國立交通大學

網路工程研究所

碩 士 論 文

基 於 行 動 端 點 位 置 之 動 態 查 詢 分 配

Dynamic Query Assignment

based on the Location of Mobile Hosts

研 究 生：游柏瑞

指導教授：張明峰　教授

中 華 民 國　九 十 八　年 八 月

基 於 行 動 端 點 位 置 之 動 態 查 詢 分 配

Dynamic Query Assignment based on the Location of Mobile Host

研 究 生：游柏瑞　　Student：Bo-Rei Yu

指導教授：張明峰　　Advisor：Ming-Feng Chan

國 立 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

August 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年八月

基 於 行 動 端 點 位 置 之 動 態 查 詢 分 配

學生：游柏瑞　　指導教授：張明峰教授

國立交通大學網路工程研究所 碩士班

摘要

隨著無線網路與行動通信的快速發展，與手持裝置與行動裝置的廣泛使用，以及全球衛星定位系統的普及，位置服務能基於使用者的位置提供在地化的資訊，使得現代人可以享受的智慧的生活型態。位置查詢的型態也分為很多種類，連續的位置查詢會持續一段時間，期間的結果也須保持正確，被查詢的物體也可能會移動。範圍監測查詢是一個連續的查詢，如果指定區域內的狀態有改變，須回報伺服器來更新結果。目前位置服務大多是採用伺服器與用戶端的架構，當使用者數目或是查詢要求增加，系統的負擔與可調適性將會是個挑戰。分散式的計算能讓行動端點分擔伺服器的計算。在本研究中，主要的範圍分為兩個，一個是查詢結果的更新問題，另一個是分配範圍監測查詢給行動端點之問題。本系統之設計為分配一個矩形區域給行動端點，包含數個範圍監測查詢，如果行動端點其在此區域內且不影響到範圍監測查詢，則不須向伺服器回報其位置更新以及查詢結果之改變。本系統能基於行動端點之位置和能力，分配範圍監測查詢，使得查詢結果更新之次數減少，整個系統的負擔將會減輕，可調適性增加，系統便可提供更大容量與更多樣的服務。

Dynamic Query Assignment based on the Location of Mobile Host

Student: Bo-Rei Yu Advisor: Prof. Ming-Feng Chang

Institute of Network Engineering

National Chiao Tung University

Abstract

LBS services are capable of providing information based on the user's location. There are many varieties of LBS Query. Continuously queries continuously update its result until the users terminate the query requests. Range monitoring queries are typical examples of continuous queries. If there is any change in the region under monitored, LBS must update the query result immediately. LBSs are usually implemented in client-server architecture. When the user population is huge and many requests are sent to the server. The scalability of the LBS server becomes a problem. Distributed client-server architecture enable clients to share their computing ability with the server. One possible distributed solution is that the LBS server assigned queries near each mobile client so that the clients can check if the queries match. There are two problems in this approach. One problem is query result update. The other problem is query assignment. In our design, we compute a rectangle area for every mobile host. The mobile hosts aware the range monitoring queries in the region. If the mobile host is inside the region and does not affect any query in the system, it does not need to report to the server. Our system dynamically assigns range monitoring queries to the mobile hosts based on their capability. The query update messages and the server load in the system are reduced. The system scalability is also increased.

誌謝

首先感謝最敬重的導師 張明峰教授。學生就讀碩士班的期間，教授有耐心的指導、訓練獨立思考、指引研究方法方向，及細心發掘問題。學校與張教授也提供良好完善的研究與實驗環境、器材，讓學生能順利完成論文，感謝老師兩年來的教誨。

在這兩年碩士班就讀期間，感謝實驗室的學長，冠璋、忠育、坤揚和君飛，同學玄亞、威凱，在研究與修課過程互相勉勵與生活上多采多姿。

最後將論文獻給我最親愛的家人、晏伶。感謝您們在我求學期間全心全意的支持與鼓勵，讓我得以順利的完成學業。

游柏瑞 謹識於

國立交通大學網路工程研究所碩士班

中華民國九十八年八月

List of Figures

List of Tables

Table of Contents

Chapter 1 Introduction

1.1 Motivation

The vast development of wireless network and mobile communication technologies, the widely use of handheld and laptop devices, and the popularity of Global Positioning System (GPS) make Location-Based Services (LBS) attractive to a lot of people. In recent years, many powerful mobile phones are available to the market. The tiny devices have 3G, Wireless LAN, and Bluetooth connectivities, and many are equipped with two cameras and GPS receiver. User can not only use them to take pictures but also can use LBS to enjoy the intelligent lifestyle.

