures in moving object environments. Based on this thinking, the object monitoring methods were designed and improved by considering factors that affect updating pro-

cedures only. In most practical systems, the only reason to provide up-to-date location information is to provide answers precise enough to queries concerning these objects. Therefore, if no query is issued for the whole trip of a moving object, it is not necessary to ”produce” the information for the non-existent ”consumer”. Thus, both movement pattern and query information are integrated into the monitoring method of the object.

There is one piece of work which proposes an adaptive monitoring method for location-dependent continuous queries [41]. Their approach integrates querying issues into the updating procedure and strives to increase the correctness of the query results. The basic idea is that those objects that fall into a query region receive a close monitoring, so a small update threshold is used, while those objects outside the query region are assigned larger threshold values. The approach taken in this thesis also assigns adaptive thresholds for each object.

Unlike previous work, which is based on the location of the objects only, the setting in this thesis is changed dynamically based on both the query pattern and movement pattern. The proposed strategy in this thesis aims to absorb both querying and updating procedures into system performance consideration. To sum up, the purpose of the query-aware model is to reduce the communication cost and gain overall improvements in terms of query precision by combining query information into updating activities. The motto in this thesis is to invest the resource (location updates) wisely.

4.2 Location Updating Issues in the Query-aware Model

The query-aware location updating model is designed based on the general location management model introduced in Chapter 3. It is basically a distance-based updating protocol which makes use of the distance bound to decide when and how often an update should happen. Client-side activities of the query-aware location updating model are summarized in Figure 4.1.

The location server also communicates with the query processor and provides query-

Procedure for moving object o_i
1: repeat
2: collect its own location p_i and current time via its positioning device
3: if distance threshold is exceeded then
4: send to the server the update report: $\langle o_i, p_i \rangle$
5: $p_i^{reported} \leftarrow p_i$
6: negotiate with the server for new distance threshold
7: endif
8: forever

Figure 4.1: The Procedure for Moving Object

aware location monitoring services. The procedure running in the server is described in Figure 4.2.

Procedure for the location server
1: repeat
2: receive a message, m
3: if m is an update report from o_i then
4: update the stored location of o_i with new p_i
5: negotiate with o_i the new update condition
6: else if m is a service request from the query processor then
7: return the required object's location information to the query processor
8: endif
9: forever

Figure 4.2: The Procedure for Location Server

The service request issued from the query processor is normally a location retrieval of a set of moving object records. The returned information includes not only the recorded positions of each required object but also the distance threshold it may deviate from its current position.

4.3 Query Processing Issues in the Query-aware Model

4.3.1 Query Processing

The query processor examines the queries issued, sends service requests to the location server and gets back object location information which is used to compute the results to be returned to the issuer. Figure 4.3 depicts the main procedure running for query

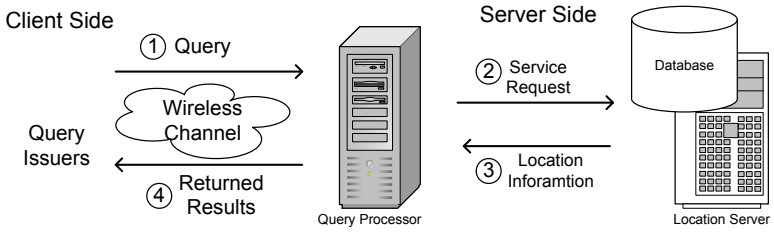


Figure 4.3: The Procedure for Query Processor

processing.

The service request that is passed from the query processor to the location server includes identifiers of a set of requested objects, represented as $\mathcal{O} = \{o_1, o_2, o_3, ...\}$. Matching the identifiers using the index in the database, the location server retrieves the location information of the requested objects and returns the information back to the query processor. The returned message has the format $\mathcal{O} = \{< o_1, p_1, d_1 >, < o_2, p_2, d_2 >, < o_3, p_3, d_3 >, ...\}$. Here p represents the current position of the requested object and d represents the agreed deviation (i.e. distance threshold) with location information. Having the location information, the query processor continues to filter the results and returns query results to the query issuer. The query results that the query issuer obtain are based on the agreed distance bound.

In previous work, as long as an agreement on the distance bound between the query issuer and the system has been decided, the query issuer has no control of the accuracy of the result. In the query-aware model proposed in this thesis, this limitation is removed by providing the adaptive threshold.

To examine the performance improvement that the query-aware model may achieve, the concept of query precision is defined formally. The formal measurement of the precision is also provided.

Quantifying query precision has two benefits:

- It can provide an important metric to examine system performance. Previously, query performance lies on the agreement on location deviation and this metric is not accurate enough.

- It can provide the freedom for the query issuer to control the result accuracy. Query results returned can be measured and if the precision is too low beyond the issuer's expectation, further refining work can be required.

Two common query types and their precision definitions are described in the next section.

4.3.2 Query Analysis

4.3.2.1 Point Query

Point queries are object-based. They search for the locations of specific objects at the time the queries are issued. Examples of point queries are "Where is BUS No.104 now?" and "How far is BUS No. 104 away from the Second Stop?". In a distance-based monitoring scheme, results returned to these point queries are of the format: $< o_i, p_i, d_i >$. Here, o_i is the requested object; p_i is the returned location at the queried time; d_i is the maximum distance threshold that the returned location results may deviate from the current location.

Depending on various scenarios, the query issuer may have different requirements on the results at different time. Some may be fulfilled with the agreed deviation while others may not. Suppose every query q_j has its *deviation requirement* r_j and the result returned has a threshold d_i.

The precision of the point query q_j regarding an object o_i is also provided using a nonnegative value $Prec(r_j, d_i)$. The query issuer makes use of this value to examine the performance of the monitoring service. Results returned with low precision whose value is under the acceptable level can be revised by further probing for the object to obtain more precise location information.

To compute $Prec(r_j, d_i)$, the relationship between r_j and d_i is examined. A *Precision Region (PR)* is defined which has p_i as its center and r_j as its region radius and a *Possible Moving Region (MR)* which has also p_i as its center but d_i as its radius.

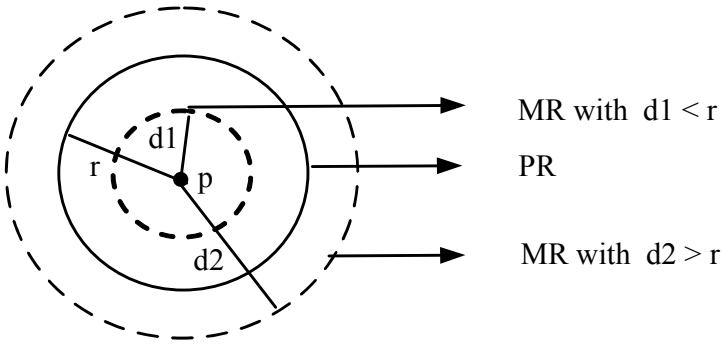


Figure 4.4: Point Query Processing

The probability that the object may reside in the precision region *PR* is taken as the precision.

Figure 4.4 depicts the scenario. When $r_j \leq d_i$, the probability that the object resides in the *PR* region is equal to 1. This means that the query result can fulfil the requirement set by the query issuer and $Prec(r_j, d_i) = 1$. When $r_j > d_i$, the precision is never equal to 100%. The probability can be computed as the ratio between areas of *PR* and *MR*, $\frac{Area(PR)}{Area(MR)} = \frac{\pi r_j^2}{\pi d_i^2} = (\frac{r_j}{d_i})^2$. To generalize the precision equation, the query precision for a point query is defined as: $Prec(r_j, d_i) = \min(1, (\frac{r_j}{d_i})^2)$

4.3.2.2 Range Query

Range queries concern specific spatial regions and search for those objects whose locations are within these specified regions. Typical range queries are "Report all taxicabs which are within 500m of my current position." and "which buses are now at Time Square? " To process this kind of queries, the query processor should examine the query region and ask the location server to return all possible objects which have a chance to reside within the region. The location server returns objects' locations and their deviations (i.e. distance threshold) with the format $< o_i, p_i, d_i >$. With this location information, the query processor can categorize the returned objects into two sets: a certain set and an uncertain set.

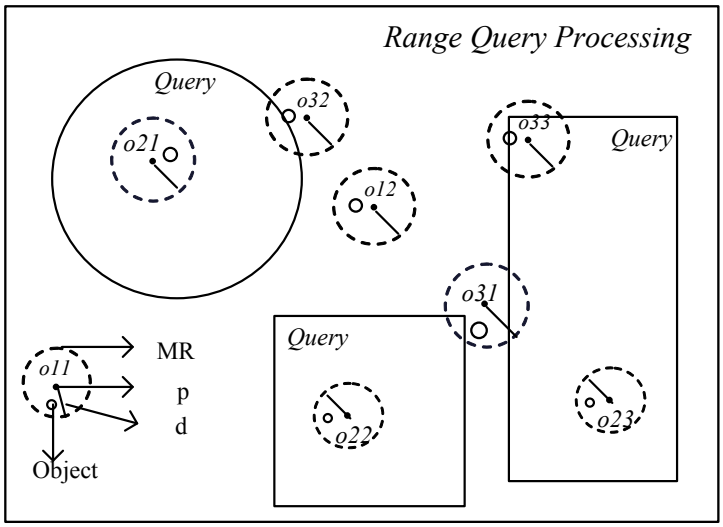


Figure 4.5: Range Query Processing

Figure 4.5 depicts the categorization. Several range queries with different region shapes are issued for processing. Each object is represented as a moving region (*MR*) with its p as region center and d as region radius. Objects whose *MR*s have no overlap with the query region (e.g. o_{11}, o_{12}) are not returned by the location server because they have no chance of residing within the region. Objects whose *MR*s are fully covered by the query region (e.g. o_{21}, o_{22}) are categorized to the certain set. Objects whose *MR*s are partially covered by the query region (e.g. o_{31}, o_{32}) belong to the uncertain set.

Two kinds of results can be generated by the query processor to answer queries from the issuer according to the issuer's requests.

- *Conservative Result:* returns the certain set only and the issuer are ensured that the returned objects are really within the query region.

- *Full Result:* returns both the certain set and uncertain set and the issuer may obtain some objects which do not reside in the query region.

The limitation of returning either type of result sets is that the query issuer has no control of the quality of the result. With *Conservative Result*, the query issuer may miss some objects which may reside within the region but their *MR* is not covered by

the query region. With *Full Result*, the query issuer may receive objects which are not moving in the region but their *MR*s are overlapped with the query region. To obtain results with quality between the two extremes, the precision definition is needed for the range query.

There are two metrics to examine the precision of the range query: precision and recall. The precision (Q_p) is defined as the ratio between the number of correct returned objects and all returned objects. The recall (Q_c) is defined as the ratio between the number of correct returned objects and all correct objects. Here, a correct object is one which is actually residing within the scope of the query region. According to the definition, Conservative Results have 100% Q_p but fairly low Q_c while Full Results have 100% Q_c but relatively low Q_c. To be able to control the query precision between the two extremes, the concept of the object precision for the range query should be introduced first.

The object precision for object o_i in a range query q is defined as the probability that o_i may reside in the query region. Take objects in Figure 4.5 as examples. $Prec(o_{11}) = 0$ and $Prec(o_{21}) = 100\%$. Assuming that the locations of the objects follow a uniform distribution in the whole space, then the object precision can be generally computed as $Prec(o_i) = \frac{Area(OverlapRegion)}{Area(MR_{o_i})}$.

With this object precision defined, the query issuer can provide a precision threshold with each range query. This precision threshold functions like a filtering heuristic that decides whether an object should be returned as a query result. Only those objects whose precision is higher than the threshold should be returned.

In this way, an approximated result set is computed based on a heuristic threshold which filters the result set. The result set is said to be approximate because an object with a low precision being filtered by the precision threshold still has the chance to reside in the query region and this fact yields imprecision in the result. Also, an object which is partially overlapped with the query range and has a higher precision has the chance to reside outside the range.

The precision threshold is the key for the range query precision. As in Figure 4.5, if the threshold is small (unwilling to miss a potential object), both o_{31} and o_{33} will be returned. If the threshold is large, both will not be returned. With a medium value, o_{33} may be returned while o_{31} may not be returned. Two extreme values are 0 which leads to *Full Result* and 1 which leads to *Conservative Result*. Between value 0 and 1, the query issuer may adjust the threshold to obtain acceptable precision and recall.

4.4 Aqua

The query-aware scheme *Aqua* is proposed in this section. *Aqua* is an **A**daptive **QU**ery-**A**ware location updating scheme. It is built upon general query-aware updating and querying models introduced in previous sections. There are two detailed design issues that need to be addressed in the Aqua scheme.

- Distance-based Protocol Setting: among several distance-based updating methods, which one is chosen for Aqua? Also, in the chosen method, how to set the distance bound/threshold for updating activity is the essential design point for a specialized scheme like Aqua.
- Query Pattern Collection: as query-awareness is the goal for this scheme design, how to collect the query pattern of different query types for system improvement is an important issue that should be addressed.

The following sections present the solutions to these two design problems in detail.

4.4.1 Distance-based Protocol Setting

4.4.1.1 Adaptive Threshold Setting

The query pattern is characterized by the *query arrival rate* λ. For a specific moving object, the larger λ is at a specific moment, the smaller d should be defined. Whether a

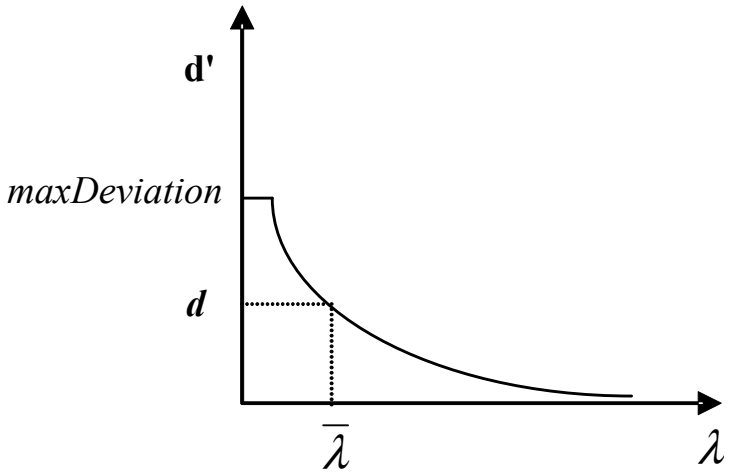


Figure 4.6: General Relationship among Factors Affecting Threshold

query arrival rate λ is large or not depends on the overall query arrival rate over time, $\overline{\lambda}$.

In the Aqua scheme, the query-adjustable-deviation d' is provided for each object. d' is defined based on an initial threshold d_0 and the query pattern. Thus, $d' = g(d_0, \overline{\lambda}, \lambda)$, where g is a generic function to be defined properly. The general idea is shown in Figure 4.6 and the relationship among the three factors is examined more clearly. d' values for different objects are set adaptively according to the query pattern. Intuitively, the more frequently an object is involved in query results, the smaller its d' value should be. The reason of defining *maxDeviation* is to set a bound on the adaptive deviation for the extreme cases. For example, when the current query arrival rate λ of an object is equal to or close to 0, d' will be unreasonably large, despite the need from that object to update its current position and occasionally this can lead to unacceptable location imprecision.

Among common functions, e.g., power function, exponential function, logarithmic function, a decreasing power function ($f(x) = x^{-\kappa}$ with $\kappa > 0$) appears to be a most appropriate one for g, since both f and g match the general relationship among the

three factors for d', with similar function properties. Thus d' can be defined as:

$$d' = \min(d_0(\frac{\lambda}{=\overline{\lambda}})^{-\kappa}, maxDeviation) \tag{4.1}$$

Here, κ is a system parameter for performance tuning; d_0 and $\overline{\lambda}$ are known to the system.

4.4.1.2 Prediction Protocol

The prediction protocol can be applied to the Aqua scheme. First of all, the concept of *quasar* is proposed to implement the query-adjustable-deviation technique.

Quasar stands for **QU**ery-**A**djustable moving **SA**fe **R**egion. A *safe region* is defined here as a region in which a moving object can be found located. An *adaptive safe region* expands with different speed according to the speed of the moving object [46]. The adaptive safe region is further extended to *quasar* by allowing the center and hence the whole safe region to move. Conceptually, *quasar* is defined as a moving circular region out of which the object should send an update message to the location server. The area or covering scope of *quasar* is defined adaptively for different moving objects and at different moment for the same object.

Formally, a *quasar* is expressed as $\langle c^q, r^q \rangle$, where c^q and r^q are the region center and region radius respectively. c^q is a moving point which indicates the predicted position of a moving object, while r^q bounds the maximum allowable deviation from the predicted position. c^q is modeled as a function of time $f(t)$. The actual region center c^q should be computed on-the-fly according to a specific function whenever *quasar* is used. The adaptive nature of *quasar* comes from the adjustable setting of r^q. Applying a general adaptive deviation function, r^q is defined as follows:

$$r^q = \min(d(\frac{\lambda}{=\overline{\lambda}})^{-\kappa}, maxr^S) \tag{4.2}$$

Note that in the general function, *maxDeviation* is set as a bound on the adaptive

deviation for the extreme cases. Here, $maxr^S$ is used for the same reason. By setting r^q adaptively, Aqua actually trades the communication cost of infrequently queried objects for those frequently queried ones. The immediate benefit is a reduction in the communication cost when the query distribution is skewed with some hot objects or objects moving around hot areas. A large amount of location updating messages from less interested objects can be eliminated. Even if no communication cost can be saved, reducing *quasar* size can lead to better precision results. Despite the simple power function adopted, Aqua performs surprisingly well in delivering a satisfactory performance under the shadow of inevitable communication cost and precision tradeoff.

4.4.2 Query Pattern Collection Methods

Query-adjustable-deviation is passed from the location server to moving objects. To compute the function, the location server needs query information provided by the query processor because the query pattern is collected there.

The query pattern reflects the query arrival rate for different objects. To collect the query pattern of each single object is resource inefficient. A group-based approach is then taken in the collection procedure. According to the application nature, there are two kinds of grouping methods. One is based on object nature. The other is based on spatial nature.

4.4.2.1 Grouping Methods

The first grouping method is for the *point query* which is object-based. For example, in the Intelligent Transportation Systems application, a taxi service provider may be interested in a point query like "Report the location information of all taxicabs that are available to the service center." Some objects (e.g. in this case the taxicabs) are hotter which have larger query rate and smaller distance threshold.

In the system, N groups are defined, i.e. $\mathcal{G} = \{G_1, G_2, G_3, ...G_N\}$. Each group G_k

has a group feature F_k (e.g. taxicabs or buses). An object that has Feature F_k belongs to G_k. Every object can belong to one and only one group. To define formally, $o_i \in G_k$ iff o_i has the feature F_k. The query issuing pattern for each group of objects is recorded and monitored at the server. It is reasonable to expect that query arrival rates differ for objects from different groups. This difference is used to determine the query pattern and thus the adaptive d' (i.e. r^q in *quasar*) values.

The other grouping method is based on spatial nature which is suitable for *range query*. Let $\mathcal{D}$ be the spatial domain, i.e., the entire region covered by the mobile computing environment, within which moving objects can freely move around. $\mathcal{D}$ can be divided into non-overlapped sub-regions, i.e. $\mathcal{D} = D_1 \bigcup D_2 \bigcup D_3 \bigcup ... D_n$. The group definition is similar to that in the object-based definition. In the object-based definition, N groups are defined, i.e. $\mathcal{G} = \{G_1, G_2, G_3, ...G_N\}$ and $o_i \in G_k$ iff o_i has the feature F_k. In spatial definition, group feature is specified as region feature. This means objects that move within D_k belong to G_k.

Certain sub-regions in $\mathcal{D}$ are of stronger interests (witnessing higher query arrival rates) than others, which we call "hot regions". The query rates in "hot regions" are higher and moving objects residing around these hot regions should be informed that a higher query arrival rate λ prevails. For example, D_k is one of the "hot regions", having λ_k. All objects moving within D_k belong to group G_k and thus could expect the query rate λ_k.

To manage query arrival rates for different regions efficiently, the spatial domain $\mathcal{D}$ is conceptually fragmented into sub-regions, according to the distribution of query arrival rates. For example, we can use a grid model to realize this space fragmentation. Figure 4.7 illustrates the grid model. The entire domain is divided into square grid cells of size z by z, where z is a system parameter.