LBS services are capable of providing information based on the user's location [1-3]. The application of LBS services is ubiquitous in our life. Every morning, we can get local weather information based on our location. If you have a meeting with somebody, you can just follow the directions of your phone and know where the person is. When driving, users can get turn by turn navigation to places. The traffic information can be used to help us not to get stuck on traffic, which saves time and be more environmental friendly. On the way to destination, a user can get the information of nearby gas stations and the lowest gas price on the route. Before arrival, you can use LBS to help you find a parking place. If you want to use public transportation, you also can get the schedule and its real-time status. After work, you see e-coupons on the phone sent by the restaurant in the city you lives. You can choose whatever you what and make a reservation. On the way home, you can see where your children are on the phone and pick them up. When got home, you can feel free to relax because the air conditioner has been ready. All this can be done from a

handheld device.

LBS Queries are submitted by users of LBS Services. The server receives a query request, processes it, and then sends result back to the user. There are many varieties of LBS Query [4]. The type of query can be range query [5], nearest-neighbor query [6], and many variants. The range query in LBS is that given a region area, we can retrieve the interested information in the area. The nearest-neighbor query is to find the closet point of interest in a space. The duration of the query can be snapshot or continuously. The snapshot query is one-shot. The result is static. The continuously query continuously update its result until the user terminate the query request. The time of the query can be past, present, and future. The query and object in the query can be stationary or moving. For example, the user at home sent a query request for the nearest hospital. The type of this query is nearest-neighbor query. The duration is snapshot. The time is present. The query and objects in this case are stationary. The police officer sent a query request for continuous report the number of cars on the freeway. The type of this query is range query. The duration is continuous. The time is present and future. The query is stationary and the objects in this case are moving. The user driving on the highway may be interested in what are the nearest gas stations for the next hour. The type of this query is nearest-neighbor query. The duration is continuous [7]. The time is future. The query is moving and the objects in this case are stationary. The other user at home may be interested in what is the closest distance between him and his friends yesterday. The type of this query is closest point query [8]. The duration is snapshot. The time is past. The query and the objects are moving.

LBSs are usually implemented in client-server architecture [9]. The server processes the requests from the client side. The client side is usually mobile hosts.

Mobile hosts usually have limited computation ability, battery capacity and wireless bandwidth. To ensure the correctness of query results, the client side usually periodically reports its position.

When the user population is huge and many requests are sent to the server. The server will be a bottleneck. The scalability of the LBS will also become a problem. There are also other architectures for implementation of LBS. The idea of distributed client-server architecture is to let clients share their computing ability. Divide the work into server site processing and client site processing. The server managers and maintains queries, and act as a mediate of requests and clients. The clients monitor the nearby queries and send the updates of the query result. Another approach is decentralized peer to peer architecture [10-11] . This type of LBS system does not have central control or knowledge of all nodes.

In this thesis, the architectures we used to implement LBS Services is assumed to be distributed client-server architecture. The type of query is range query. The duration is continuous. The time is present. The query is stationary and the objects are moving. This query is also known as range monitoring query. The definition is: given a set of rectangle regions, we want to know the mobile hosts inside them. And we can get real-time update when the mobile hosts move in and out of these regions.

1.2 Research Questions

In detail, the primary research question that we address is as follows. The result of range monitoring query is dynamic. Mobile hosts may frequently move in and out of query region. The result may only valid for a very short time. Continuous update the results or periodically updates make heavy and unbalanced load on the server. The mobile hosts also need to communicate to the server through Wireless LAN or Mobile

communication interface, which is battery consuming. There are also huge messages on the network. Decreasing the location update requests not only can lessen the server load but also save the battery power of mobile hosts. Although the range monitoring query process was introduced in many papers [12-16], we present a more effective algorithm based on the location of mobile hosts.