Each object can map its current position to the grid cell it just moves in and set its λ value to the corresponding value of the grid cell. The λ value for each grid cell is either pre-stored into the moving objects as a default value, or obtained on-the-fly

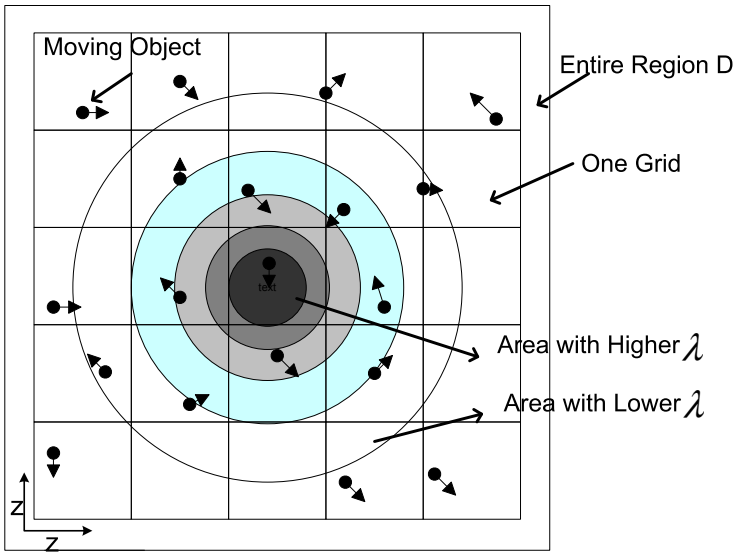


Figure 4.7: Grid Model of Unevenly Distributed Queries in Spatial Domain $\mathcal{D}$

via wireless channels from the server. The former method consumes less downlink bandwidth but is not flexible because the hot and cold sub-regions are only relative and may change over time when the query pattern changes. The latter can adapt the "temperature" of sub-regions to changing query patterns and deliver the changes to the moving object appropriately, at the expense of higher communication cost.

4.4.2.2 Pattern Computation

Suppose the object o_i which belongs to group G_k needs to compute its r^q. The computation needs the cooperation of the server which can provide the up-to-date values of both λ and $\overline{\lambda}$. At the server, it is easy to compute $\overline{\lambda}$ by averaging λ values from all the groups, i.e. $\overline{\lambda} = \frac{\lambda_1+\lambda_2+...+\lambda_n}{N}$ where N is the number of groups in the whole system. The server needs to monitor query arrival activities in order to keep a record of λ_k for a particular group G_k. To track changes in query patterns, the server can make use of an exponentially weighted moving average method according to the following formula:

$$\lambda_{now} = \omega\lambda_{previous} + (1-\omega)/(T_{currentQuery} - T_{lastQuery}) \tag{4.3}$$

where $T_{currentQuery}$ represents the time when current query is issued to G_k and $T_{lastQuery}$ represents the time when last query was issued. ω is the adjustable weight.

According to the equation, update-to-date λ value can be computed and kept in the server. When requested, values of λ and $\overline{\lambda}$ kept by the server are propagated to the moving object, for example, via broadcasting.

4.5 Simulation Studies

In this section, simulation experiments to evaluate the proposed Aqua scheme are conducted. In each experiment, the running time for the movement of the moving objects is 1000 time units. The service area is a square-shaped region of size 100 by 100 units. The Random Walk model is used as the object movement model which is well-known for performance evaluation of object mobility patterns. In the random walk model, all objects move in steps and each step moves a distance of k along an arbitrary direction. Queries arrive with mean rate λ.

4.5.1 Simulation Studies with Point Query

First of all, simulations are conducted to examine the scheme performance for the point query. The total objects are grouped into 10 groups and distribute queries for different groups using zipf's law and setting the exponent to 1. Each query is accompanied by a *deviation requirement* sampled from a uniform distribution, $\mathcal{U}(0, r_{max})$. Table 4.1 summarizes parameters used in all experiments.

Totally, five sets of experiments are conducted. Table 4.2 summarizes the parameter setting for each set.

Parameter	Value Range	Default Value
Number of moving objects	1000 to 10000	1000
Object speed	1, 5, 10	5
Query rate (λ)	1000, 5000, 10000	1000
Initial threshold (d_0)	1, 5, 10	5
Power function parameter (κ)	0.1, 0.2, 1, 2	1
Query requirement (r_{max})	10	10

Table 4.1: Simulation Parameters for Point Query

	# of objects	Speed	Initial Threshold	Query Rate	κ
Experiment #1	1000, 5000, 10000	5	5	1000	1
Experiment #2	1000	1, 5, 10	5	1000	1
Experiment #3	1000	5	5	100, 500, 10000	1
Experiment #4	1000	5	1, 5, 10	1000	1
Experiment #5	1000	5	5	1000	0.1,0.5,1,2

Table 4.2: Experimental Parameter Setting for Point Query

4.5.1.1 Experiment #1: Effect of the Number of Objects

The first set of experiments examines the effect of the number of objects. The query-aware scheme is compared with the updating scheme with the static threshold setting. Two performance metrics are computed for comparison. One metric is the average number of updates and the other metric is the average precision of queries.

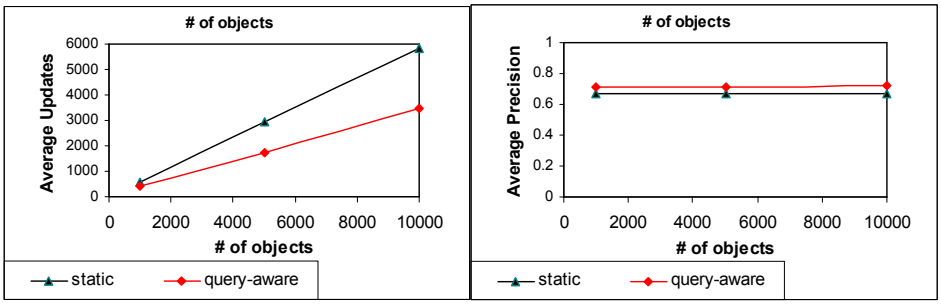


Figure 4.8: Point Query: Effect of # of Objects

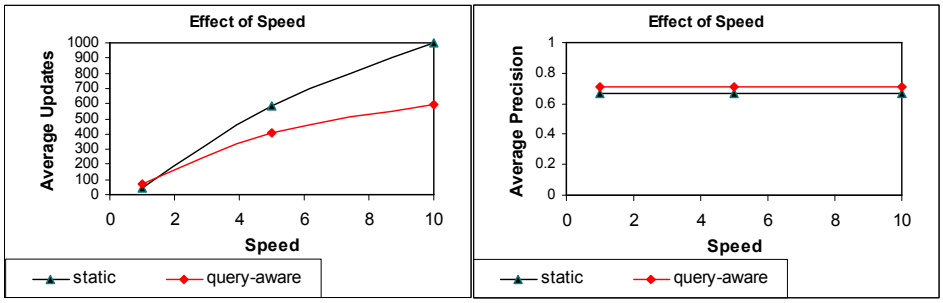


Figure 4.9: Point Query: Effect of Speed

Figure 4.8 presents results of this set of experiments. Obviously, the number of updates is increased with the number of objects. However, the query precision varies little with different number of objects. From the results, Aqua is effective to reduce the total number of updating messages. At the same time, Aqua can improve the performance of querying activities as the precision of the query result is better by using Aqua than that by using the traditional static scheme.

4.5.1.2 Experiment #2: Effect of Speed

The second set of experiments is to examine the situation when objects move with different speeds. Figure 4.9 shows that when an object moves with a faster speed, the tracking needs more updates. It is obvious that the movement speed has little impact on querying activities. Both the number of updating messages and query results are compared between Aqua and the static threshold scheme. The results show that the Aqua scheme with different movement speeds can obtain better performance in both aspects.

4.5.1.3 Experiment #3: Effect of the Initial Threshold

The third set of experiments evaluates the effect that the initial threshold setting brings to the performance. Figure 4.10 presents the results. We can see that when the initial threshold is larger, the total number of updates of both schemes drops. This is

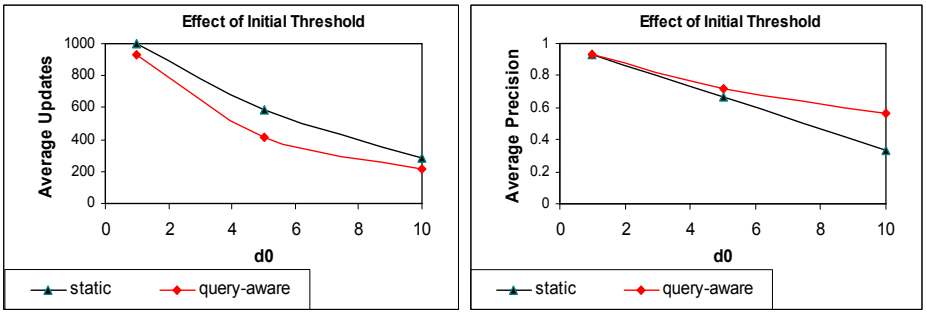


Figure 4.10: Point Query: Effect of the Initial Threshold

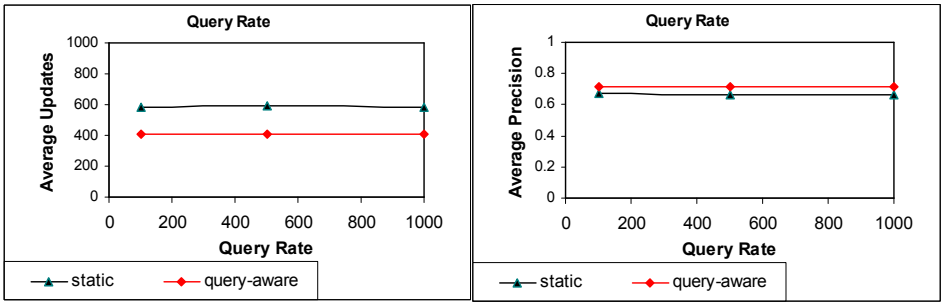


Figure 4.11: Point Query: Effect of Query Rate

easy to understand. A larger threshold gives a larger boundary in which the moving object can move freely without updating its current location. As a result, the number of updating messages is reduced. However, a larger threshold leads to more imprecise location information. This is the reason for the dropped query precision in Figure 4.10. Comparing with the static threshold scheme, the Aqua scheme has less message consumption and higher querying precision.

4.5.1.4 Experiment #4: Effect of Query Rate

The fourth set of experiments examines whether various query rates affect the system performance. Figure 4.11 shows that no matter the query issuing frequency is high or low, the total number of updates remains the same. This is because the updating activities are affected only by the movement factors. Figure 4.11 also shows that the query performance is scalable for both the static threshold scheme and Aqua.

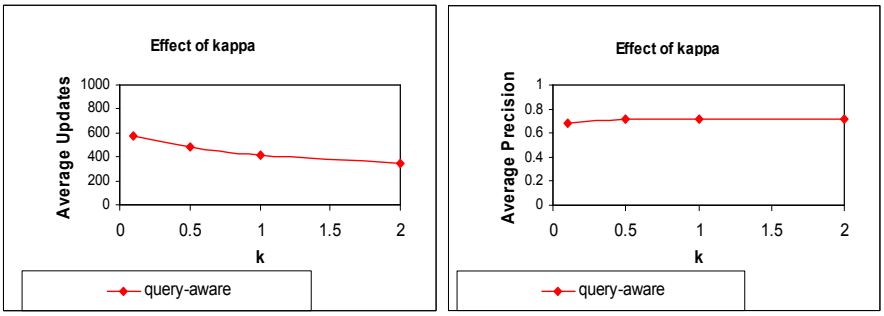


Figure 4.12: Point Query: Effect of κ

4.5.1.5 Experiment #5: Effect of Power Function Parameter

The last set of experiments examines the effect of the power function parameter κ. The results are presented in Figure 4.12. The impact that different κ brings in mainly lies in the updating activities. Query processing performance remains the same with the changes in κ as a fairly straight line as shown in Figure 4.12. Generally, larger κ can lead to fewer total updating messages. This is because larger κ in the power function means more skewed updating policy for different groups of objects. Those objects in infrequently queried groups have larger thresholds and issue fewer updating messages.

4.5.2 Simulation Studies with Range Query

This set of simulations examines the performance of the Aqua scheme for processing the range query. The whole working space is separated into 5×5 grids. Each range query covers a square with area 1×1. The query filtering threshold (i.e. object precision threshold for query answers) is 50%. Queries are placed to the area following normal distribution with the point whose coordinates are $(50, 50)$ as its mean and the standard deviation is the parameter which can affect the performance. Table 4.3 summarizes all parameters used in all experiments. Totally, six sets of experiments are conducted. Table 4.4 summarizes the parameter setting for each set.

Parameter	Value range	Default value
Number of moving objects	1000 to 10000	1000
Object speed	1, 5, 10	5
Query rate (λ)	100, 500, 1000	1000
Initial Threshold (d_0)	1, 5, 10	5
Power Function Parameter (κ)	0.1, 0.2, 1, 2	1
Query std. deviation (Δ)	5, 10, 50, 100	10

Table 4.3: Simulation Parameters for Range Query

	# of Objects (k)	Speed	d_0	λ (k)	κ	Δ
#1	1, 5, 10	5	5	0.1	1	10
#2	1	1, 5, 10	5	0.1	1	10
#3	1	5	5	0.1, 0.5, 1	1	10
#4	1	5	1, 5, 10	0.1	1	10
#5	1	5	5	0.1	1	5, 10, 50, 100
#6	1	5	5	0.1	0.1,0.5,1,2	10

Table 4.4: Experimental Parameter Setting for Range Query

4.5.2.1 Experiment #1: Effect of the Number of Objects

The first set of experiments examines the effect of the number of objects. The Aqua scheme is compared with the static threshold scheme. Three performance metrics are computed for comparison. The first one is the average number of updates. The second one is the average precision of queries. The third one is the average recall of queries. Figure 4.13 presents results of this set of experiments.

In both Aqua and the static scheme, the number of updates is increased with the number of objects. This is because the average number of updates is computed as the total number of update messages divided by time. When the number of objects increases, more updating messages are generated for location tracking activities. Comparing the two schemes, it has been found that the Aqua scheme has a better perfor-

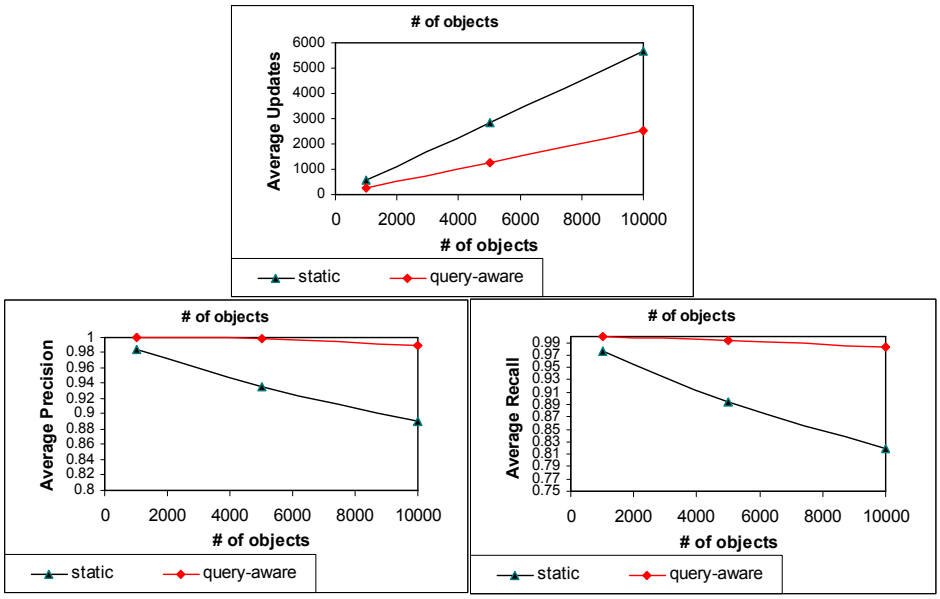


Figure 4.13: Range Query: Effect of the Number of Objects

mance in terms of the increasing trend at a slower pace.

The results in Figure 4.13 show that the precision and recall in the static scheme decrease sharply with the increase of the number of objects. This phenomenon can be explained by the computation methods of both precision and recall. When more objects are placed into the system place, the chance for the query area having overlaps with the object's moving region is increased. As a result, the number of objects which are returned as the query result is increased while the number of correctly returned objects remains the same. This leads to lower query precision. More objects may drop in the query area and the number of real query result objects is increased. As the number of correctly returned objects is the same, the recall is decreased.

In Aqua, no matter how many objects are involved in the system, the querying activities keep steady and perform very well. Both the query precision and recall are close to 100%.

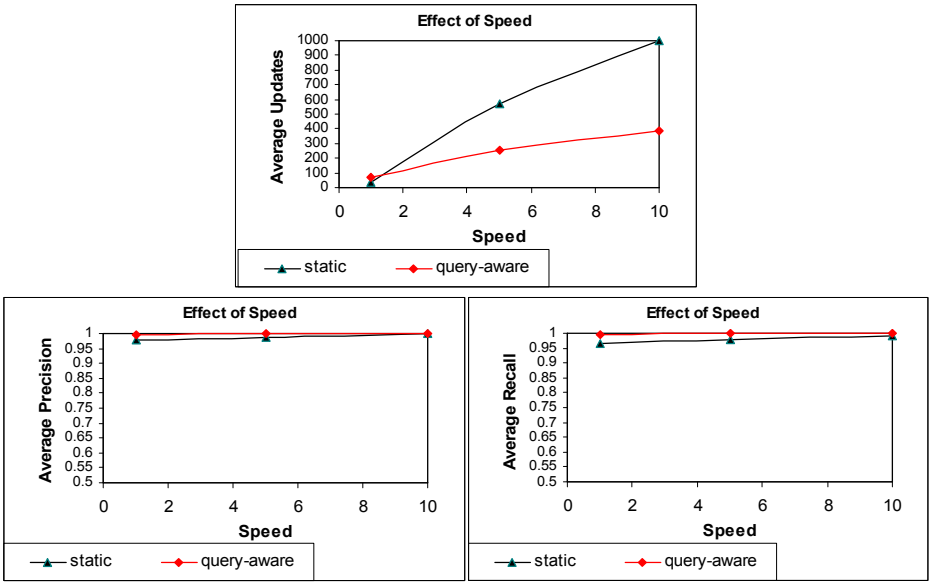


Figure 4.14: Range Query: Effect of Speed

4.5.2.2 Experiment #2: Effect of Speed

The second set of experiments is to examine the situation when objects move with different speeds. Figure 4.14 shows that when an object moves with a faster speed, its tracking needs more updates in both Aqua and the static threshold scheme. This is easy to understand. Faster objects have more chance to move out of their moving region and need to issue more update messages in order to keep their location record in the server up-to-date. Figure 4.14 shows that the Aqua scheme consumes less updating messages under all situations with objects moving at different speeds.

In both Aqua and the static threshold scheme, the variation of movement speed has little impact on the query performance. The Aqua scheme has a better performance in terms of both the precision and recall.

4.5.2.3 Experiment #3: Effect of the Initial Threshold

The third set of experiments evaluates the effect that the initial threshold setting brings to performance. Figure 4.15 presents the results. We can see that when the initial

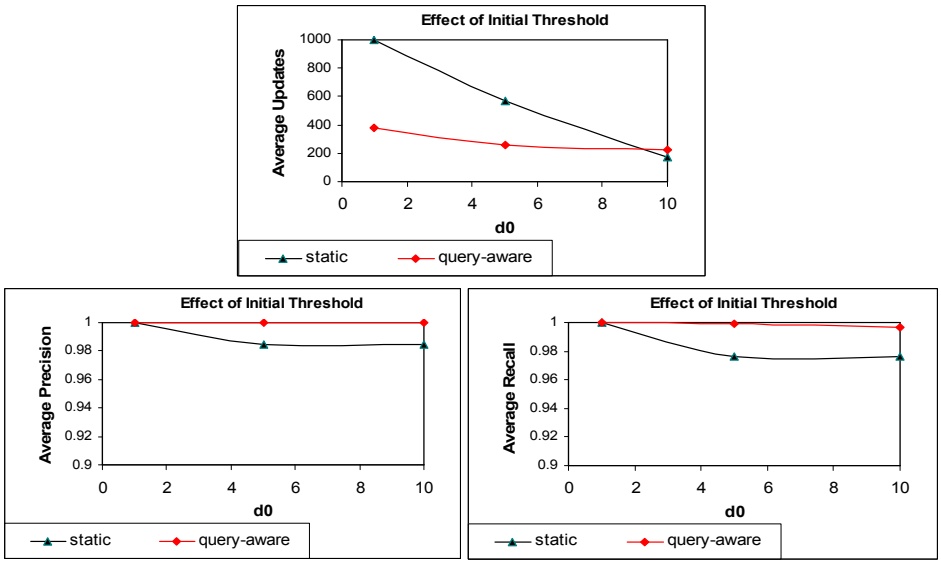


Figure 4.15: Range Query: Effect of Initial Threshold

threshold is larger, the total updates of both schemes drops. This is because larger setting gives moving objects larger free regions in which no updating message needs to be sent. Comparing the decreasing trend, it has been found that the static threshold scheme has a sharper drop. This implies that smaller initial threshold setting can benefit the static scheme more. However, Figure 4.15 also shows that larger initial threshold may make the query performance in the static threshold scheme become worse while the performance in the Aqua scheme remains the same.