In distributed client-server architecture of LBS, the division of work into server side is processing and client side processing is still a problem to resolve. Using the client recourse better can make the LBS more scalable. The distributed client-server architecture of LBS was used in previous work. However, we present an algorithm based on the capability of the mobile host.

Another related question is that mobile hosts have different capability. How to allocate suitable work that meets the capability of every mobile host is important. We proposed the algorithm that tries to fit every mobile host's capability.

1.3 Related work

To deal with the problem of huge location update in range monitoring query, Prabhakar et al. proposed the concept of Safe Region[17]. A safe region is defined to be a circular or a rectangle region that contains mobile host's location but not overlap with any query boundary. If mobile host is inside the safe region, it does not affect any query in the system. If it moves out of the safe region, it instructs the server and requests a new safe region. Because the safe region cannot overlap with query boundary, the size of safe region is usually small when the query density is high. Unfortunately, computing a rectangular safe region takes from O(n) to $O(n\log^3 n)$, where n is the number of queries. When adding new queries to the system, the server may also need to compute all safe regions for the mobile hosts because the new query

may affect existing safe regions.

Ying et al. proposed a scalable and adaptive technique, Monitoring Query Management (MQM)[13], for real-time processing of range monitoring queries. The resident domain concept they proposed is that a server assigns each mobile host a resident domain (region) that contains a number of queries based on mobile host's current location and ability. The server also notifies the mobile host the queries that overlap with that domain. When a mobile host detects that it has cross over some query boundary, it contacts the server to updates the query results. When a mobile host leaves the resident domain, it reports to the server to get a new resident domain. They allow the mobile host to monitor its nearby queries and report the result to the server.

To leverage the computing capability of mobile host, MQM proposed a BP-Tree (Binary Partitioning Tree) spatial access method in the server for efficient query management. BP-Tree can be used to find the resident domain with a number of queries that meets the capability of the mobile host. The detail design of MQM will be described in Chapter 2.

1.4 Overview

This dissertation is organized as follows. Chapter 2 presents related work. Chapter 3 presents the algorithms and system architecture. Chapter 4 discusses the results in the system design. Finally, Chapter 5 concludes this thesis and describes the future work.

Chapter 2 Related Work

In this research, we focus on the query result update problem and query assignment problem. In the following sections, we define the problems and describe the related research work.

2.1 The Query Result Update Problem

Unlike conventional range queries, a range-monitoring query is a continuous query. The query stays active for a period of time until it is terminated by the user. The simplest way to keep the result correct is that whenever mobile host moves, report its position to the server. The server then processes the request and updates the query result. There are many disadvantages in this approach: The constant or periodically location update from mobile hosts can quickly exhaust the mobile devices' battery. For the central server, the requests from mobile host are huge. The server will be busy process these messages only. The scalability is poor.

In Safe Region, the safe region is defined to be either a circular or a rectangular region that contains the mobile host's location and does not overlap with any query boundary. The mobile host does not need to report its position when it is within the safe region.

In MQM, a scalable and adaptive technique for real-time processing of range monitoring queries, which allows mobile hosts to monitor their nearby queries. Every time when a mobile host moves, it checks if the queries are affected by it. Then send update messages to the server to ensure real-time and accurate query results. They

proposed a resident domain concept to decrease the number location update; a spatial access method BP-Tree for efficient query management at the server side.

2.2 The method to deal with Query Result Update

In safe region, consider a mobile host that is far way from any query. The mobile hosts need to travel a large distance before move in one of the queries. The short distance between a mobile host and a query boundary is called Safe Distance. The mobile host needs to move a distance of at least Safe Distance before its cross over any query boundary. They also define a SafeRect, a safe maximal rectangle around the mobile host's current location. Figure 2-1 shows mobile host A and its safe region, the safe region does not overlap any query rectangle. If mobile host A does not move out its safe region, it does not report to the server. If mobile host A cross over its safe region boundary, it reports to the server. However, mobile host exits its safe region does not guarantee that the query is affected. When the queries in the domain are dense, the safe region will be very small. In addition, computing a rectangular safe region takes from O(n) to O($nlog^3n$), where n is the number of queries.