4.5.2.4 Experiment #4: Effect of Query Rate

The fourth set of experiments shows the effect of query rate. Figure 4.16 presents the results. Whether the frequency of query issuing is high or low, the total number of updates remains the same for both schemes. This is because the updating activities are affected only by the movement factors. Figure 4.16 also shows that the query performance is scalable for both the static threshold and the Aqua scheme. We can see that Aqua has a better performance.

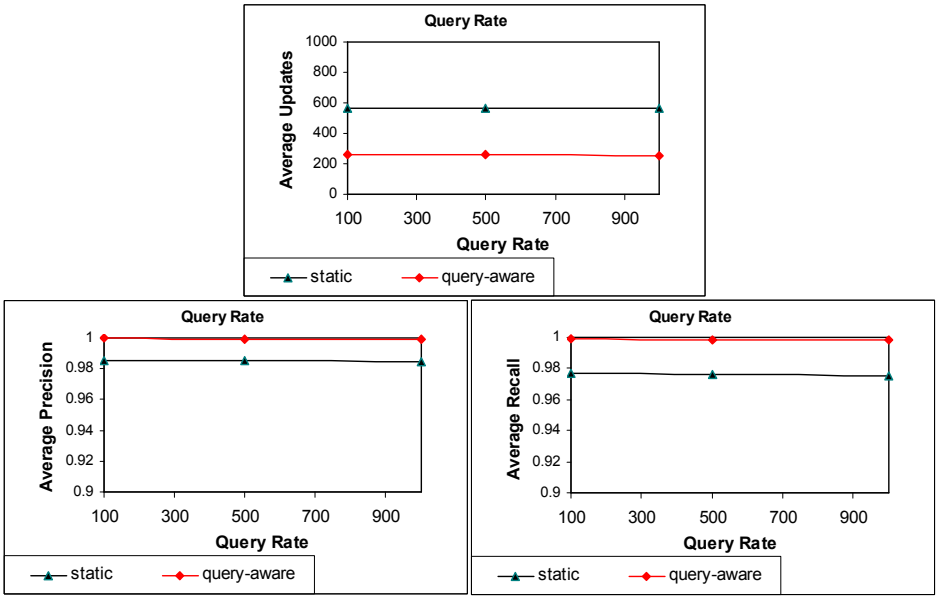


Figure 4.16: Range Query: Effect of Query Rate

4.5.2.5 Experiment #5: Effect of Standard Deviation of Query Distribution

Figure 4.17 presents results of the fifth set of experiments which examines the impacts of the standard deviation of the query distribution. The physical meaning of standard deviation is the skewness of query distribution. The larger the deviation is, the more skewed the queries are distributed. Figure 4.17 shows that Aqua has a better performance in updating activities when the query distribution is skewed. This is because the rationale of Aqua is to make use of the query pattern to reduce the updating messages. Considering query processing performance, Aqua can keep the performance level even when no skewed query distribution is available.

4.5.2.6 Experiment #6: Effect of Power Function Parameter

The last set of experiments examines the impact on the power function parameter κ. The results are presented in Figure 4.18. The impact that different κ brings in mainly lies in the updating activities. The query processing performance remains the same with the changes in κ regarding both precision and recall. Generally speaking, larger κ can lead to less total updating messages. This is because larger κ in the power

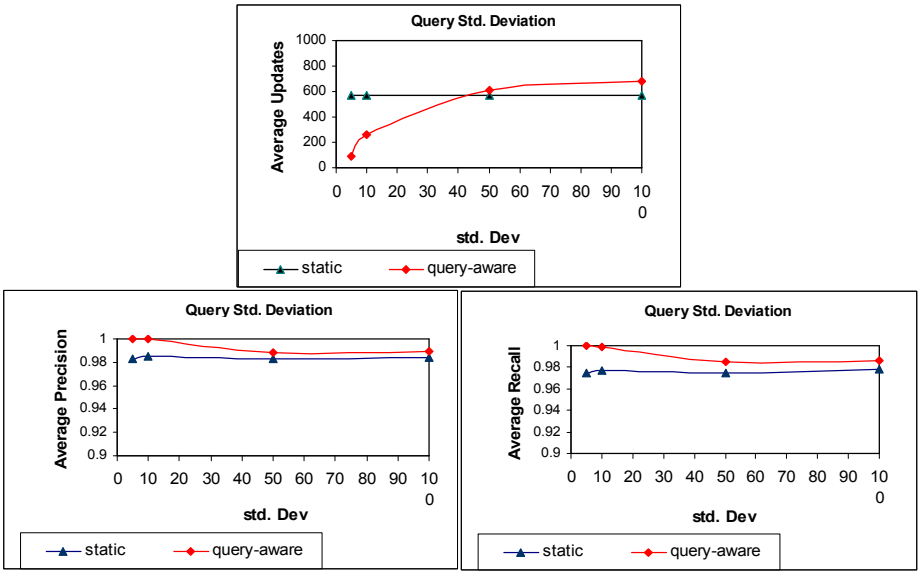


Figure 4.17: Range Query: Effect of Δ

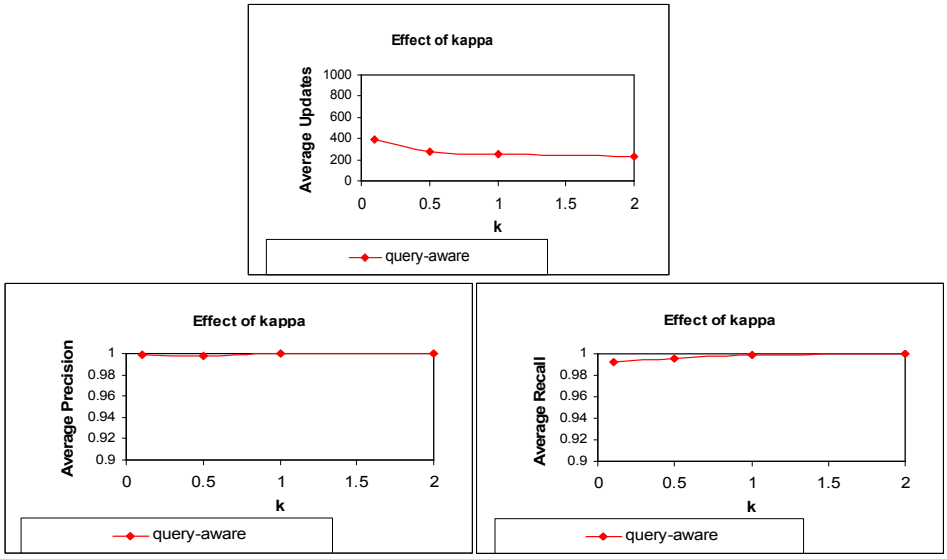


Figure 4.18: Range Query: Effect of κ

function means more skewed updating policy for different groups of objects. Those objects in infrequent queried groups have larger thresholds and issue fewer updating messages.

4.5.3 Confidence of Experimental Results

The experimental results are examined by computing the confidence intervals (CI) of each experiment point. 95% CI is examined and the standard error (i.e., s) is found out for computing the confidence intervals. The findings are summarized here. The magnitude of the experimental values which present the number of updates is from 100 to 1000 and the standard error (i.e., s) for these values is less than 10. Therefore, the 95% CI is roughly $[-20, 20]$. And the magnitude of the experimental values which present the precision and recall rate is around 0.1 and the magnitude of the standard error (i.e., s) for these values is around 0.001. Therefore, the 95% CI is roughly $[-0.002, 0.002]$.

4.6 Discussion

The background rationale of the query-aware model is to invest the precious resources to where it is needed most. It is believed that the need to answer queries is the most important reason that the objects keep their location information updated. As a result, frequent updating activity is only conducted by those "hot" objects frequently queried by the user.

The Aqua scheme is designed according to the query-aware model. Aqua takes querying activities into the consideration of object location updating design. The key design point is the adaptive safe region, (i.e. *quasar*). Prediction distance-based protocol is applied in *quasar* and the distance threshold is set based on different query workload.

Simulation studies have shown that the Aqua scheme has benefits in two aspects. It cannot only reduce updating resource consumption but also improve the query performance for the whole system.

However, as Aqua is the preliminary step towards the solution for the tradeoff problem introduced by the uncertainty, there are obvious drawbacks in the scheme design. Aqua cannot allow the user to control the quality of the query result. For example, if a user issues queries on some "cold objects" which are seldom asked by others, then the results returned may have fairly low precision according to Aqua's design principle. There is no mechanism provided by Aqua to improve the performance satisfaction for the user. Surely, the user can make use of the single probing methods after several trials. However, these methods are not involved in the main design consideration in Aqua. The goal of Aqua is to improve the updating and querying performance for the system as a whole. No precision guarantee is provided for a single query.

To remove the limitation and keep the query-aware benefits, the design of the cost-based models in the following chapters has been proposed. In the cost-based model, the probing methods have been provided and the total system performance in terms of both updating and querying activities has also been improved.

Chapter 5

Basic Cost-based Model

To remove the limitations of the Aqua scheme, this study proposes a cost-based model. As discussed in previous chapters, the tradeoff problem between location updating and query processing always exists and is the key problem for location management in mobile environments. Apparently, the best way to handle tradeoff is to optimize it and through the optimization procedure, it is natural to quantify tradeoff in the system as several costs consumed by updating and querying activities. In this chapter, an overview of the optimization approach in the cost-based model is presented. The detailed updating and querying models are then addressed. Based on the models, the basic cost-based scheme, CUP (Cost for Updating and query Processing) is presented, followed by an introduction of several adaptive optimization algorithms. The adaptive algorithms are designed based on the CUP scheme and are efficient tools for handling a dynamic environment and achieving cost optimization. Simulation studies are conducted to examine the system performance. Limitations of CUP are discussed and possible extensions are explored at the end of this chapter.

5.1 Overview of Cost-based Design

From the literature review and analysis of related research papers, it has been found that as uncertainty is an inherent problem in a moving object environment, most methods proposed in previous work for moving objects location management aim at bounding the location uncertainty and minimizing the update overhead. However, the tradeoff between these two factors always exists. Most work reported in the literature tries to adapt uncertainty to location update frequency. The approaches used in the field are either to trade the query precision for efficient resource utilization or to provide probabilistic answers based on acceptance of the uncertainty existence.

The major limitation of previous methods is that none of them consider any mobility patterns or query patterns that may be helpful to achieve the best system performance while accepting the existence of the irremovable tradeoff. As surveyed in previous sections, the previous models are not suitable for either movement optimization or query optimization.

Though both precise answers to queries and the lowest resource consumption cannot be achieved at the same time, efforts made to optimize the quantified tradeoff value are worthwhile.

5.1.1 Cost Optimization

The basic perspective for the cost-based approach is that the best way to handle the tradeoff problem when it is inherently introduced because of the two competitively leading factors is to optimize it. The first step towards the optimization is to quantify the abstract term, i.e., tradeoff. Intuitively, the cost-based approach is the most natural choice for the quantification procedure in moving object environment because the system performance here is measured by resource consumption which is eventually subject to some kinds of costs. Therefore, for a preliminary study of the optimization approach, a cost-based solution for tradeoff handling is developed.

First of all, the possible kinds of costs are categorized into two big groups. The following sections summarize the characteristics of these costs.

5.1.1.1 Costs for Updating Moving Objects

Costs for updating moving objects consist of three parts:

- **Costs in client side:** Most of the time, objects should send update reports to the server for location tracking purpose, and this updating activity carries a significant computational overhead for these handheld devices with limited battery power and short operational time.

- **Costs in server side:** The moving object environment is characterized by large volumes of location updates data from a large population of moving objects. Although some kinds of efficient spatial indexing have been employed [27, 81, 37], the overheads for handling huge location change data still exist and most of the cost is due to the repeated reporting of the changing locations of the moving objects.

- **Costs for communication between client and server:** Communication between the server and the client in the moving object environment is done through the wireless network which has limited uplink bandwidth for updating activities. A high volume of updates may overload the network and degrade the system performance. This would increase the cost of the application services too.

All the three types of costs introduced above can be probably measured by the number of updates, for example, issuing one update from a single object will have 1 unit cost.

5.1.1.2 Costs for Query Processing

Another type of costs should also be taken into consideration for a practical application or service: the penalty for an imprecise result of the query. We argue that the main purpose of tracking moving objects is to conduct query processing issued from the system user. An important measurement for system performance is the precision of query results returned to the user. As a result, costs exist if imprecise answers are generated.

Location management models are expected to handle the imprecise results which cause extra processing costs. In the eager-probing protocol, extra probing messages from the server to the moving object whose location information is not accurate enough as well as updating messages from these objects submitted to the server are generated. In the lazy-probing protocol, although no immediate probing is needed, further re-evaluation requirements may be issued when the imprecise results are far from the user' expectation. In both cases, the query processing costs can be measured by the number of extra probing and updating messages. Here an assumption is made that the computation costs consumed in the location server are eligible compared to the communication costs which make use of precious network resources.

5.1.1.3 Approach towards Optimization

Comparing all possible systems with the above costs, the system with the best performance is the one with the lowest costs. Here the best system performance considers not only the updating resource consumption but also the query performance as both factors are taken into account in the cost payment.

Based on the cost analysis above, a cost-based scheme is proposed to quantify the tradeoff. This scheme is termed CUP for optimizing the Cost for Updating and query Processing. Obviously, the CUP scheme strikes a balance between update cost and querying cost and tries to achieve the optimal cost consumption.

Making use of mathematical analysis, a process is conducted to prove that the CUP scheme can derive the optimal approximation setting for the updating frequency control with known system environment. One property of the optimal setting is that it should be an adaptive value for different objects (with different movement pattern) and for different query patterns. In a word, the solution should be an adaptive scheme which is both movement-aware and query-aware. The cost efficiency is also exactly based on the awareness of the individual object and their query properties. The CUP scheme can always find the optimal approximation whenever cost functions can be derived from the movement pattern. For the situation where unknown movements and query patterns are witnessed, adaptive optimization algorithms are proposed.

5.1.2 Comparison with Related Work

Before the detailed description about the proposed scheme given in the following sections, a comparison is made with similar previous work which also takes the optimization approach to handle the problem.

To the best of our knowledge, the most similar work which also takes the cost-based approach to deal with the tradeoff is from Wolfson et al [86]. In their work, an information cost model has been invented based on three separate costs, namely update cost, deviation cost and uncertainty cost. They have derived the optimal settings for the deviation threshold to achieve minimum information cost. The novelty and difference of the approach taken in this thesis compared with theirs can be analyzed as follows.

First of all, previous work considers querying and updating as two separated procedures for an application/system in the moving object environment. It is argued that the uncertainty and deviation have a cost or penalty in terms of incorrect decision making. Based on this argument, an information cost function integrates both update cost and the penalty for uncertainty. Previous work derives the minimal cost of a trip by assigning the optimal values to the object's deviation threshold. After achieving the optimal

information trip, the query issues are then considered. However, in most practical cases the only reason to provide up-to-date location information is to provide precise answers to queries concerning these objects. Therefore, if there are no queries issued for the whole trip of the moving object, it is not worth consuming the resources for updating the location information. A different approach has to be taken to conduct the cost-based model. Instead of counting deviation and uncertainty costs, the cost-based model investigated in this thesis uses another kind of cost mentioned previously, the query evaluation cost. In this way, both movement patterns and the query information are integrated in the cost model and the resulting system based on the cost model shows efficiency in the cost of both resource consumption and query precision.

Second, in previous work [86], the setting of the optimal deviation threshold value is based on a predicted deviation function. In these approaches, the deviation threshold at each update is adjusted to the current speed pattern. To derive the numerical setting, the changes on the current motion pattern are reflected by the parameter change on the predicted deviation function. Intuitively, a simple linear function $f(t)$ is given and the optimal cost is based on the deviation predicted by the function. In contrast, the cost-based model in this thesis can work without any predicted deviation function for optimal cost achievement.

Third, the previous work is only applicable when the destination and motion plan of the moving objects is known as a priori. In other words, it is assumed in the policies that the route is fixed and known to both the moving object and the server, and the update method is used to revise the time at which the moving object is expected to be at various locations on the fixed route. However, in most cases the destination or future route of a tracked object is not known. For example, the wireless service provider tracks every cellular phone, but it does not know their future route or destination. The user may simply not be willing to provide their private information to the tracking server, or this information may change too often to make the availability practical. The extended cost-based model introduced in Chapter 6 relaxes the movement assumption.

It does not require users to provide any motion information or makes assumptions about predefined routes.

5.2 Location Updating and Query Processing in Cost-based Model

Similar to the case in query-aware model, location updating and query processing models in cost-based model are based on the general location management model introduced in Chapter 3.

5.2.1 Location Updating

Let the set of moving objects be $\mathcal{O} = \{o_1, o_2, o_3, ...\}$ and the set of queries issued be $\mathcal{Q} = \{q_1, q_2, q_3, ...\}$. The basic distance-based protocol is applied here. A moving object, $o \in \mathcal{O}$, issues an update whenever the distance between the current object location, $p_i^{current}$, and the stored location, p_i^{stored}, exceeds the *threshold* d, i.e., $|p_i^{current} - p_i^{stored}| > d$. Upon each update, a tuple of the current location and the chosen threshold d_i for each object o_i is stored at the server.

The updating procedures of distance-based strategy running in both the moving object and the server part are depicted in Figure 5.1 and Figure 5.3.

Procedure for moving object o_i
1: monitor its own location p_i via devices such as GPS
2: if $|p_i^{current} - p_i^{reported}| > d_i$ then // move beyond the distance threshold
3: send update report to server S: $\langle o_i, p_i^{current} \rangle$
4: $p_i^{reported} \leftarrow p_i^{current}$
5: coordinate with S for its new threshold d_i
6: endif

Figure 5.1: The Procedure for Moving Object

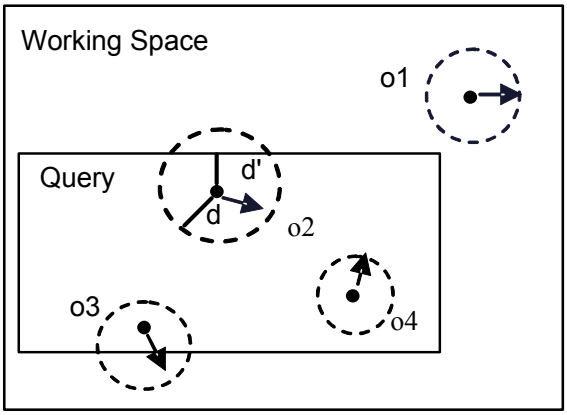


Figure 5.2: Querying Scenario

5.2.2 Query Processing

5.2.2.1 Query Analysis

Queries issued to the server are answered using stored object locations. The most basic query type is of the form: $q = \langle o_i, r_j \rangle$. This is a typical point query considering only one object, namely the object with identifier o_i and having precision requirement r_j. In the distance-based tracking strategy, r is specialized as a length value and it sets up the maximum distance that the reported location of o_i may deviate from the real position. Whatever the distance-based strategy is used, both the server and the moving object are aware of the distance threshold. The precision expectation of queries is subject to the needs of applications. This precision requirement is assumed to be provided at the time the queries are issued.

Let us suppose the threshold value is d. In general, to process q, if $d \leqslant r$, the stored location at the server satisfies the precision requirement and is returned to the query issuer immediately. If $d > r$, the stored location is inadequate in precision. The server needs to take different actions according to the querying protocol used.

Complex query types such as the nearest neighbor query and range query consider more than one object's location information. All these queries can be divided into several basic query types, each of which is related to one specific object only.

Take the range query as an example. As the location information retrieved is not the exact position of the objects but the position with maximum deviation (i.e. the threshold), the results returned to the query accordingly should consider not only the location stored but also the threshold the object may deviate from the location. In Figure 5.2, we take the basic distance-based tracking strategy and use a dotted circle to represent the area the object may be in (assuming it moves without any constraints) and the threshold is d for each object. Two sets of answers are obtained. One is the certain set. The objects in this answer set are the query results with 100% certainty. The other set is the possible set which includes all the objects that may be the results of the query such as o_2 and o_3. To make the results precise, the server needs to probe these object for its current location. For a particular object o_i which is in the possible set, the situation can be translated to a basic query $\langle o_i, r_j \rangle$ where r_j is the maximum threshold which can make the object be the certain set of the range query. Take o_2 as an example, its maximum threshold is d'. As its original threshold $d > d'$, further action is needed. In this way, the relationship between threshold and query precision requirement in complex query types is as the same as that in basic query type when distance-based strategy is used.

5.2.2.2 Querying Procedures for Different Protocols

As introduced in Chapter 3, query processing at the server can be run using two different protocols: eager-probing and lazy-probing protocols.

In eager-probing protocol, whenever the stored location is inadequate in precision, the server needs to probe the moving object o_i for its current location, $p_i^{current}$ immediately.

In lazy-probing protocol, no immediate probing is involved. Rather, processing errors involved in query results are reported to query issuer. The decision of whether to make efforts to obtain better query results depends on the user's further instructions.

Besides, the server revises their stored locations and negotiates a new threshold

with those probed objects. Collectively, the server part of both location updating and querying is depicted in Figure 5.3.

Procedure for server S
1: receive the next message
2: if it is an update report from o_i then
3: update the stored location of o_i with p_i
4: determine the new threshold d_i
5: endif
6: if it is a query $q_j = \langle o_i, r_j \rangle$ from a user then
7: if $d_i \leq r_j$ then
8: return the stored location to the query issuer
9: else if $d_i > r_j$ then
10: if eager-probing protocol is applied then
11: probe o_i for its exact current location
12: return the result to the query issuer
13: determine the new threshold value d_i
14: else if lazy-protocol is applied then
15: return the imprecise results to the query issuer
16: endif
17: endif
18: endif
19: piggyback the new threshold d_i to o_i

Figure 5.3: The Procedure for Server

5.3 CUP Scheme

The cost-based scheme CUP is presented in this section. CUP is a cost-based scheme to measure the ”cost” of updating and querying moving objects. It strikes a balance between the costs of updating versus query evaluation. The total system cost, $\mathcal{C}$, can be considered as a sum of the two component costs, C^U and C^Q, where C^U represents the cost consumed for object location updating activities and C^Q stands for the cost paid for query processing. Thus, $\mathcal{C} = C^U + C^Q$.

This definition of the cost function sets up the ultimate goal for the system performance: minimization of the total system cost. The key factors that are influential to each of the costs are identified and the appropriate settings which affect the design of the adaptive algorithms for optimizing the total cost $\mathcal{C}$ are derived.

5.3.1 Cost Analysis

5.3.1.1 Updating Cost Analysis

We assume that the tracking of moving objects is achieved by voluntarily issuing update reports from moving objects to the server. The cost for tracking mainly depends on the cost spent for updating activities. The analysis shows that there are three components in the updating cost: the object cost, the server cost and the communication cost.

To simplify the scenario, we fold the cost to the number of updates transmitted from moving objects to the server, since each component cost is roughly proportional to the update message count. Thus a measurable formula for the updating cost is defined as:

$$C_i^U = C_u \rho(d_i) \tag{5.1}$$

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

To define the formula for $\rho(d)$, we can analyze the object movement behavior. To simplify the situation, we assume that the moving object moves in a pre-defined route with the velocity v. It is the simplest movement case and real-life examples of this case include routes of airplanes and trains. In this case, the object movement is actually conducted in one-dimension.

With a known v, we can compute $\rho(d)$ easily. The time period for a moving object with v to go beyond the distance d is $T = \frac{d}{v}$. Then the update rate can be computed as $\rho(d) = \frac{1}{T} = \frac{v}{d}$.

5.3.1.2 Query Evaluation Cost Analysis

We assume that the eager-probing protocol is applied in the CUP scheme. In the eager-probing protocol, the query processing cost is caused when the object location at the server is imprecise and the server needs to send location probes to the object for its updated location immediately. This location probing activity introduces additional communication costs at the query time. To be specific, it includes one probe message and one object location update message. The query processing cost is attributed to the unsatisfactory distance threshold when queries with high precision constraint should be processed.

Assume there is an object o_i with the threshold d_i and a query $q_j = \langle o_i, r_j \rangle$. i is object identity, r is the maximum distance between the returned result and the real location that the query issuer can accept. A query is processed as follows. If $d \leq r$, the server returns the stored value to the query issuer. If $d > r$, the server should probe the queried object immediately for the accurate location information. Based on the above analysis, the measurable formulas are defined for the query evaluation costs.

The querying cost C_i^Q involving a specific object o_i to answer queries is defined as:

$$C_i^Q = C_q \phi(d_i) \tag{5.2}$$

where C_q is the unit cost of probing the moving object for the current location, which translates into the cost paid to probe one object and receive the reply. $\phi(d)$ is the query probe rate for moving object o_i, i.e., the number of probes generated per time unit, when the queries are not satisfied with an object o_i with threshold d, for which the query precision constraint is not fulfilled. Obviously, $\phi(d)$ depends on the query pattern, including the query arrival rate and the precision requirement. Intuitively, with a larger value of d, higher probing rate will be caused.

To define the function $\phi(d)$, a simplified case is assumed where queries arrive with rate λ, each being accompanied by a *precision constraint* sampled from a uniform dis-

tribution, $\mathcal{U}(0, r_{max})$. Then $\phi(d)$ is the number of queries issued per time unit multiplied by the probability that the precision constraint of the query is not satisfied. Thus, as long as $0 < d < r_{max}$, the probability that the precision constraint of the query is not satisfied can be computed as $Pr(r < d) = \frac{d}{r_{max}}$. If $d \geq r_{max}$, the probability will always be 1 which is the simplest case. The function $\phi(d)$ for an object with threshold d is evaluated as $\phi(d) = \frac{\lambda d}{r_{max}}$. Similar to the generalization process conducted in [57], the relationship between d and $\phi(d)$ is then defined as:

$$\phi(d) = \beta d \tag{5.3}$$

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

5.3.2 Total Cost Optimization

A complete general cost function for moving objects can be defined.

$$\mathcal{C} = C^U + C^Q = \frac{C_u v}{d} + C_q \beta d \tag{5.4}$$

Table 5.1 summarizes all parameters used in the function definitions and the following analysis.

With the cost function in hand, the optimal cost of the total system can then be obtained.

By differentiation analysis, the optimal value of d is $d^* = \sqrt{\frac{C_u v}{C_q \beta}}$. We can set the object's threshold to this optimal value to optimize the system performance.

In the above analysis, the update behavior and query workloads are assumed to be available in advance and then the optimal threshold of each object can be determined. For example, if the moving object follows a uniform motion, queries are generated with a stable arrival rate λ, and precision constraints are uniformly distributed within

Symbol	Description
$\mathcal{C}$, C^U, C^Q	Total, updating and querying cost
C_u, C_q	Unit cost of location update and query probe
d, d^*	threshold and its optimal value
$\rho(d)$, $\phi(d)$	Object update and query probe rate
v, β	Parameter due to movement or query behavior
δ	Cost ratio, defined as $\frac{\phi(d)}{\rho(d)}$
δ^*	Optimal cost ratio, $\frac{C_u}{C_q}$
ν	Rate of adaptation
ϵ	Ping-pong effect barrier parameter

Table 5.1: Symbols and Parameters

0 to r_{max}. We can know the exact value of v and derive β as $\beta = \frac{\lambda}{r_{max}}$. We can then easily achieve the optimal cost by setting the threshold to $d^* = \sqrt{\frac{C_u v r_{max}}{C_q \lambda}}$.

However, the system behavior is expected to be dynamic and change over time. A threshold that is good at one time may not become suitable at other times. To optimize the overall system performance, the optimal threshold needs to be adjusted adaptively. In the next section, the adaptive algorithms for the dynamic changing environment are proposed.

5.4 Adaptive Optimization Algorithms

With unknown system parameters and changing system conditions, the optimal threshold value cannot be directly obtained from the cost function. A feasible way to strive for the optimal system performance is to make use of adaptive optimization algorithms. In this section, several optimal threshold search algorithms are introduced. There are two types of algorithms, namely, the conjectural algorithm and the progressive algorithm.

The conjectural optimization algorithm "guesses" current system conditions. Based

on the guess, it directly determines the most possible optimal value. The progressive optimization algorithm starts at a certain threshold value and adjusts it gradually towards the optimal point. Both algorithms determine the threshold based on the current system states. Additional supportive rate monitoring algorithms are also discussed here.

5.4.1 Conjectural Algorithm

As its name suggests, this algorithm includes some degree of "guessing" when looking for the optimal value. The conjectural optimization finds directly the most probable threshold based on the monitored query arrival, object movement and system parameters at the run time. The basic idea is straightforward. Recall that the cost equation is $C = C_u\rho(d) + C_q\phi(d) = \frac{C_u v}{d} + C_q\beta d$. The minimal cost can be obtained when $d = d^* = \sqrt{\frac{C_u v}{C_q \beta}}$. In this function, C_u and C_q are system-dependent and predefined. Based on the value of v and β, we can achieve the optimal threshold d^* value. In order to get the current values of v and β, the conjectural algorithm makes a reasonable guess. Figure 5.4 presents the algorithm. It attempts to find out the values of v and β with the current system state by keeping track of the location update and query probe rates. The rate tracking is based on the rate monitoring algorithm which is to be discussed next. As $\rho(d) = \frac{v}{d}$ and $\phi(d) = \beta d$, we can guess the current values of v and β whenever we know the value of ρ, ϕ and the current value of d.

5.4.1.1 Rate Monitoring Algorithms

To provide the expected rates of the location update and location probe (both of them are considered as events), three **rate monitoring algorithms** which adopt different statistical schemes are proposed. They are the MEAN scheme, the WINDOW scheme and the EWMA (i.e., Exponentially Weighted Moving Average) scheme.

MEAN: The rate of an event is computed to be the average for the time being. In this

Conjectural Algorithm
1: initialize d, ρ and ϕ
2: upon receiving a location update or server probe do
3: if a location update is received then
4: update ρ using Rate Monitoring Algorithm
5: else
6: update ϕ using Rate Monitoring Algorithm
7: endif
8: $v \leftarrow \rho d$
9: $\beta \leftarrow \frac{\phi}{d}$
10: $d^* \leftarrow \sqrt{\frac{C_u v}{C_q \beta}}$
11: $d \leftarrow d^*$
12: endo

Figure 5.4: Conjectural Optimization Algorithm

scheme, the number of events is counted. Whenever the rate is needed, it is computed as the total number of events divided by the total time elapsed. $MeanRate = \frac{NU}{TT}$ where NU stands for the total number of events and TT means the total time. The rate monitoring algorithm using the MEAN scheme is shown in Figure 5.5.

MEAN
1: initialize TT, $MeanRate$, set $NU = 0$, $initialTime = NOW$
2: do while the rate computation is needed
3: if the server receives an event then
4: $NU \leftarrow NU + 1$
5: $TT \leftarrow NOW - initialTime$
6: $MeanRate \leftarrow \frac{NU}{TT}$
7: endif
8: endo

Figure 5.5: Rate Monitoring Algorithm: MEAN

WINDOW: The rate of an event computation is affected by only W most recent time slots. Here one time slot is defined as a fixed number of time units. The rate of an event in one time slot is computed using the MEAN method. The whole event rate is the average of rates of W time slots. $WinRate = \frac{1}{w}\sum_{j=c-w}^{c} Rt_j = \frac{1}{w}\sum_{j=c-w}^{c} \frac{NU_j}{TS_j}$, where w is the window size and c is the current time slot number. Rt_j stands for the rate in time slot j. NU_j and TS_j means the total number of events and the total time for the time slot j. The rate monitoring algorithm using the WINDOW scheme is shown in Figure 5.6.

WINDOW

1: initialize array $Rt[1...w]$, BT, TS, $WinRate$, set $c = 1$, $index = 1$
// BT is the beginning time for the current time slot. TS is the number of time units of one time slot. w is the window size.
2: do while the rate computation is needed
3: $BT \leftarrow NOW$
4: $NU \leftarrow 0$
5: do while $NOW - BT < TS$
6: upon receiving an event do
7: $NU \leftarrow NU + 1$
8: endo
9: endo
10: $Rt[index] \leftarrow \frac{NU}{TS}$
11: $c \leftarrow c + 1$
12: $index \leftarrow index + 1$
13: if $index > w$ then
14: $index \leftarrow 1$
15: endif
16: $WinRate \leftarrow \frac{1}{w} \sum_{j=1}^{w} Rt[j]$
17: endo

Figure 5.6: Rate Monitoring Algorithm: WINDOW

EWMA: EWMA assigns exponentially decreasing weights to the rate computation in previous time slots so that the most current time slot receives higher weights. With an adjustable parameter ω ($0 < \omega \leq 1$), the most recent time slot will receive a weight of ω and the next one a weight of ω^2, and so on. With a medium to low ω, the weights tail off quickly. $EWMARate = \frac{1}{\sum_{j=1}^{c} \omega^{c-j}} \sum_{j=1}^{c} \omega^{c-j} Rt_j = \frac{1-\omega}{1-\omega^c} \sum_{j=1}^{c} \omega^{c-j} Rt_j = \frac{1-\omega}{1-\omega^c} \sum_{j=1}^{c} \omega^{c-j} \frac{NU_j}{TS_j}$. c is the current time slot number. Rt_j is the rate of time slot j. As $c \rightarrow \infty$, EWMA rate at current time slot c can be computed incrementally as $EWMARate_{(c)} = \omega EWMARate_{(c-1)} + (1 - \omega) Rt_c$. The rate monitoring algorithm using the EWMA scheme is shown in Figure 5.7.

5.4.2 Progressive Algorithm

The main idea of the progressive algorithm is adjusting the threshold every time when a location update or a query occurs. When a location update occurs, it is hinted that the threshold can be set larger than the current one. On the other hand, when a pag-

EWMA
1: initialize ω, BT, TS, set $EWMARate = 0$
2: do while the rate computation is needed
3: $BT \leftarrow NOW$
4: reset Rt, set $NU = 0$
5: do while $NOW - BT < TS$
6: upon receiving an event do
7: $NU \leftarrow NU + 1$
8: endo
9: endo
10: $Rt \leftarrow \frac{NU}{TS}$
11: $EWMARate \leftarrow \omega EWMARate + (1 - \omega)Rt$
12: endo

Figure 5.7: Rate Monitoring Algorithm: EWMA

ing message is incurred, it is implied that the current threshold is larger than what the query expects, the threshold is thus reduced. We then need to determine by what amount should d be adjusted. Too large a jump would make the algorithm over adaptive and make d sensitive to minor changes of the system parameter. This is called ping-pong effect. Too small a shift will make it a lengthy process to adapt to a new object movement and query pattern.

To determine the adjustment, we look at the system property at the optimal performance point and maneuver the adjustment based on the existing deviation from the optimal property. We observe that the ratio between $\phi(d)$ and $\rho(d)$ is a constant at the optimal threshold d^*. When $\delta = \frac{\phi(d)}{\rho(d)} = \frac{C_u}{C_q}$, $d = d^* = \sqrt{\frac{C_u v}{C_q \beta}}$. Thus, $\delta^* = \frac{C_u}{C_q}$, and the optimal threshold d^* is obtained when $\delta = \delta^*$. The problem is then reduced to adjust d so that $\frac{\delta}{\delta^*} = 1$. The value of d is adjusted by an amount of ν, a nonnegative tunable parameter.

Algorithm 5.8 and 5.9 present two adaptive progressive algorithms for setting d at the server, namely, History-tracking Algorithm (HA) and Non-History-tracking Algorithm (NHA). The meaning of the symbols used can be found in Table 5.1. Both algorithms attempt to adjust the system parameter in order to make the ratio of the observed query probe rate and location update rate equal to δ^*.

HA at Server
1: initialize d, ρ and ϕ, set $\delta^* = C_u/C_q$
2: upon receiving a location update or server probe activity do
3: if a location update is received then
4: update ρ using Rate Monitoring Algorithm
5: else
6: update ϕ using Rate Monitoring Algorithm
7: endif
8: $\delta \leftarrow \frac{\phi}{\rho}$
9: if $\delta/\delta^* > 1 + \epsilon$ then
10: $d \leftarrow d/(1 + \nu)$
11: else if $\delta/\delta^* < 1 - \epsilon$ then
12: $d \leftarrow d(1 + \nu)$
13: endif
14: endo

Figure 5.8: HA at Server

History-tracking Algorithm (HA) keeps track of the location update and query evaluation rates by adjusting d, in order to compute δ and make the ratio δ/δ^* close to 1. Since larger d leads to higher query evaluation rate and lower location update rate and hence larger δ, HA decreases d when the ratio δ/δ^* is larger than the value in the optimal condition, i.e., 1. To avoid the ping-pong effect, a change is initiated only when the ratio is beyond a certain threshold ϵ from the target value of 1.

NHA at Server
1: initialize d, set $\delta^* = C_u/C_q$
2: if an object location update is received then
3: $d \leftarrow d(1 + \nu)$ with probability $min\{\delta^*, 1\}$
4: endif
5: if a server query evaluation activity is needed for an object then
6: $d \leftarrow d/(1 + \nu)$ with probability $min\{\frac{1}{\delta^*}, 1\}$
7: endif

Figure 5.9: NHA at Server

Non-History-tracking Algorithm (NHA) only makes use of the local property of the location update and query probe activity observed at the server under normal operations, without attempting to track for the object movement pattern, query pattern and query precision, nor storing their histories. Similar to HA, NHA attempts to make the ratio δ/δ^* close to 1, but by means of the probabilistic approach. To explain the algo-

rithm, let us examine the simplest case when $\delta^* = 1$. To make the ratio $\delta/\delta^* = 1$, the system should keep $\delta = 1$, i.e., the query probe rate should be equal to the location update rate. To balance the likelihood of the two types of activities, NHA would decrease d on a query probe or occurrence of the infinitive query cost and increase d on an object update in order to reach for the optimal setting. If $\delta^* > 1$, a larger d which leads to a higher query evaluation rate and smaller object update rate is preferred. Therefore, NHA would still increase r on a location update but would just decrease d with a probability $1/\delta^*$ on a query evaluation activity. Conversely, if $\delta^* < 1$, a smaller d is preferred, and NHA does not increase d on every location update.

5.5 Simulation Studies

Parameter	Value range	Default
λ	1 - 10 per time unit	1
V_{max}	1, 5, 10	-
r_{max}	10-100	50

Table 5.2: Simulation Parameter Setting

In this section, simulation experiments are conducted to evaluate the proposed adaptive optimal threshold searching algorithms. In each experiment, the running time of the movement of moving objects is 1000 time units. The service area is a square-shaped region of size 1000 by 1000 units. The number of objects simulated is 10000.

The objects movement is simulated as the scheduling of the airlines. 100 points are randomly picked up in the whole working space as 100 blocks which represent 100 cities. Every moving object chooses two blocks as its start city and destination city. Its flying schedule is either from the start city to the destination city or from the destination city back to the start one. Every time it starts flying, it chooses a maximum

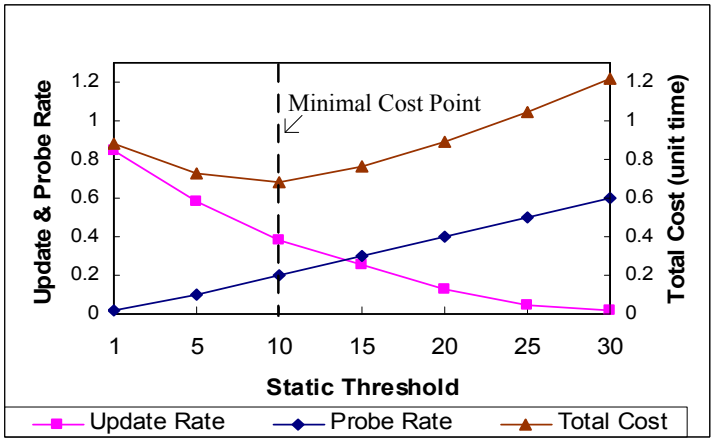


Figure 5.10: Update/probe Rate and Cost

speed from the set [1, 5, 10]. Through the first $1/10$ flying route, the moving object speeds up to its maximum speed and through the last $1/10$ route, it slows down its velocity to 0. During the route, it flies at the steady speed.

The queries arrive with rate λ, each being accompanied by a *precision constraint* sampled from a uniform distribution, $\mathcal{U}(0, r_{max})$. An assumption is made that the location update cost per object is 1 and the server probe cost is 2, with one server paging message and one object reporting message. Thus, $\delta^* = \frac{C_u}{C_q}$ would be 0.5. Note that the actual value of δ^* depends on the semantics of the applications and thus it may vary. However, its choice has no particular impact on the algorithm performance, as long as the value can be estimated accurately. Table 6.1 summarizes the parameters used in all experiments.

5.5.1 Experiment #1: Assumption Validity

This set of simulation experiments is conducted to establish the correctness of the assumptions of the relationship among $\phi(d)$, $\rho(d)$ and d, i.e., to show that generally $\rho(d)$ and $\phi(d)$ are proportional to $1/d$ and d respectively. The simulation set is also conducted to show when $\phi(d)/\rho(d) = \delta^*$, the total system cost is minimized.

A query workload with the query arrival rate of 1 and a maximum precision con-

straint of 50 is simulated. The updating and querying algorithms are conducted with a fixed d for each experiment (i.e., d is not adjusted adaptively according to different system situations), but varying d across experiments. The average number of updates and probes per time unit are measured and the results are reported in Figure 5.10. The measured values for $\phi(d)$ and $\rho(d)$ are found to be proportional to $1/d$ and d respectively. The figure verifies that the minimal total cost can indeed be attained when $\phi(d)/\rho(d) = \delta^* = 0.5$,

5.5.2 Experiment #2: Algorithm Correctness

This set of simulation experiments is conducted to demonstrate that the proposed algorithms indeed achieve the optimal performance with appropriate parameter settings. That is to validate the claim that the adaptive algorithms can adapt the threshold d towards the optimal value d^*, i.e., they converge. Two subsets are experimented.