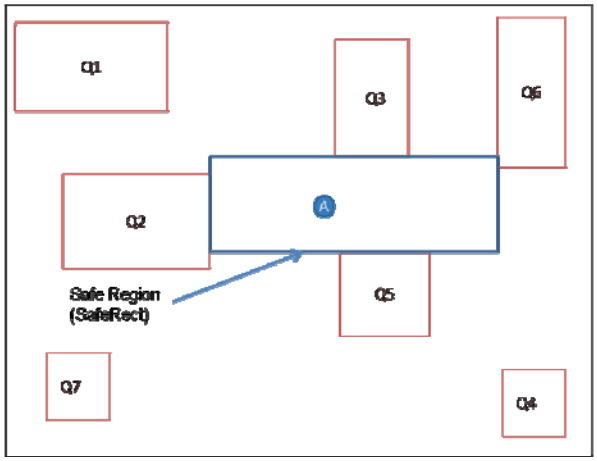

Figure 2-1 An example of Safe Region

In MQM, they assign each mobile host a resident domain, based on its current

location. The resident domain also includes the queries that overlap with the domain. When the mobile host detects that it has crossed over a query boundary, the mobile host will communicate the server to update the affected query results. When the mobile host moves out of its resident domain, it also needs to report to the server. Then the server will compute a new resident domain for the mobile host.

There are some assumptions in the MQM. They assume that each query is represented by a rectangular region. Each mobile host has a limited resource. For example, the CPU speed, memory capacity, and battery power are all limited. Mobile host can communicate to the server through wireless or mobile networks. Figure 2-2 shows mobile host A and its resident domain. If mobile host A does not exit its resident domain, it does not report to the server. If mobile host A cross over queries in its resident domain, it will report to the server to update the query result. When mobile host A exits its resident domain, the server will compute a new resident domain that fit A's capability.

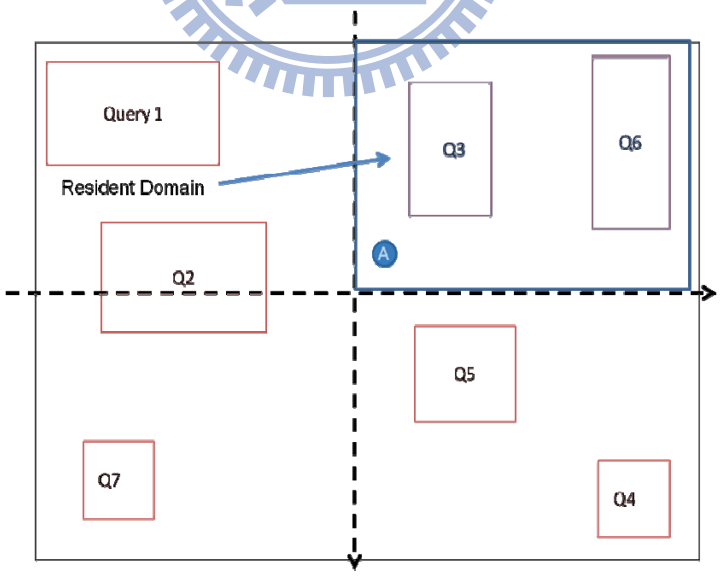

Figure 2-2 An example of Resident Domain.

2.3 Query Assignment

The computing capability of mobile host is measured by the maximum number of queries it can load and process at a time. For example, if a mobile host's capability is n queries, then the resident domain for it should contain as many queries but not exceed its capability. The problem of search for a resident domain is trying to find n query rectangles that are near an object's current location. When a query overlaps with a subdomain, the overlapping area is called monitoring region inside the subdomain. A query may create one or more monitoring region if it spans over multiple subdomains. A subdomain may have multiple queries, and we can use one or more subdomains as a mobile host's resident domain if the number of queries in side does not exceed the mobile host's capability. Subdomains and monitoring regions are maintained in a BP-Tree.

2.4 The Method of Query Assignment

In MQM, the architecture for implementation of LBS Services is distributed client-server architecture. The BP-Tree spatial structure is used for query and subdomain management. When a new query q is submitted, the server searches the BP-Tree to find the subdomain it overlaps. The server then inserts the monitoring region to the BP-Tree. When the number of monitoring regions in a subdomain exceeds the split threshold (the minimum number of query that the least capable mobile host can load), the subdomain is partitioned into two equal size subdomains. When a mobile host enters or exit a query boundary (monitoring region), it sends a location update message to the server, including its current position. Server updates the query and notifies the mobile host in that query.