5.5.2.1 Conjectural Algorithm

The first subset is to examine the conjectural algorithm. Figure 5.11 shows the results of running the conjectural algorithm with different rate monitoring algorithms. X-axis shows various initial thresholds. It is clear that the initial threshold setting has little impact on the algorithm performance.

Generally speaking, the curves with different rate monitoring methods shape similarly in this stable situation with unchanged movement and query patterns. This is not a surprising finding because the similar behavior of MEAN, WINDOW and EWMA schemes are expected for rate monitoring. Comparing the performance of different rate monitoring algorithms, it has been found that the WINDOW algorithm cannot achieve good performance if the window size is too large or too small. The best performance can be achieved by applying EWMA or WINDOW with the appropriate window size.

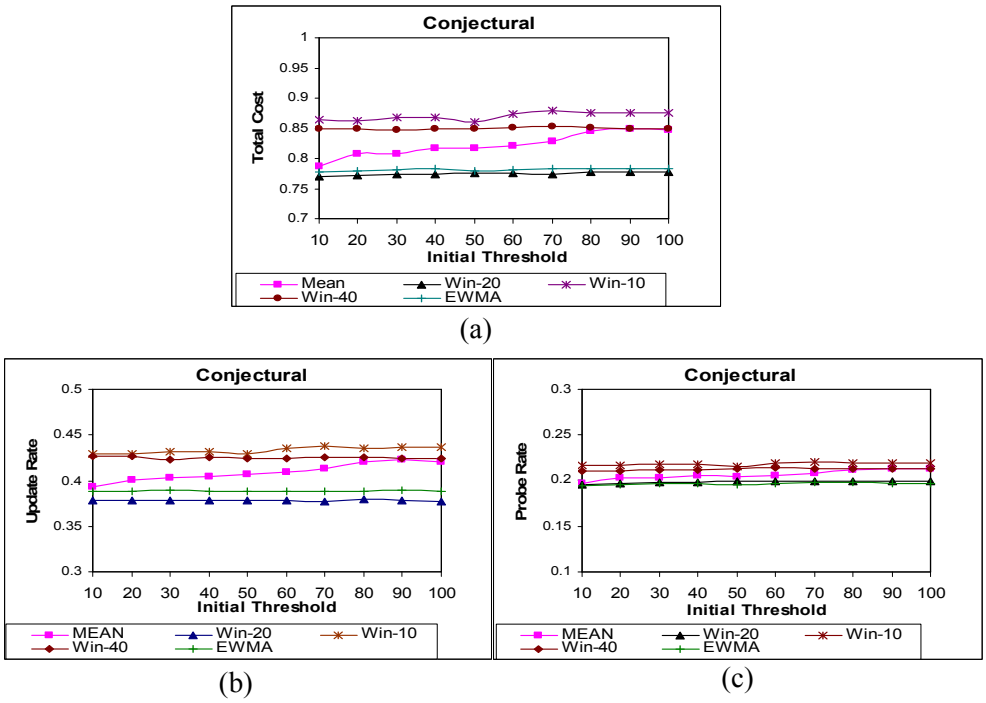


Figure 5.11: Conjectural with Rate Monitoring

	Conjectural	Progressive
ν	no need	0.05
d_0	no impact	no impact
d^*	9.53	10.25
cost $\mathcal{C}$	0.79	0.66

Table 5.3: Optimal Setting for Parameters

5.5.2.2 Progressive Algorithm

The progressive algorithm and its parameter setting are then examined. The progressive algorithm controls the magnitude of the threshold adjustment by means of ν, an important tunable parameter. ν as well as the initial threshold d_0 are varied in the experiment. After conducting numerous experiments, the optimal setting can be deduced for the tunable parameter for the best performance.

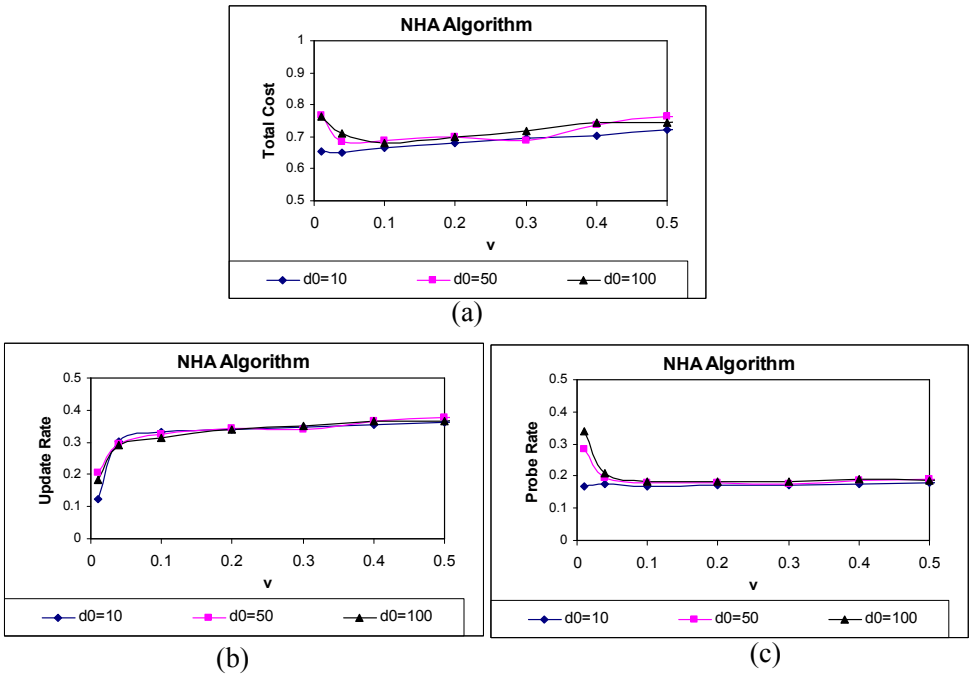


Figure 5.12: Results with NHA Algorithm

There are two types of progressive algorithms, namely NHA and HA. Figure 5.12 presents results of using NHA with different initial d and with ν as its x-axis. The rate monitoring algorithm used here is the MEAN method. The results of other two schemes are very similar. It is clear that this algorithm is unable to yield a best performance when ν is either too large or too small. This is because a large ν induces a strong fluctuation to the adjustment of d, thus missing the optimal point. Too small a value of ν will lead to a very slow convergence for the system to adapt to the optimal d. Figure 5.12 also reveals that unless ν is too extreme, the initial threshold d brings little impact on the performance.

Figure 5.13 presents the results of using HA. The performance of various d_0 and ν is similar to that in the simulation using NHA. Based on these two sets of simulations, we can safely conclude that the initial threshold has little impact on the performance and we need to choose ν carefully for good algorithm performance in both progressive algorithms.

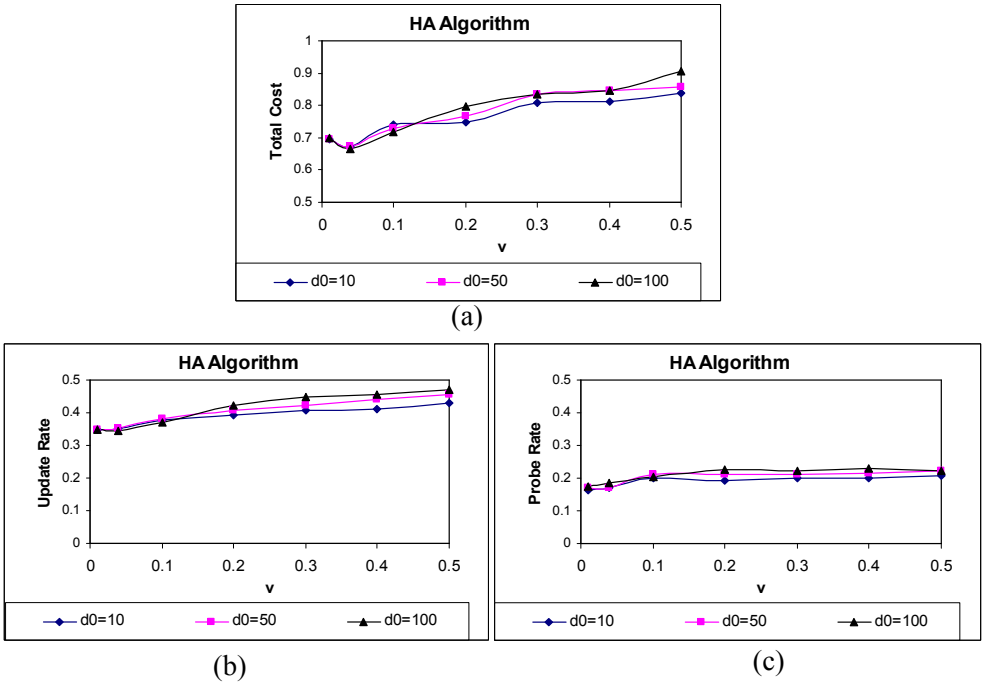


Figure 5.13: Results with HA Algorithm

Table 5.3 summarizes the best setting for the adjustable parameters and the best threshold d^* returned by the algorithms. Compared with the results in Figure 5.10, the best threshold returned from both the conjectural and progressive algorithm are close to the optimal, with at most 5% difference. The results also indicate that progressive algorithms perform better in exerting a lower total system cost.

5.5.3 Experiment #3: Query Patterns

The third experiment evaluates the effect of the query pattern. The main focus is on evaluating the effect on the query arrival rate and various maximum query precision r_{max}. The results are depicted in Figure 5.14 and Figure 5.15.

All algorithms result in higher system cost at a higher query rate and smaller maximum precision constraint. This is expected due to a higher degree of query activities. Examining the figures in detail, it can be found that the conjectural algorithm yields the highest system cost. HA performs a little better with a lower total cost. Analysis

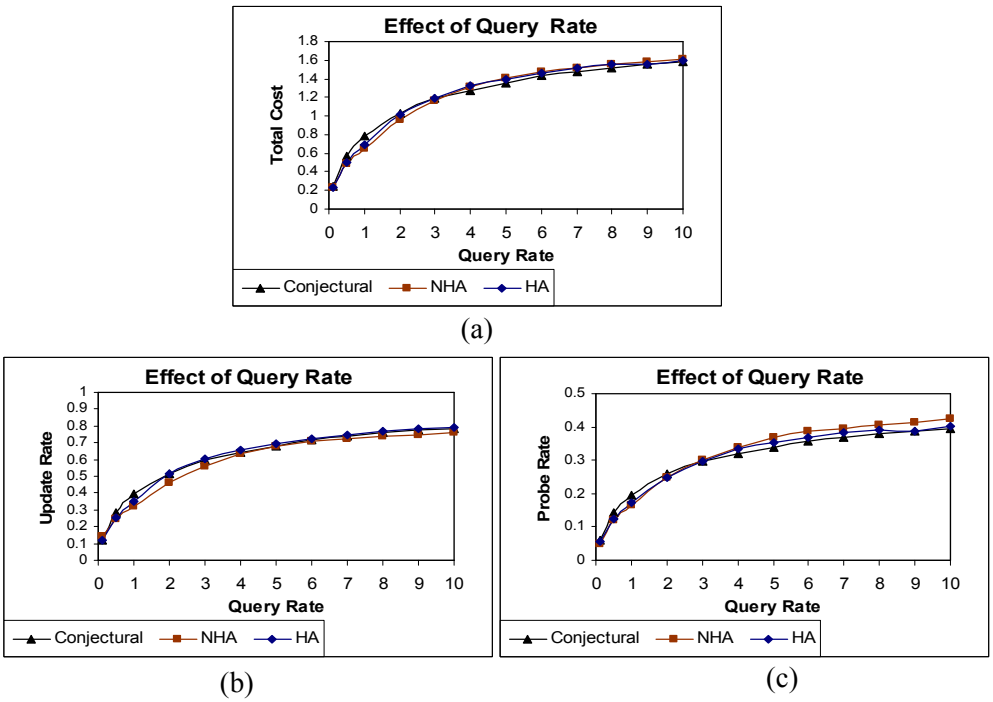


Figure 5.14: Effect of Query Rate

shows that the better performance is due to its stronger ability to adapt to changes, compared with NHA. There is a performance tradeoff between the two progressive algorithms. NHA requires less computational power at the server because no history tacking is needed while HA can adapt to the changing situation more quickly, yielding a better performance at the expense of higher computational cost.

5.5.4 Confidence of Experimental Results

The confidence intervals of all the experiments are summarized here. It is found that the magnitude of the experimental values which present the total cost is around 0.1 and the magnitude of the standard error (s) for these values is around 0.001. From the literature, a 95% CI is sufficient for results testing. The CI values for all sets of experiments in this chapter can be summarized as: $[-0.00196, 0.00196]$. This interval is relatively insignificant compared to the experimental results of the system performance.

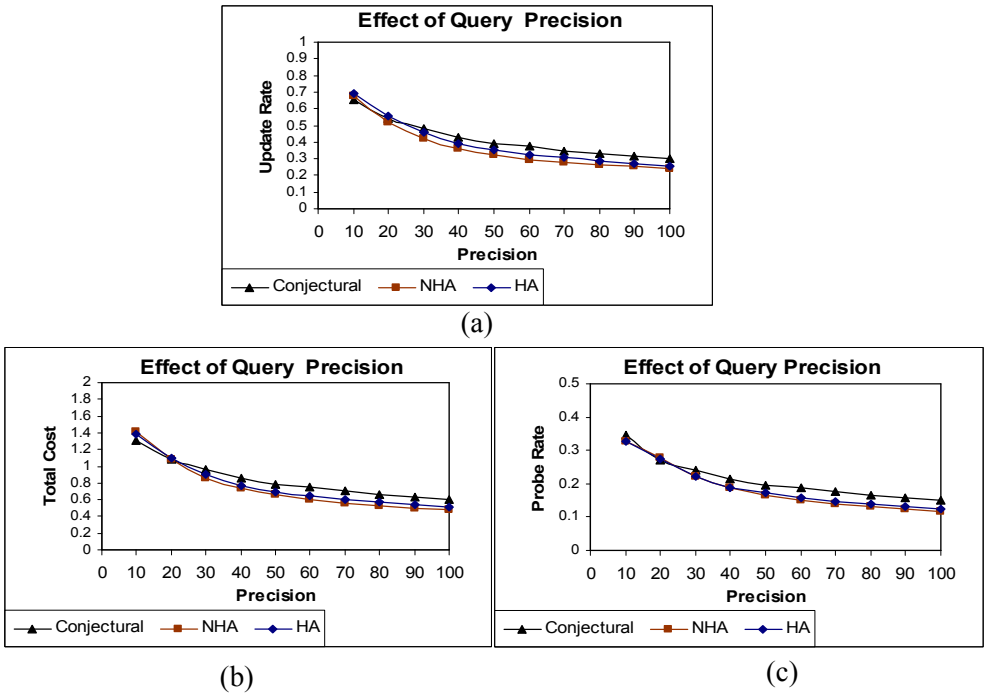


Figure 5.15: Effect of Query Maximum Precision

5.6 Discussion

The CUP scheme is a concrete design of the cost-based model. Its main purpose is to minimize system costs and derive the optimal distance threshold for both updating and querying activities. In the analysis and simulation studies, we can find that the cost-based scheme is effective for handling tradeoff problem in the term of balancing two conflicting factors.

During the analysis and design, some assumptions on the movement of the object and the query patterns are made. These assumptions simplify the complex scenario and make analysis procedure easy to be handled. However, these assumptions may not always be true. To make the CUP scheme work in more general scenarios, further extensions are needed.

In the next chapter, the assumptions are relaxed on the movement pattern and the lazy-probing protocol scenario is discussed for the CUP scheme. Both extensions make the problem more complex and difficult to be handled.

Chapter 6

Extended Cost-based Model

This chapter extends the basic cost-based model introduced in Chapter 5. Two main types of costs are considered in the cost-based model. To simplify the definition of the cost function, assumptions are made. In this chapter, relaxations to the assumptions and extensions to the basic model are conducted for the movement in the first section and for the query pattern in the following one. As a result of relaxation and extension, the cost function as well as the adaptive optimization algorithms should be reviewed. Simulation studies are also conducted to evaluate the performance of the new algorithms.

6.1 Relaxation of Movement Assumption

In Chapter 5, it has been assumed that a moving object follows a pre-defined route and its movement actually is 1-dimensional. In reality, most moving objects such as public buses, taxis and walking persons usually conduct 2-dimensional movements. In this section, revision of the cost-based model for 2-dimension scenario is conducted. Note that besides 1-dimensional and 2-dimensional movements, there is so called 1.5-dimensional movement which describes the movements of those objects who are restricted by pre-defined constraints. For example, buses can only move along the city

streets. As 1.5-dimensional movement is actually a restricted 2-dimensional movement, the cost model and related algorithms proposed for 2-dimensional movement is applicable for 1.5-dimensional movement.

6.1.1 Updating Cost: A Review

The key change to relax the movement assumption is the need to re-define the updating cost function, i.e. $C^U = C_u\rho(d)$. The change is due to the different computation of object updating rate $\rho(d)$. To define the formula for $\rho(d)$, we need to analyze the movement behavior of the object when a 2-dimensional movement is conducted.

Assume that moving objects obey a 2-dimensional random walk model. All the objects move in steps and in each step, each object travels a distance of k along an arbitrary direction. Each step takes a duration of L time units.

Lemma 1 *If the movement of an object o_i follows the random walk model, each movement step lasting for a period of time L_i and the distance moved in each step being k_i, then the rate at which o_i moves beyond a distance d is $\frac{k_i^2}{(L_i d_i^2)}$.*

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

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

The drunken person problem is basically a random walk problem which addresses a mathematical formalization of a trajectory that consists of taking successive steps in random directions [55, 42, 43]. The movement of an object is similar to that of the drunken person. The lemma can be proven with the result from the Drunken Person Problem. For an object o_i to update its location, it should move at least a distance of d beyond its latest reported location, corresponding to the starting point in the Drunken

Person Problem. The normalized distance from the starting point to the furthest point the object can reach is $\frac{d_i}{k_i}$. Let T be the expected time that o_i moves beyond d. It is obvious that $\rho(d_i) = \frac{1}{T}$. Suppose at t_0, o_i is located at the latest reported location and at time $t_0 + T$, it is expected to move to a furthest point it can reach. The distance moved between this time period is $\zeta_1 = \frac{d_i}{k_i}$ and the number steps that o_i moves is $n = \frac{T}{L_i}$. Thus, the distance between the starting point and the current point is $\zeta_2 = \sqrt{n} = \sqrt{\frac{T}{L_i}} = \sqrt{\frac{1}{L_i \rho(d_i)}}$. Since $\zeta_1 = \zeta_2$, we have $\frac{d_i}{k_i} = \sqrt{\frac{1}{L_i \rho(d_i)}}$. Thus, $\rho(d_i) = \frac{k_i^2}{L_i d_i^2}$. □

Similar to the generalization method used in [57], the relationship between $\rho(d)$ and d can be generalized as:

$$\rho(d) = \frac{\alpha}{d^2} \tag{6.1}$$

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

6.1.2 Cost Optimization and Adaptive Algorithms

With the revision of updating cost, the total cost function is re-defined as:

$$\mathcal{C} = C^U + C^Q = \frac{C_u \alpha}{d^2} + C_q \beta d \tag{6.2}$$

Although the cost function varies, the optimization procedure and adaptive algorithms proposed in Chapter 5 can be applied easily to the new case.

By differentiation analysis, the optimal value for d is $d^* = \sqrt[3]{\frac{2C_u\alpha}{C_q\beta}}$. If the update behavior and query workloads are available in advance, the optimal threshold for every object can be determined.

In the case of unknown update behavior and query workloads, we need the adaptive algorithms for performance optimization. Similar to the case in the basic cost-based model and based on the cost function, the conjectural and progressive algorithms proposed in the previous chapter can also be reviewed and applied to the new situation.

A different optimal threshold value is computed in the conjectural algorithm. In the progressive algorithm, the change is in the optimal δ value. We observe that the ratio between $\phi(d)$ and $\rho(d)$ is a different constant at the optimal threshold d^*. When $\delta = \frac{\phi(d)}{\rho(d)} = \frac{2C_u}{C_q}$, $d = d^* = \sqrt[3]{\frac{2C_u\alpha}{C_q\beta}}$. We thus define $\delta^* = \frac{2C_u}{C_q}$, and the optimal threshold d^* is obtained when $\delta = \delta^*$.

The changes of algorithms are shown in Figure 6.1, Figure 6.2 and Figure 6.3. Boxes are used to highlight the differences between the previous algorithms and the revised ones.

Conjectural Algorithm
1: initialize d, ρ and ϕ
2: upon receiving a location update or server probe do
3: if a location update is received then
4: update ρ using Rate Monitoring Algorithm
5: else
6: update ϕ using Rate Monitoring Algorithm
7: endif
8: $\boxed{\alpha \leftarrow \rho d^2}$
9: $\beta \leftarrow \frac{\phi}{d}$
10: $\boxed{d^* \leftarrow \sqrt[3]{\frac{2C_u\alpha}{C_q\beta}}}$
11: $d \leftarrow d^*$
12: endo

Figure 6.1: Updating Cost Review: Conjectural Algorithm

6.1.3 Simulation Studies

In this section, simulation studies are conducted to evaluate the performance of the adaptive algorithms in the new movement scenario. Each experiment models the movement of the moving objects for 1000 time units. The spatial domain of interest is a square-shaped region of size 1000 by 1000. Two mobility models are used in the simulation, namely, *Random Walk* and *Random Waypoint*. Both are well-known for performance evaluation of moving object management systems. The random walk

HA at Server

initialize d, ρ and ϕ, set $\boxed{\delta^* = \frac{2C_u}{C_q}}$
upon receiving a location update or server probe activity do
  if a location update is received then
    update ρ using Rate Monitoring Algorithm
  else
    update ϕ using Rate Monitoring Algorithm
  endif
  $\delta \leftarrow \frac{\phi}{\rho}$
  if $\frac{\delta}{\delta^*} > 1 + \epsilon$ then
    $d \leftarrow \frac{d}{1+\nu}$
  else if $\frac{\delta}{\delta^*} < 1 - \epsilon$ then
    $d \leftarrow d(1 + \nu)$
  endif
endo

Figure 6.2: Updating Cost Review: HA at Server

NHA at Server

initialize d, set $\boxed{\delta^* = \frac{2C_u}{C_q}}$
if an object location update is received then
  $d \leftarrow d(1 + \nu)$ with probability $min\{\delta^*, 1\}$
endif
if a server query evaluation activity is needed for an object then
  $d \leftarrow \frac{d}{1+\nu}$ with probability $min\{\frac{1}{\delta^*}, 1\}$
endif

Figure 6.3: Updating Cost Review: NHA at Server

model is suitable to simulate small-scale scenarios, while the random waypoint model fits large-scale on-purpose movements better. In random walk, all objects move in steps and each moves a distance of k along an arbitrary direction at each step, with a duration of L. In random waypoint, each object chooses a random point in the space as its destination and moves to it at a speed randomly selected from the range [0, V_{max}]; upon arrival or expiration of a constant movement period randomly picked from the range [0, T_{max}], it chooses a new destination and repeats the same process. Queries arrive with rate λ and maximum precision constraint r_{max}. We assume that the location update cost per object is 1 and the server probe cost is 2, with one server paging message and one object reporting message. Thus, $\delta^* = 2C_u/C_q$ would be 1. Note that

the actual value of δ^* depends on the semantics of the particular application system and may vary. Table 6.1 summarizes the parameters used in all experiments.

Parameter	Value range	Default
L (in second)	1	1
k (in meter)	1 (slow), 5 (moderate), 15 (fast)	5
λ	1 - 10 s^{-1}	1
V_{max}	15	15
T_{max}	2	2
r_{max}	10-100	50
δ^*	1	1

Table 6.1: Simulation Parameter Setting

6.1.3.1 Experiment #1: Assumption Validity

A simulation is run to establish the correctness of the assumption for the relationship among d and $\rho(d)$ and $\phi(d)$, i.e., to show that generally $\rho(d)$ and $\phi(d)$ are proportional to $1/d^2$ and d respectively. The simulation also aims to show that when $\frac{\phi(d)}{rho(d)} = \delta^*$, the total system cost is minimized. Both movement models are used for the evaluation. For random walk, we consider a set of moving objects with moderate speed. For random waypoint, we take the maximum velocity to be 15 and maximum expiration time 2 time units. A querying workload with query arrival rate of 1 and a maximum precision constraint of 50 is used in the simulation.

We run the updating and querying algorithm with a fixed d for each experiment (i.e., we do not adjust d adaptively according to different system situation), but vary d across experiments. The average number of updates and probes per time unit are measured and the results are reported in Figure 6.4. The measured values for $\rho(d)$ and $\phi(d)$ are found to be proportional to $1/d^2$ and d respectively. From the figure for random walk and random waypoint models, we can verify that the minimal total cost

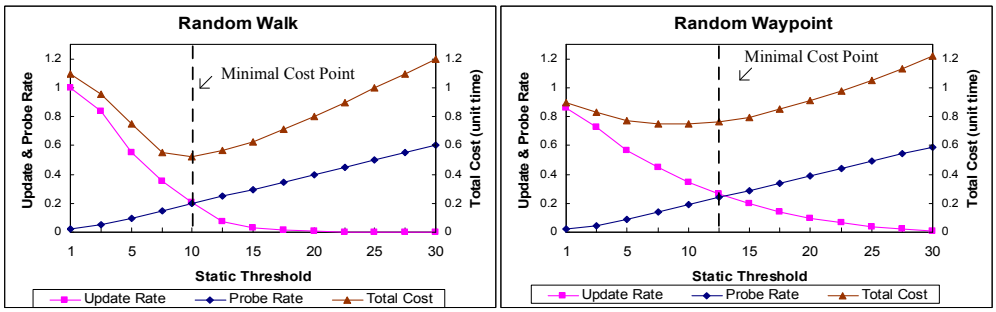


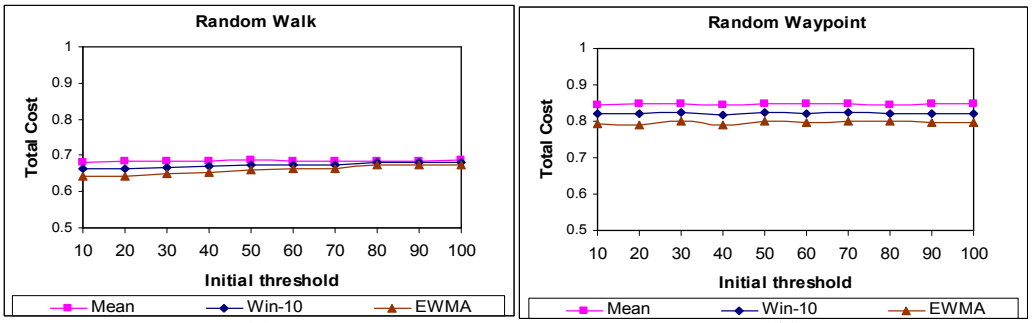
Figure 6.4: Update/probe Rate and Cost

can indeed be attained when $\frac{\phi(d)}{\rho(d)} = \delta^* = 1$, i.e., at the intersection point of the update rate curve and probe rate curve, where $\rho(d) = \phi(d)$.

6.1.3.2 Experiment #2: Algorithm Correctness

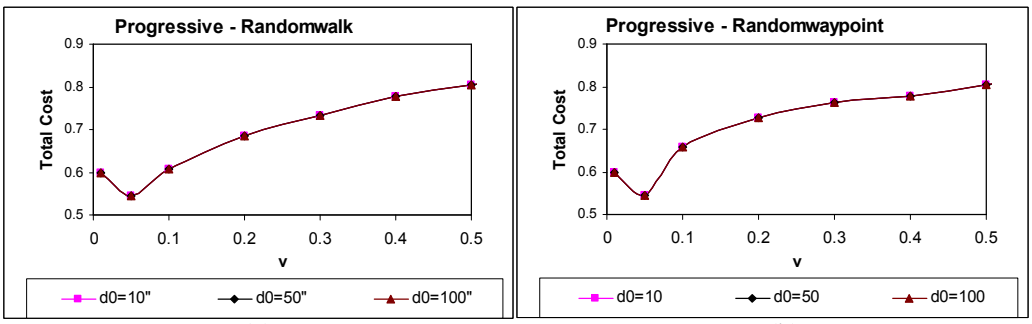
We now attempt to validate the correctness of the adaptive algorithms. Two subsets are conducted to evaluate the conjectural algorithm and the progressive algorithm respectively. Figure 6.5 shows the results of running the conjectural algorithm with different rate monitoring algorithms. X-axis shows various initial thresholds and the results of running MEAN, WINDOW and EWMA are plotted as different curves. The figure shows that the curves with various initial threshold settings are shaped almost like a straight line. This implies that the d_0 value has little impact on algorithm performance. However, the curves with different rate monitoring methods have various performances in terms of the total cost they consume although the cost difference is not very significant. Generally speaking, EWMA scheme has the best performance to monitor the current rate of updating and querying and thus yields lower system cost. Table 6.2 shows the optimal threshold and minimal cost that the conjectural algorithm can obtain.

The progressive algorithm and its parameter setting are then examined. The two parameters we examine are ν and d_0. ν is an important tunable parameter and we expect that d_0 has also little impact on the progressive algorithm as in the case of the



(a) (b)

Figure 6.5: Conjectural with Rate Monitoring



(a) (b)

Figure 6.6: Results with HA Algorithm

conjectural algorithm.

With ν as x-axis, Figure 6.6 presents the results of using HA progressive algorithm with different initial d under both random walk and random waypoint movement models. As expected, the curves with different d_0 values have nearly the same shape in both random walk and random waypoint models. The performance deviations are brought by the setting of ν. It is very clear that the minimal cost point is achieved when ν value is set between 0 and 0.1. Too large or too small setting can lead to cost increase. We repeat the experiment to measure the performance of NHA and the results are very similar to those in Figure 6.6.

Table 6.2 summarizes the best setting for the adjustable parameters and the best threshold d^* returned by the algorithms.

	Conjectural		Progressive	
Parameter	Ran. walk	Ran. waypoint	Ran. walk	Ran. waypoint
ν	no need	no need	0.05	0.05
d_0	no impact	no impact	no impact	no impact
d^*	10.47	12.91	10.94	12.59
cost $\mathcal{C}$	0.64	0.79	0.54	0.57

Table 6.2: Optimal Setting for Parameters

6.1.3.3 Experiment #3: Movement and Query Patterns

The random walk movement model is employed to evaluate the effect of the object movement speed and query pattern. Three speed patterns are tested for the movement pattern examination, i.e., slow, moderate and fast. The tunable parameters are set to their optimal setting according to the experimental results in Experiment #2. The results are presented in Figure 6.7(a). The total system cost increases when the moving speed of objects increases no matter in which algorithm. The reason behind this phenomenon is that a faster movement speed makes an object more easily beyond

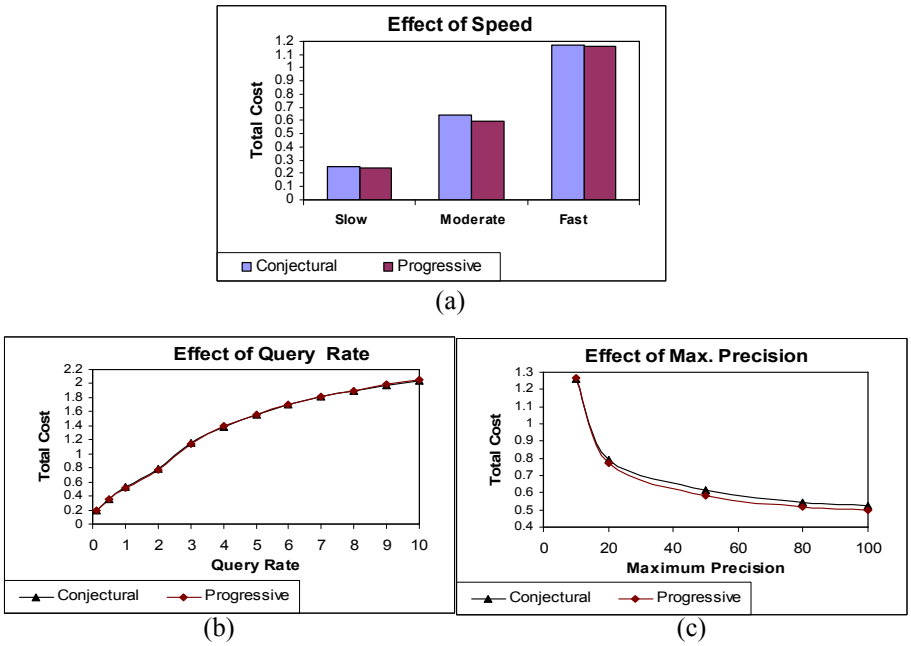


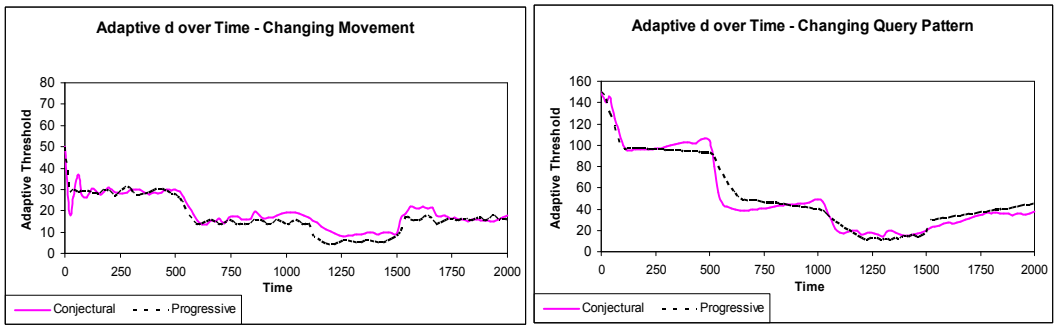
Figure 6.7: Movement and Query Pattern

the distance threshold and tight tracking is needed. It is apparent that the progressive algorithm has a slightly better performance than the conjectural algorithm.

For query pattern examination, we focus on evaluating the effect on the query arrival rate λ and varying query precision r_{max}. The results are depicted in Figure 6.7(b)(c). Both algorithms result in higher system cost at higher query rate and smaller maximum precision constraint. This is expected due to a higher degree of querying activities. In both cases, both algorithms have similar performance.

6.1.3.4 Experiment #4: Algorithm Effectiveness

The last set of experiments is conducted to evaluate the effectiveness of the adaptive algorithms. The aim is to find out how fast the algorithms converge to their best threshold setting when there are changes in system conditions such as object movement pattern, query arrival rate and query precision requirement. To examine the adaptive procedure, the simulation time is extended to 2000 time units and the random walk model is used to generate two sets of simulation data to model changing movement patterns



(a) (b)

Figure 6.8: Convergence of Algorithms

and query pattern respectively. The rate monitoring algorithm used is EWMA. Among the three schemes (i.e. MEAN, WINDOW and EWMA), EWMA is the best as it can adapt to changing situations according to the simulation observations. Thus, EWMA is chosen to track current updating and probing rate.

To simulate the changing movement pattern, the speed of the moving objects is first slowed down after a period of time (e.g. 500 time units in the simulation). To visualize the changing condition, the movement pattern changes suddenly at a time point (e.g. at the time point 500) and remains the same for a while (e.g. 500 time units). Specifically, in this set of simulation, the speed of the moving object is changed from fast to moderate at time point 500, from moderate to slow at time point 1000, and from slow to moderate at time point 1500. The simulation results are presented in Figure 6.8(a). We can see from the figure that during the first 1500 time units both the conjectural and progressive algorithm can adapt the threshold to a smaller value according to the changing speeds. During the last 500 time units, we speed up the moving objects suddenly to see whether the adaptive algorithms can adapt quickly. The results show that the algorithms are effective in this case too. Comparing the performance between the two algorithms, we find that progressive algorithm exhibits a slightly better performance with faster convergence, especially under changing system conditions.

Another simulation changes the query pattern from very infrequent and loose pre-

cision constraint to frequent and tight precision constraint and then relaxes the query frequency and precision constraint after a certain period of time (e.g. 500 time units in this case). To visualize the changing condition, the query pattern also changes suddenly. Specifically, the precision pattern has been changed three times at time point 500, 1000, 1500. The results which are presented in Figure 6.8(b) show that both algorithms can converge to a stable threshold after an adaptation time.

6.1.3.5 Confidence of Experimental Results

To make sure that the simulation results are credible, 95% confidence intervals for each simulation set are computed. As CIs are computed using the standard errors (i.e., s), values of s for various simulation sets are first figured out. And the magnitude of various s is around 0.001. This value is compared with the magnitude of the experimental values which present the total cost in the simulation. The results show that the magnitude of the cost values which is around 0.1 is much significant than that of all the s values.

6.2 Cost-based Scheme for Lazy-probing Protocol

In Chapter 5, the cost function for eager-probing protocol has been defined. The situations for the lazy-probing protocol are more complex. The cost-based scheme is extended in this section to handle the case of lazy-probing protocol.

In the lazy-probing protocol, query processing involves no immediate server probing. Rather, the processing errors involved in the query results are reported to the query issuer. The decision of whether to make any effort to obtain better query results depends on the user's further instructions. However, a query evaluation cost still exists because imprecise answers have impact on the system user's decision making and an important measurement for system performance is the precision of query results returned to the user. In this situation, this kind of cost is quantified as further com-

munication costs. The rationale is that when imprecise results returned to the query issuer do not satisfy the requirement, extra request messages to re-evaluate the query should be issued for better results. The extra request messages from the user's further instructions consume communication costs. To be specific, they include a probe message from the server to the moving object, which should update the imprecise location information, an object location update message from the object to the server as well as the cost of sending query re-evaluation messages from the user to the server. This query processing cost is attributed to the imprecise results returned.

According to the analysis, to define cost function for lazy-probing protocol, we need to review the query processing cost.

6.2.1 Query Processing Cost: A Review

Although no immediate probing is needed in the lazy-probing protocol, cost will still be generated if the imprecision cannot be accepted by the user and further probing requirement is issued. Whether to send query re-evaluation messages to the server is up to the user's requirement. Basically, query issuers accept query results with some errors because of timing consideration and would ask for re-evaluation if the errors involved are too significant to be accepted. As a result, the query processing cost here actually depends mainly on the re-evaluation rate. The re-evaluation rate can be computed based on the user's specification. With the function of re-evaluation rate computation, we can define cost function in the lazy-probing protocol as follows:

$$C^Q = C_e \eta(d, r) \tag{6.3}$$

where C_e is unit cost of query re-evaluation, which translates into the cost paid to re-send the query, probe the object and receive the reply. $\eta(d, r)$ is the re-evaluation rate, i.e., the number of probes generated per time unit, when query precision constraint is not fulfilled and the query needs to be re-evaluated in the server. According to the

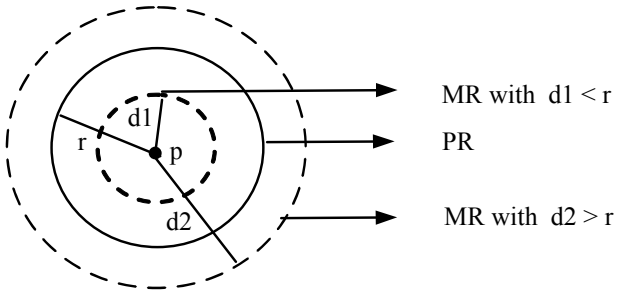


Figure 6.9: Query Precision

user's requirement, the re-evaluation rate can be computed using different methods. To simplify the discussion, a case study for computing $\eta(d, r)$ is presented below.

In the following case study, the precision-based re-evaluation method is proposed. Query precision is an important performance metric for moving object management schemes. When query requirement is met (i.e. $d \leq r$), the precision is 100%. Re-evaluation should be triggered if the precision is too low. To realize the precision-based re-evaluation, we first define the precision.

Given query $q = \langle o_i, r \rangle$ and returned object o_i with threshold d, to define precision of the result $Prec(d, r)$, we now examine the relationship between r and d.

First, we define a *Precision Region (PR)* which has the returned location p of o_i as its center and r as its region radius and a *Possible Moving Region (MR)* which has also p as its center but d as its radius. We then take the probability the object may reside in the precision region as the precision of the result. Figure 6.9 depicts the scenario. When $r \leqslant d$, the probability the object residing in the PR region equals 1. This means query results can fulfill the query requirement and $Prec(d, r) = 1$. When $r > d$, precision never equals 100%. The probability can be computed as the ratio between areas of PR and MR, $Prec(d, r) = \frac{Area(PR)}{Area(MR)} = \frac{\pi r^2}{\pi d^2} = (\frac{r^2}{d^2})$.

Assume that queries arrive with rate λ, each being accompanied by a *precision constraint* sampled from a uniform distribution, $\mathcal{U}(0, r_{max})$. Given a precision threshold Θ, as long as $\sqrt{\Theta}d < r_{max}$, we can then define the probability that the query should be

re-evaluated as $Pr(Prec(d,r) < \Theta) = Pr(r < \sqrt{\Theta}d) = \frac{\sqrt{\Theta}d}{r_{max}}$. When $Prec(d,r) < \Theta$ and Θ is between 0 and 1, it implies that $r < d$. Also, if $\sqrt{\Theta}d \geq r_{max}$, the re-evaluation probability will always be 1 which is the simplest case. Then re-evaluation function $\eta(d)$ is the number of queries issued per time unit multiplied by the probability that the precision constraint of the query is not satisfied and needs to be re-evaluated according to the user's requirement, i.e. $\eta(d) = \lambda\frac{\sqrt{\Theta}}{r_{max}}d$.