A BP-Tree consists of two types of nodes: domain node and date node. All non-leaf nodes are domain node. Each domain node is the decomposition of its parent node. Figure 2-3 shows the decomposition of D_1 is D_{11} and D_{12}. Each domain node has a variable, *size*, to record the total number of monitoring regions under this domain. A data node is a leaf node, consists an array of rectangles that stores queries (monitoring regions) under its parents. There is also a variable, *size*, to record the total number of monitoring regions under this node.

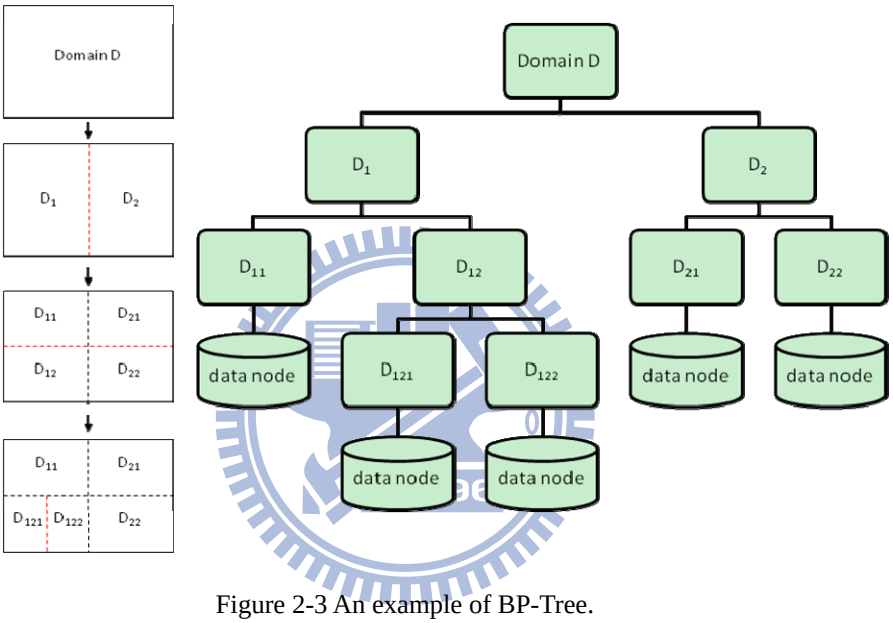

Figure 2-3 An example of BP-Tree.

To let all mobile hosts can load at least one subdomain, the size of data nodes is limited by the minimum processing capability mobile hosts. The parameter is the split threshold for data nodes.

The BP-Tree efficiently supports the resident domain search. Given a position p and capability n queries of the mobile host, the server searches the BP-Tree from the root. If the number of monitoring regions inside the subdomain fit the mobile host's capability (smaller or equal to n) and the mobile host is also inside, the subdomain can be the resident domain of the mobile host. Otherwise, the server descends the tree to check the domain that contains the mobile host. In the worst case, we may take the

subdomain that has data node. When a resident domain is selected, the server then retrieves all queries inside to notify the mobile host.

In Figure 2-4, the BP-Tree's split threshold is set to 1 for clearly show on the figure. Each subdomain has only one monitoring region (query). The mobile host A's capability is 5. Figure 2-5 shows the BP-Tree of this domain.

To search a resident domain for mobile host A in Figure 2-4, the server first process the request of resident domain from mobile host A. When mobile host is initializing or exits a resident domain, it will request the server to assign a resident domain. The mobile host's capability is 5. Server searches from root of BP-Tree, find subdomain d2 is suitable for the mobile host. Finally, server assigns d2 as the mobile host's resident domain. The queries in the resident domain is also fit A's capability.