We substitute γ for $\lambda\frac{\sqrt{\Theta}}{r_{max}}$ which stands for the other parameters affecting the cost issue other than d. The re-evaluation function can then be expressed as:

$$\eta(d) = \gamma d \tag{6.4}$$

6.2.2 Cost Optimization and Adaptive Algorithms

Up to now, we define the general cost function $\mathcal{C} = C^U + C^Q$, the updating cost $C^U = C_u\rho(d)$ and also the querying cost $C^Q = C_e\eta(d)$.

The final goal is to achieve minimal system cost and optimize the system performance. The optimal setting is passed to the moving object for running the distance-based updating activities.

To do the optimization procedure, analysis on movement and query pattern is essential. Although the exact pattern functions cannot be obtained, the function types can be identified. Here, we assume a 2-dimensional movement is applied and the updating cost can be computed as: $C^U = C_u\rho(d) = C_u\frac{\alpha}{d^2}$. According to the analysis in previous sections, the formula to compute query cost depends on the re-evaluation methods the system user chooses. We define the re-evaluation rate function as: $\eta(d) = \gamma d$ and the function of the total cost can therefore be written as:

$$\mathcal{C} = C_u\rho(d) + C_e\eta(d) = C_u\frac{\alpha}{d^2} + C_e\gamma d \tag{6.5}$$

By differentiation analysis, the optimal value for d is $d^* = \sqrt[3]{\frac{2C_u\alpha}{C_e\gamma}}$. If the up-

date behavior and query workloads are available in advance, the optimal threshold for every object can be determined. Similarly, when we have no knowledge of the changing system environment, we should apply adaptive algorithms to achieve optimization adaptively. As the cost function has a similar format as that in Section 6.1, both the conjectural algorithm and the progressive algorithm for minimizing the total cost can be applied. Table 6.3 lists the symbols that are used in the following sections.

Symbol	Description
$\mathcal{C}$, C^U, C^Q	Total, updating and querying cost
C_u, C_e	Unit cost of location update, probe and re-evaluation
d, d^*	threshold and its optimal value
$\rho(d)$	Object update rate
$\eta(d)$	Re-evaluation rate
r	Precision requirement
α	Summarizing parameter that affects update rate
γ	Summarizing parameter that affects re-evaluation rate
Θ	Precision threshold
ν	Rate of adaptation
δ	Cost ratio, defined as $\frac{\eta(d)}{\rho(d)}$
δ^*	Optimal cost ratio
ϵ	Ping-pong effect barrier parameter

Table 6.3: Symbols and Parameters in Lazy-probing Protocol

Different optimal threshold value is computed in the conjectural algorithm. In the progressive algorithm, the change is in the optimal δ value. The ratio between $\eta(d)$ and $\rho(d)$ is a different constant at the optimal threshold d^*. When $\delta = \frac{\eta(d)}{\rho(d)} = \frac{2C_u}{C_e}$, $d = d^*$. We thus define $\delta^* = \frac{2C_u}{C_e}$, and the optimal threshold d^* is obtained when $\delta = \delta^*$.

The changes of algorithms are highlighted in Figure 6.10, Figure 6.11 and Figure 6.12. The difference between the previous algorithms and the revised ones are highlighted by boxes.

Conjectural Algorithm

initialize d, ρ and η
upon receiving a location update or server probe do
  if a location update is received then
    update ρ using Rate Monitoring Algorithm
  else
    update η using Rate Monitoring Algorithm
  endif
  $\boxed{\alpha \leftarrow \rho d^2}$
  $\boxed{\gamma \leftarrow \frac{\eta}{d}}$
  $\boxed{d^* \leftarrow \sqrt[3]{\frac{2C_u\alpha}{C_e\eta}}}$
  $d \leftarrow d^*$
endo

Figure 6.10: Optimization: Conjectural Algorithm

HA Algorithm

initialize d, ρ and η, set $\boxed{\delta^* = \frac{2C_u}{C_e}}$
upon receiving a location update or server probe activity do
  if a location update is received then
    update ρ using Rate Monitoring Algorithm
  else
    update η using Rate Monitoring Algorithm
  endif
  $\delta \leftarrow \frac{\eta}{\rho}$
  if $\frac{\delta}{\delta^*} > 1 + \epsilon$ then
    $d \leftarrow \frac{d}{1+\nu}$
  else if $\frac{\delta}{\delta^*} < 1 - \epsilon$ then
    $d \leftarrow d(1 + \nu)$
  endif
endo

Figure 6.11: Optimization: HA Algorithm

6.2.3 Simulation Studies

In this section, experiments to examine the performance of the algorithms are conducted for a system running the lazy-probing protocol. Similar to previous experiments, the running time for each experiment is 1000 time units. The spatial domain of interest is a square-shaped region of size 1000 by 1000. The two mobility models, Random Walk and Random Waypoint are used again. Queries arrive with rate λ and

NHA Algorithm
1: initialize d, set $\boxed{\delta^* = \frac{2C_u}{C_e}}$
2: if an object location update is received then
3: $d \leftarrow d(1+\nu)$ with probability $min\{\delta^*, 1\}$
4: endif
5: if a server query evaluation activity is needed for an object then
6: $d \leftarrow \frac{d}{1+\nu}$ with probability $min\{\frac{1}{\delta^*}, 1\}$
7: endif

Figure 6.12: Optimization: NHA Algorithm

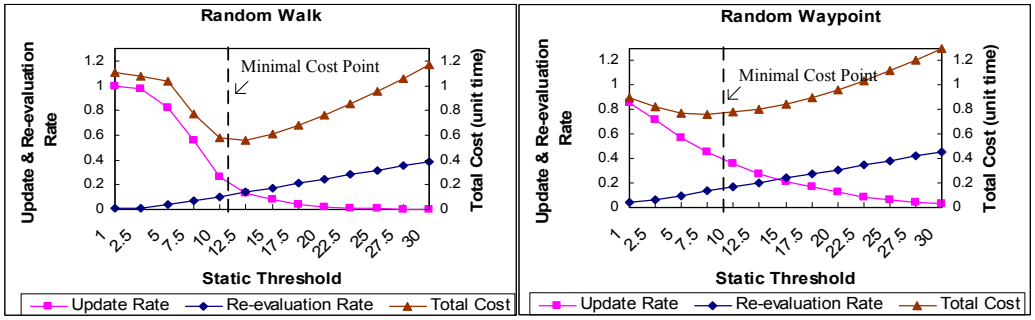
maximum precision constraint r_{max}. An assumption is made that the location update cost per object is 1 and the server re-evaluation cost is 3, with one re-evaluation request message, one server probing message and one object reporting message. Thus, $\delta^* = 2C_u/C_e$ would be 2/3. Table 6.4 summarizes the setting of the simulation parameters.

Parameter	Value range	Default
L (in second)	1	1
k (in meter)	1 (slow), 5 (moderate), 15 (fast)	5
λ	1 - 10 s^{-1}	1
V_{max}	15	15
T_{max}	2	2
r_{max}	10-100	50
δ*	2/3	2/3
Θ	0.1-1	0.5

Table 6.4: Lazy-protocol Simulation Parameter Setting

6.2.3.1 Experiment #1: Assumption Validity

Similar to previous experiments, simulation studies are conducted to establish the correctness of the assumption for the relationship among d and $\rho(d)$ and $\eta(d)$, i.e., to show that generally $\rho(d)$ and $\eta(d)$ are proportional to $1/d^2$ and d respectively. Both movement models with their default value set are used in the evaluation.



(a) (b)

Figure 6.13: Update/probe Rate and Cost

The updating and querying algorithms with a fixed d for each experiment are run, but d is varied across experiments. The average number of updates and probes per time unit are measured. Figure 6.13 presents the simulation results. The measured values for $\rho(d)$ and $\eta(d)$ are found to be proportional to $1/d^2$ and d respectively. The figure also shows that the minimal cost point is achieved when the ratio between re-evaluation and update rate is close to 2/3.

6.2.3.2 Experiment #2: Algorithm Correctness

This set of experiments is conducted to validate the claim that the adaptive algorithms can adapt the threshold d towards the optimal value d^*, i.e., they converge. Two subsets are experimented. The first subset is to examine the conjectural algorithm. Figure 6.14 shows the results of running the conjectural algorithm. X-axis shows various initial thresholds. For both random walk and waypoint movement models, it is clear that the initial threshold setting has little impact on the performance of the conjectural algorithm. Table 6.5 shows the optimal threshold and minimal cost that the conjectural algorithm can obtain.

The progressive algorithm and its parameter setting are now examined. The progressive algorithm controls the magnitude of the threshold adjustment by means of ν, an important tunable parameter. ν as well as the initial threshold d in the experiment are varied.

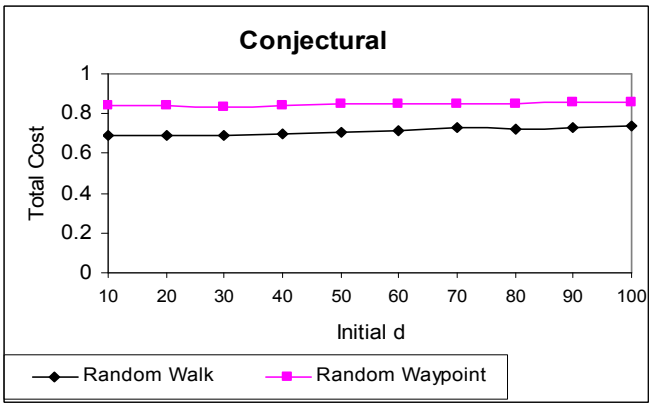
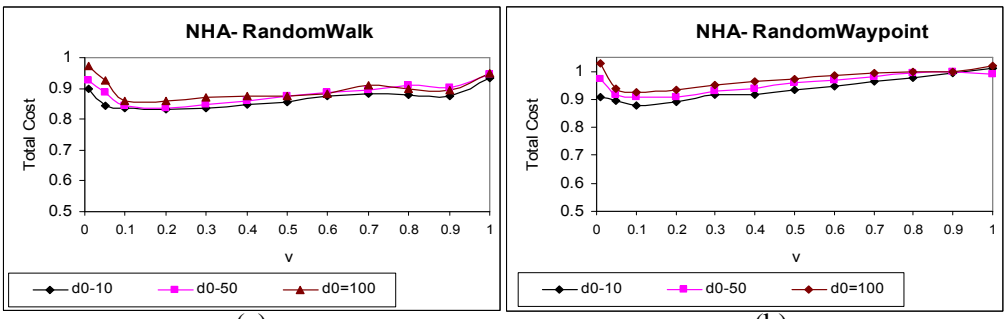


Figure 6.14: Results with Conjectural Algorithm



(a) (b)

Figure 6.15: Results with NHA Algorithm

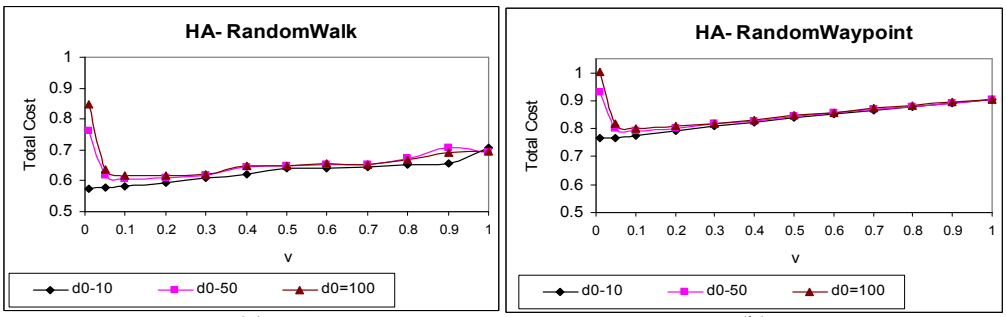
	Conjectural		NHA		HA	
Parameter	Ran. walk	Ran. waypoint	Ran. walk	Ran. waypoint	Ran. Walk	Ran. waypoint
ν	no need	no need	0.1	0.1	0.1	0.1
d_0	no impact	no impact	no impact	no impact	no impact	no impact
d^*	10.21	7.02	10.62	8.72	10.57	7.33
cost $\mathcal{C}$	0.69	0.81	0.77	0.82	0.60	0.75

Table 6.5: Optimal Setting for Parameters in Lazy-probing Protocol

With ν as x-axis, Figure 6.15 presents the results of using NHA progressive algorithm with different initial d under both random walk and random waypoint movement models. It seems that similar to the case in the conjectural algorithm, the initial setting for threshold d has little impact on the performance. The performance difference among the three simulation sets with different initial threshold settings is less than 0.05. The slight performance difference can be explained by the various time periods that are taken to adapt the initial threshold to the optimal threshold value. For a large initial threshold (e.g. $d_0 = 100$), the time paid to adapt the initial value to the optimal one (i.e. d^* is around 10 in this case) is longer. As a result, the cost should be higher as a penalty for this longer adaptation procedure. Figure 6.6 also reveals that ν should be set between 0 and 0.1 in order to obtain the optimal performance result.

The experiment is repeated to measure the performance of HA algorithm and the results are presented in Figure 6.16. The performance trends and phenomena are very similar to the case in NHA algorithm.

Table 6.5 summarizes the best setting for the adjustable parameters and the best threshold d^* returned by the algorithms. Compared with the results in Figure 6.13, the best threshold returned from both conjectural and progressive algorithms are close to the optimal value. The results also indicate that progressive algorithms perform better in exerting a lower total system cost.



(a) (b)

Figure 6.16: Results with HA Algorithm

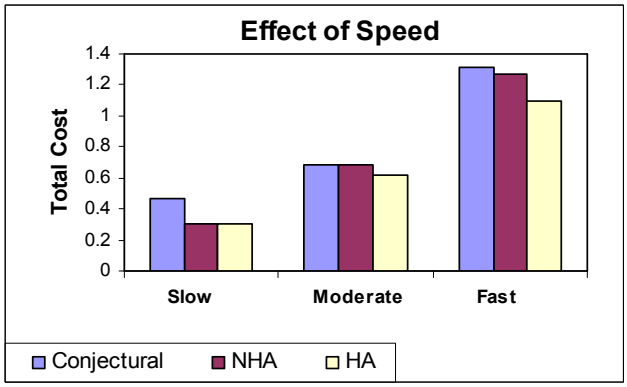
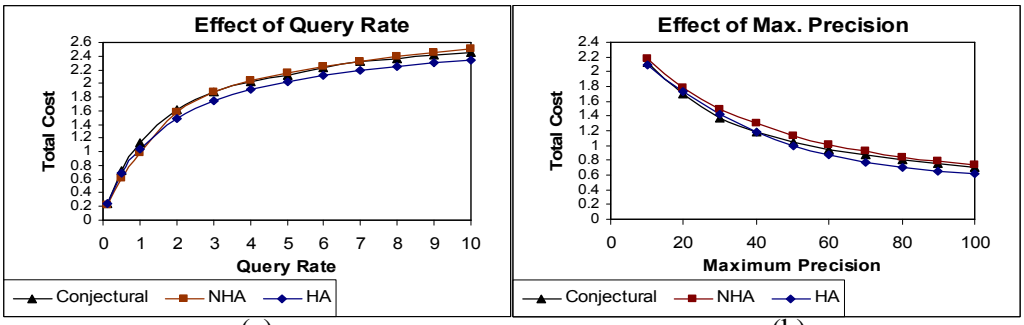


Figure 6.17: Effect of Movement Pattern

6.2.3.3 Experiment #3: Effect of Movement Patterns

The random walk movement model is employed to evaluate the effect of the object movement pattern. Three speed patterns are tested for movement pattern examination, i.e., slow, moderate and fast. The results are presented in Figure 6.17. The total system cost increases when moving speed of objects gets faster no matter in which algorithm. The reason behind this phenomenon is that faster movement speed makes an object more easily beyond the distance threshold and tight tracking is needed. It is apparent that the progressive algorithm has a slightly better performance than the conjectural algorithm. Overall speaking, HA algorithm has the best performance.



(a) (b)

Figure 6.18: Effect of Query Pattern

6.2.3.4 Experiment #4: Effect of Query Pattern

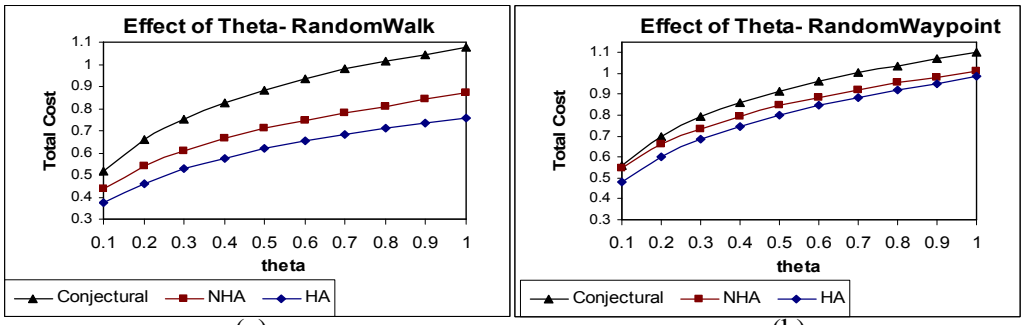
For query pattern examination, we focus on evaluating the effect on the query arrival rate λ and varying query precision r_{max}. The results are depicted in Figure 6.18. All the three algorithms result in higher system cost at higher query rate and smaller maximum precision constraint. This is expected due to a higher degree of querying activities. In both cases, all the algorithms have similar performance.

6.2.3.5 Experiment #5: Effect of Precision Threshold

The last set of simulation is conducted to evaluate the impact on the setting of the precision threshold Θ. Both random walk and random waypoint movements are evaluated. The simulation results are presented in Figure 6.19. As a larger Θ means that the user requires more precise query results, the cost of answering this kind of queries is expected to be higher than that in the case of requiring smaller Θ. For the conjectural algorithm and the two progressive algorithms, similar impact has been brought by Θ.

6.2.3.6 Confidence of Experimental Results

The experimental results are examined by computing the confidence intervals (CI) of each experiment point. It is found that the magnitude of the experimental values which present the total cost is around 0.1 and the magnitude of the standard error



(a) (b)

Figure 6.19: Effect of Precision Threshold

(i.e, s) for these values is around 0.001. Therefore, the 95% CI for the results is $[-0.00196, 0.00196]$. This range is small and is insignificant to credible simulation results.

6.3 Discussion

In this chapter, the basic cost-based model introduced in Chapter 5 has been extended. First, we have relaxed the one-dimensional movement assumption in the basic CUP scheme. It has been found from the analysis that the adaptive algorithms for performance optimizing are applicable to the new scenario. The algorithms have been revised and the effectiveness has been examined by simulation work. Second, some ways to apply the adaptive scheme in the lazy-probing protocol have been discussed. It has been found that unless further re-evaluation situation is provided by the system user, we cannot proceed with optimization work for the lazy-probing scenario. A case study has been employed to apply the adaptive model and to investigate the probability for performance optimization. The simulation studies show positive results.

Chapter 7

Conclusion

This chapter gives concluding remarks to this thesis, and proposes some potential future directions for extending previous work.

7.1 Concluding Remarks of the Thesis

In this thesis, the efficient location management techniques for moving objects in mobile environment have been explored. As continuous movement is the essential feature of moving objects in the mobile environment, location management plays a fundamental and essential role in providing efficient LDIS services. Overall speaking, location management problem in moving object environment involves several interrelated components, namely, positioning technology, location modeling, the storing problem, location updating and query processing. Among all the components, location information updating and query processing are two key issues in location management.

The goal for a good location updating design is less network traffic and more precise location information. While for query processing, the goal is efficient processing and precise results. To build a good location management scheme, both goals are the performance targets. It is revealed that the *uncertainty problem* affects both updating and querying issues. To handle the uncertainty and build an efficient updating and query-

ing model, three approaches were proposed in previous studies. However, all of the approaches have limitations and problems. Based on the analysis of these limitations and problems, this current work proposes that several desirable features which can help to remove the limitations should be equipped in a location management model.

These proposed features include query awareness, movement awareness, cost optimization and error tolerance. As query processing on moving objects is the main purpose of moving object location tracking with respect to most practical applications, query awareness should be considered in an efficient location management model. Movement awareness can help to reduce resource consumption and the risks of location uncertainty. As the best way to handle a tradeoff when it is inherently generated from two competitively leading factors is to optimize it, cost optimization is desired for efficient location management. Error tolerance is another desired feature when time is also an important resource for query issuers. Under certain circumstances query issuers need replies from the server as soon as possible without further probing, which is much more time consuming. When the error tolerance feature is provided, this situation can be well handled without sacrificing the query precision.

As the fundamental step towards the design of an efficient management scheme which integrates the desired features, a detailed examination of the two common location management models and the essential components of each of them was first conducted in this study. Updating and querying models that can be equipped in a location management scheme were then explored. Three location management models are proposed to fulfill the requirement of the desired features.