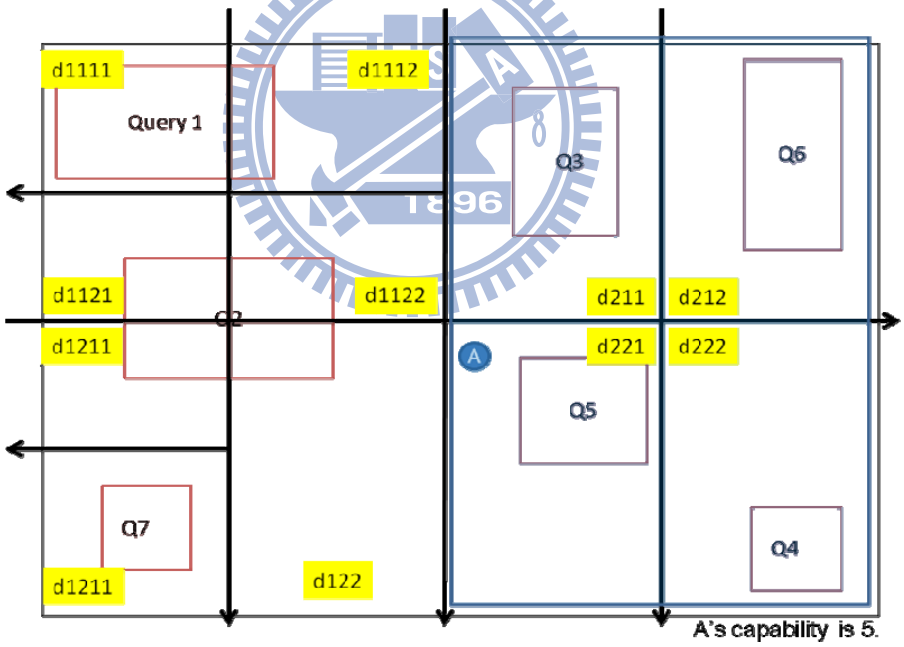

Figure 2-4 Resident Domain for mobile host A.

When mobile host A moves out of its current resident domain, as show in Figure 2-6. The mobile host requests the server to assign new resident domain again. The server assign d1 as A's new resident domain.

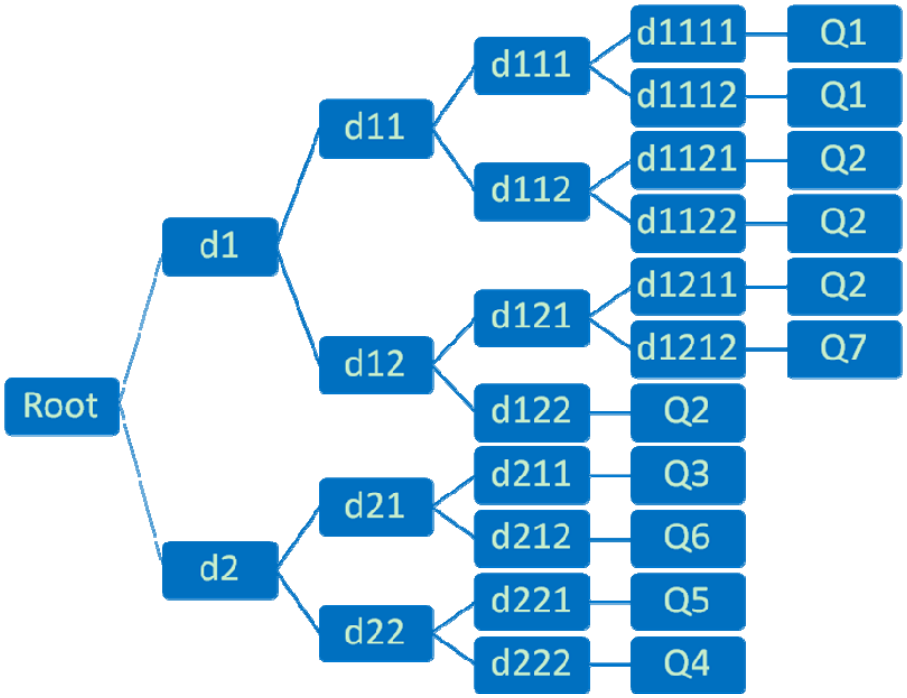

Figure 2-5 BP-Tree for the above domain.