The Aqua scheme is an Adaptive QUery-Aware location updating model. It is built upon the general query-aware updating and querying model. The target of query-aware model is to take query information into consideration for the design. As query processing on moving objects is the main purpose of moving object location tracking, the query-aware location management model has been designed to remove the limitations of previous work which has ignored the mutual impact between updating and

querying.

The CUP scheme is a design using the cost-based model and provides a target cost function and several adaptive optimization algorithms to achieve the minimal cost point. These algorithms can adapt objects management activities according to not only the changing movement pattern of moving objects, but also the changing query situation in the system.

To make the basic cost-based model more general, the CUP scheme has been further extended to handle the error tolerance requirement set by the system user. Error tolerance ability is useful when the time consumption is one of the important performance metrics the user takes into account. The extended cost optimization model provides a mechanism to allow the user to control the errors that may be involved in query results.

7.2 Future Work

One of the directions for further work is to generalize the query processing model in the proposed schemes. In Chapter 3, the query types and the three dimensions involved have been explained. With regard to the different classes of these dimensions, only the non-joint query under the spatial dimension, the time-slice query the under temporal dimension and the common query under the lasting time dimension have been examined in this study. Other types of useful queries, such as join queries, continuous queries and nearest neighbor queries have not yet been investigated. Further explorations could focus on the possibility of applying the adaptive approaches to deal with different types of queries in moving object environments.

As the proposed location management schemes are so far applicable only in the client-server model, extending these schemes into the peer-to-peer model is another direction for possible future work. In the peer-to-peer location management model, the architecture between location information holders and consumers is different. Al-

though the desired features required by efficient management schemes remain the same, the design components and adaptive algorithms to achieve these features may vary to a large extend. Further work on the possibility of extending the existing models and methods to fit the peer-to-peer environment should be valuable.

Appendix A

Notation List

Table A.1: List of Symbols in Chapter 4

Symbol	Meaning
o_i	Moving object with identity i
p	Object position
m	Message passed between object and server
d	Distance threshold
d_0	Initial threshold
d'	Query adjustable distance threshold
q_j	Query with identity j sent by issuer
r	Query precision requirement
r_{max}	Maximum query precision requirement
PR	Precision region
MR	Possible moving region
Q_p	Precision
Q_c	Recall
λ	Query arrival rate
κ	Parameter for performance tuning

Table A.1: List of Symbols in Chapter 4

Symbol	Meaning
c^q	Region center of quasar
r^q	Region radius of quasar
N	Total number of groups
G	Object group
k	Group identity
F	Group feature
D	Spatial domain
z	System parameter of grid size
ω	Adjustable weight parameter for tracking query patterns
s	Zipf's law parameter
Δ	Query standard deviation

Table A.2: List of Symbols in Chapter 5 and Chapter 6

Symbol	Meaning
o	Moving object
i	Object identity
q	Query
p	Object position
d	Object distance threshold
d_0	Initial threshold
d^*	Optimal threshold value
r	Query precision requirement
r_{max}	Maximum query precision requirement

Table A.2: List of Symbols in Chapter 5 and Chapter 6

Symbol	Meaning
v	Object moving velocity
$\mathcal{C}$	Total cost
C^U	Update cost
C^Q	Query cost
C_u	Unit cost of update activity
C_q	Unit cost of probing moving object
C_e	Unit cost of query re-evaluation
ρ	Update rate of moving object
ϕ	Query probing rate
λ	Query arrival rate
α	Summarizing parameter that affects update rate
β	Summarizing factor that affects query arrival rate
γ	Summarizing parameter that affects re-evaluation rate
δ	Cost ratio
δ^*	Optimal cost ratio
ν	Rate of adaptation
ϵ	Ping-pong effect barrier parameter
w	Window size
c	Current time slot
Rt_j	Rate in time slot j
ω	Adjustable weight parameter in EWMA algorithm
k	Distance of each step in random walk movement
L	Time duration of each step in random walk movement
n	Number of steps in drunken person problem
V_{max}	Maximum velocity in random waypoint movement
T_{max}	Maximum time duration in random waypoint movement
η	Query re-evaluation rate

Table A.2: List of Symbols in Chapter 5 and Chapter 6

Symbol	Meaning
Θ	Precision threshold

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 175–186, 2000.

[2] I. F. Akyildiz, J. Ho, and Y.-B. Lin. Movement-based location update and selective paging for PCS networks. *IEEE/ACM Transactions on Networking*, 4(4):629 – 638, Aug. 1996.

[3] I. F. Akyildiz and J. S. M. Ho. Dynamic mobile user location update for wireless PCS networks. *Wireless Networks*, 1(2):187–196, 1995.

[4] A. Bar-Noy, I. Kessler, and M. Sidi. Mobile users: To update or not to update? *ACM/Baltzer Journal on Wireless Networks*, 1(2):175–186, 1994.

[5] D. Barbara. Mobile computing and databases: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):108 – 117, Jan. 1999.

[6] B. Becker, S. Gschwind, B. S. T. Ghler, and P. Widmayer. An asymptotically optimal multiversion B-trees. *The VLDB Journal*, 5(4):264–275, 1996.

[7] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis. Nearest neighbor and reverse nearest neighbor queries for moving objects. In *Proceedings of International Database Engineering and Applications Symposium*, pages 44–53, Edmonton, Canada, 2002.

[8] Y. Cai, K. A. Hua, and G. Cao. Processing range-monitoring queries on heterogeneous mobile objects. In *Proceedings of the 2004 IEEE International Conference on Mobile Data Management*, pages 27–38, Jan. 2004.

[9] J. Chen, X. Meng, B. Li, and C. Lai. Tracking network-constrained moving objects with group updates. In *Proceedings of 7th International Conference on Advances in Web-Age Information Management*, pages 158–169, Hong Kong, China, 2006.

[10] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1112–1127, Sept. 2004.

[11] R. Cheng, S. Prabhakar, and D. V. Kalashnikov. Querying imprecise data in moving object environments. In *Proceedings of the 19th International Conference on Data Engineering*, pages 723–725, Bangalore, India, Mar. 2003.

[12] H. D. Chon, D. Agrawal, and A. El Abbadi. NAPA: Nearest available parking lot application. In *Proceedings of the 18th International Conference on Data Engineering*, pages 496–497, California, USA, 2002.

[13] H. D. Chon, D. Agrawal, and A. El Abbadi. Range and kNN query processing for moving objects in grid model. *ACM/Kluwer MONET Journal*, 8(4):401–412, 2003.

[14] A. Civilis, C. S. Jensen, J. Nenortaite, and S. Pakalnis. Efficient tracking of moving objects with precision guarantees. In *Proceedings of the 1st Annual International Conference on Mobile and Ubiquitous Systems*, pages 22–26, Boston, Massachussets, USA, 2004.

[15] A. Civilis, C. S. Jensen, and S. Pakalnis. Techniques for efficient road-network-based tracking of moving objects. *IEEE Transactions on Knowledge and Data Engineering*, 17(5):698–712, 2005.

[16] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceeding of the 22nd International Conference on Distributed Computing Systems*, pages 23– 32, Austria, July 2002.

[17] M. Erwig, R. H. Güting, M. Schneider, and M. Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *GeoInformatica*, 3(3):269–296, 1999.

[18] L. Forlizzi, R. H. Guting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. In *Proceedings of 2000 ACM SIGMOD Conference*, pages 319–330, Dallas, 2000.

[19] B. Gedik and L. Liu. Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In *Proceedings of the 9th International Conference on Extending Database Technology*, pages 67–87, 2004.

[20] H. Gowrisankar and S. Nittel. Reducing uncertainty in location prediction of moving objects in road networks. In *Proceedings of the 2nd International Conference on Geographic Information Science*, Boulder, Colorado, 2002.

[21] R. H. Guting, M. H. Bohlen, M. Erwig, and C. S. Jensen. A foundation for representing and querying moving objects. *ACM Transactions on Database Systems*, 25(1):1–42, 2000.

[22] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of 1984 ACM SIGMOD Conference*, pages 47–57, Boston, Massachusetts, 1984.

[23] J. Hightower and G. Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, 2001.

[24] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, 1999.

[25] H. Hu, J. Xu, and D. L. Lee. A generic frame work for monitoring continuous spatial queries over moving objects. In *Proceedings of the 24th International Conference on Management of Data*, pages 479–490, Baltimore, Maryland, 2005.

[26] P. Y. C. Hwang and R. G. Brown. *Introduction to Random Signals and Applied Kalman Filtering*. 2nd Ed. John Wiley and Sons, Inc., 1992.

[27] T. Imielinski and B. R. Badrinath. Querying in highly mobile distributed environments. In *Proceedings of the 18th Conference on Very Large Databases*, 1992.

[28] T. Imielinski and B. R. Badrinath. Data management for mobile computing. *SIGMOD Record*, 22(1):34–39, 1993.

[29] T. Imielinski and B. R. Badrinath. Mobile wireless computing: Challenges in data management. *Communications of the ACM*, 37(10):18–28, 1994.

[30] G. Iwerks, H. Samet, and K. Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. In *Proceedings of the 29th International Conference on Very Large Data Bases*, Berlin, Germany, Sept. 2003.

[31] G. S. Iwerks, ozy Hanan Samet, and K. P. Smith. Maintenance of spatial semijoin queries on moving points. In *Proceedings of the 30th International Conference on Very Large Data Bases*, pages 828–839, Toronto, Canada, 2004.

[32] C. S. Jensen, D. Lin, and B. C. Ooi. Query and update efficient B+-tree based indexing of moving objects. In *Proceedings of the 30th International Conference on Very Large Data Bases*, pages 768–779, 2004.

[33] C. S. Jensen and S. Saltenis. Towards increasingly update efficient moving-object indexing. *IEEE Data Engineering Bulletin*, 25(2):35–40, 2002.

[34] C. S. Jensen, D. Tiesyte, and N. Tradisauskas. Robust B+-tree-based indexing of moving objects. In *Proceedings of the 7th International Conference on Mobile Data Management*, page 12, 2006.

[35] D. V. Kalashnikov, S. Prabhakar, and S. E. Hambrusch. Main memory evaluation of monitoring queries over moving objects. *Distributed and Parallel Databases*, 15(2):117–135, 2004.

[36] M. Khambatti and S. Akkineni. Location management in peer-to-peer mobile ad hoc networks. In *Technical Report, Computer Science and Engineering Department, Arizona State University*, April 2002.

[37] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 261–272, 1999.

[38] R. kr. Majumdar, K. Ramamritham, and M. Xiong. Adaptive location management in mobile environments. In *Proceedings of the 4th International Conference on Mobile Data Management*, pages 196–211, Melbourne, Australia, 2003.

[39] A. Kumar, V. Tsotras, and C. Faloutsos. Designing access methods for bitemporal databases. *IEEE Transactions on Knowledge and Data Engineering*, 10(1):1–20, 1998.

[40] G. Lam, H. Leong, and S. Chan. GBL: Group-based location updating in mobile environment. In *Proceedings of the 9th International Conference on Database Systems for Advanced Applications*, pages 762–774, Jeju Island, Korea, 2004.

[41] K. Y. Lam, O. Ulusoy, T. S. H. Lee, E. Chan, and G. Li. An efficient method for generating location updates for processing of location-dependent continuous queries. In *Proceedings of Database Systems for Advanced Applications*, pages 218–225, Hong Kong, China, 2001.

[42] G. Lawler. *Intersection of random walks*. Birkhuser Boston, 1996.

[43] G. Lawler. *Conformally Invariant Processes in the Plane*. American Mathematical Society, 2005.

[44] I. Lazaridis, K. Porkaew, and S. Mehrotra. Dynamic queries over mobile objects. In *Proceedings of the 8th International Conference on Extending Database Technology*, pages 269–286, 2002.

[45] D. L. Lee, J. Xu, B. Zheng, and W.-C. Lee. Data management in location-dependent information services. *IEEE Pervasive Computing Journal*, 1(3), 2002.

[46] K. C. K. Lee, H. V. Leong, and A. Is. Approximating object location for moving object database. In *Proceedings of International Workshop on Mobile Distributed Computing*, pages 402–407, Providence, RI, USA, May 2003.

[47] M. L. Lee, W. Hsu, C. S. Jensen, B. Cui, and K. L. Teo. Supporting frequent updates in R-tree: a bottom-up approach. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 608–619, Berlin, Germany, Sept. 2003.

[48] A. Leonhardi, C. Nicu, and K. Rothermel. A map-based dead-reckoning protocol for updating location information. In *Proceedings of the 16th International*

Parallel and Distributed Processing Symposium, pages 15–23, Washington, DC, USA, 2002.

[49] A. Leonhardi and K. Rothermel. A comparison of protocols for updating location information. *Cluster Computing*, 4(4):355–367, 2001.

[50] B. Liang and Z. J. Haas. Predictive distance-based mobility management for PCS networks. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1377–1384, Mar. 1999.

[51] Y. Manolopoulos, Y. Theodoridis, and V. Tsotras. *Advanced Database Indexing*. Kluwer Academic publisher, 1999.

[52] M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: scalable incremental processing of continuous queries in spatio-temporal databases. In *Proceedings of the Special Interest Group on Management Of Data*, Paris, France, June 2004.

[53] Z. Naor and H. Levy. Minimizing the wireless cost of tracking mobile users: an adaptive threshold scheme. In *Proceedings of the 17th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 720–727, San Francisco, CA, Mar. 1998.

[54] M. A. Nascimento and J. R. O. Silva. Towards historical R-trees. In *Proceedings of ACM Symposium on Applied Computing*, pages 235 – 240, Atlanta, Georgia, United States, 1998.

[55] J. R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics (No. 2), University of Cambridge, 1998.

[56] C. Olston and J.Widom. Efficient monitoring and querying of distributed, dynamic data via approximate replication. *IEEE Data Engineering Bulletin*, 28(1):11–18, 2005.

[57] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *Proceedings of 2001 ACM SIGMOD Conference*, Santa Barbara, California, USA, 2001.

[58] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 144–155, Cairo, Egypt, Sept. 2000.

[59] J. M. Patel, Y. Chen, and V. P. Chakka. STRIPES: An efficient index for predicted trajectories. In *Proceedings of International Conference on Management of Data*, pages 635 – 646, Paris, France, 2004.

[60] N. W. Paton, A. A. A. Fernandes, and T. Griffiths. Spatio-temporal databases: Contentions, components and consolidation. In *Proceedings of the 11th International Workshop on Database and Expert Systems Applications*, pages 851–855, Greenwich, London, 2000.

[61] E. Pitoura and G. Samaras. Locating objects in mobile computing. *Knowledge and Data Engineering*, 13(4):571–592, 2001.

[62] G. P. Pollini and C.-L. I. A profile-based location strategy and its performance. *IEEE Journal on Selected Areas in Communications*, 15(8):1415–24, Oct. 1997.

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

[64] K. Raptopoulou, A. N. Papadopoulos, and Y. Manolopoulos. Fast nearest-neighbor query processing in moving obect databases. *Geoinformatica*, 7(2):113 – 137, 2003.

[65] J. F. Roddick, M. J. Egenhofer, E. Hoel, and D. Paradias. Spatial, temporal and spatio-temporal databases - hot issues and directions for PhD research. *SIGMOD Record*, 33(2):126–131, 2004.

[66] C. Rose. Minimizing the average cost of paging and registration: A timer-based method. *Wireless Networks archive*, 2(2):109 – 116, 1996.

[67] C. Rose. State-based paging/registration: A greedy technique. *IEEE Transactions on Vehicular Technology*, 48(1):166–173, Jan. 1999.

[68] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of 1995 ACM SIGMOD Conference*, pages 71–79, San Jose, CA,, May 1995.

[69] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proceedings of the International Conference on Management of Data*, pages 331–342, Dallas, Texas, 2000.

[70] B. Salzberg and V. Tsotras. A comparison of access methods for temporal data. *ACM Computing Survey*, 31(2):158–221, 1999.

[71] T. K. Sellis. Research issues in spatio-temporal database systems. In *Proceedings of the 6th International Symposium on Spatial Databases*, pages 5–11, Hong Kong, China, 1999.

[72] S. K. Sen, A. Bhattacharya, and S. K. Das. A selective location update strategy for PCS users. *ACM/Baltzer Jounal on Wireless Networks*, 5(5):313–26, Sept. 1999.

[73] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proceedings of the 13th International Conference on Data Engineering*, pages 422–432, Birmingham, UK, Apr. 1997.

[74] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Querying the uncertain position of moving objects. In *Temporal Databases: Research and Practice*, pages 310–337, Germany, June 1997.

[75] Q. Sun and H. Garcia-Molina. Slic: A selfish link-based incentive mechanism for unstructured peer-to-peer networks. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems*, pages 506–515, Hachioji, Tokyo, Japan, March 2004.

[76] Y. Tao and D. Papadias. Time-parameterized queries in spatio-temporal databases. In *Proceedings of 2002 ACM SIGMOD Conference*, pages 334–345, Madison, Wisconsin, 2002.

[77] Y. TAO and D. Papadias. Spatial queries in dynamic environments. *Transactions on Database Systems*, 28(2):101–139, 2003.

[78] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 287–298, Hong Kong, China, 2002.

[79] Y. Tao, D. Papadias, and J. Zhang. Cost models for overlapping and multi-version structures. *ACM Transactions on Database Systems*, 27(3):299–342, 2002.

[80] J. Tayeb, O. Ulusoy, and O.Wolfson. A quadtree-based dynamic attribute indexing method. *The Computer Journal*, 41(3):185–200, 1998.

[81] Y. Theodoridis, T. K. Sellis, A. Papadopoulos, and Y. Manolopoulos. Specifications for efficient indexing in spatiotemporal databases. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pages 123–132, Capri, Italy, 1998.

[82] G. Trajcevski, O. Wolfson, B. Xu, and P. Nelson. Real-time traffic updates in moving objects databases. In *Proceedings of the 13th International Workshop*

on Databse and Expert Systems Applications, pages 698–704, Aix-en-Provence, France, Sept. 2002.

[83] M. Vazirgiannis, Y. Theodoridis, and T. Sellis. Spatio-temporal composition and indexing for large multimedia applications. *Multimedia Systems*, 6(4):284–298, 1998.

[84] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and imprecision in modeling the position of moving objects. In *Proceedings of the 14th International Conference on Data Engineering*, pages 588–596, Orlando, FL, Feb. 1998.

[85] O. Wolfson, S. Chamberlain, K. Kalpakis, and Y. Yesha. Modeling moving objects for location based services. In *Proceedings of the NSF Workshop on Developing an Infrastructure for Mobile and Wireless Systems*, pages 46–58, London, UK, 2002.

[86] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases Journal*, 7(3):257–387, 1999.

[87] O. Wolfson, P. Sistla, B. Xu, J. Zhou, S. Chamberlain, yelena Yesha, and N. Rishe. Tracking moving objects using database technology in DOMINO. In *Proceedings of The Fourth Workshop on Next Generation Information Technologies and Systems (NGITS)*, pages 112–119, Zikhron-Yaakov, Israel, 1999.

[88] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving objects databases: Issues and solutions. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pages 111–122, Capri, Italy, 1998.

[89] O. Wolfson and H. Yin. Accuracy and resource consumption in tracking moving object. In *Proceedings of the 8th International Symposiumon Spatial and Temporal Databases*, pages 325–343, Santorini Island, Greece, 2003.

[90] V. Wong and V. C. M. Leung. Location management for next-generation personal communications networks. *IEEE Network*, 16(5):18–24, Sept. 2000.

[91] X. Xiong, M. F. Mokbel, and W. G. Aref. SEA-CNN: scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *Proceedings of the International Conference of Data Engineering*, Tokyo, Japan, 2005.

[92] B. Xu and O. Wolfson. Time-series prediction with applications to traffic and moving objects databases. In *Proceedings of ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 56–60, San Diego, CA, USA, 2003.

[93] J. Xu, X. Tang, and D. L. Lee. Performance analysis of location-dependent cache invalidation scheme for mobile environments. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):474–488, 2003.

[94] J. Xu, Y. Tang, and W.-C. Lee. Ease: An energy-efficient in-network storage scheme for object tracking in sensor networks. In *Proceedings of the 2nd IEEE Conference on Sensor and Ad Hoc Communications and Networks*, Santa Clara, CA, 2005.

[95] X. Yu and S. Mehrotra. Capturing uncertainty in spatial queries over imprecise data. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, pages 192–201, Prague, Czech Republic, 2003.

[96] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *Proceedings of 21st International Conference on Data Engineering*, pages 631–642, 2005.

[97] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based spatial queries. In *Proceedings of 2003 ACM SIGMOD Conference*, pages 443 – 454, San Diego, California, USA, 2003.

[98] M. M. Zonoozi and P. Dassanayake. User mobility modeling and characterization of mobility patterns. *IEEE Journal on Selected Areas in Communications*, 15(7):1239–1252, 1997.