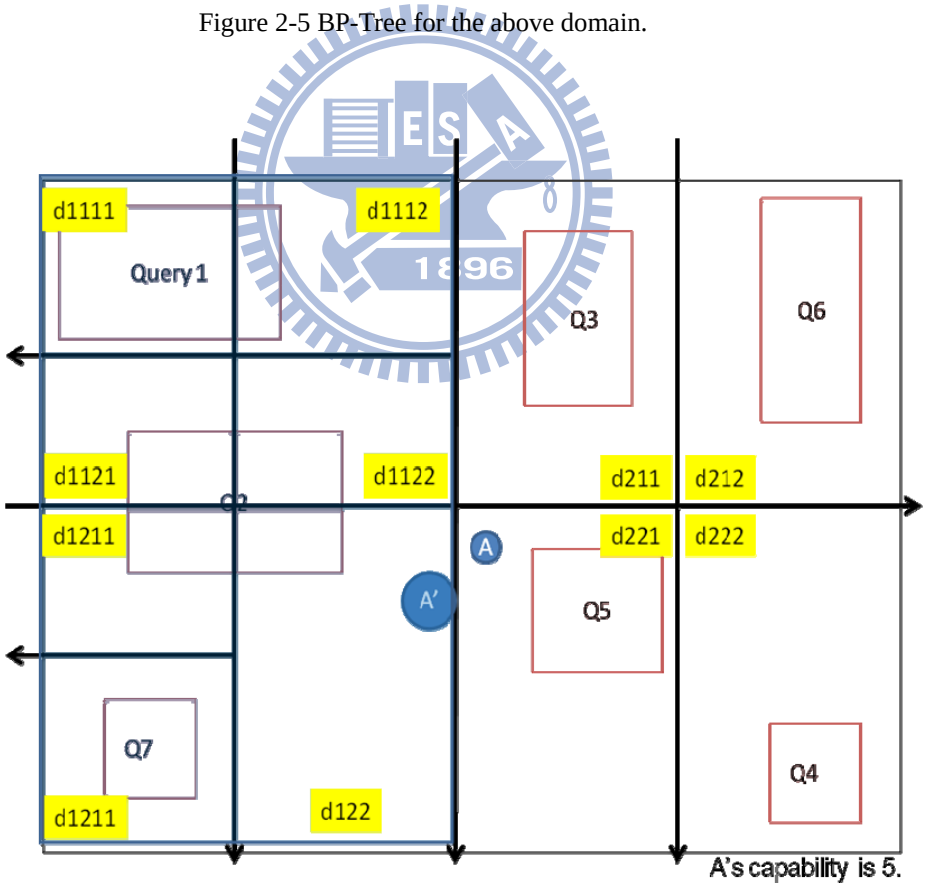

Figure 2-6 Resident Domain for mobile host A'.

However, A is not in the center of its resident domain, A may easily moves out of its current resident domain. And most of time the mobile host has requested for new resident domain, it just moved out of the last resident domain. The position of mobile host is near the boundary of multiple resident domains. In addition, the position reported by mobile host from GPS has some error, the mobile host may detect that it is cross over its resident domain boundary. This make mobile host request for a new resident domain again and again.

There is still one problem in MQM. The mobile host A's capability is 5 queries, but the resident domain assign to it contains 4 queries. The server searches resident domain from the root. In this case, the number of queries in root domain exceeds the mobile host's capability, and then the server descends the tree to check the subdomain d2 that contains the mobile host A. Server only allocate one or more whole unit domain to mobile host. This makes the mobile host's capability would not always be satisfied.

The distributed client-server architecture, computing ability requirement, and memory storage requirement will also be required in our system. We then present the algorithm based on the location and capability of mobile host. We will allocate a rectangle area for the mobile host, which the mobile host is in the center of rectangle area. We will extend the safe region rectangle to overlap queries, not only load a whole unit domain, so that we can try to meet the capability of the mobile host. In this study, we may decrease the number query result update by using rectangle area, and share the server load with the mobile hosts by query assignment.

Chapter 3 System Design and Implementation

3.1 System Overview

In this research, we aim to create a real-time range-monitoring query management system. The system accepts range-monitoring query from a user, and real-time update the query result. The design of our system is based on the following assumptions: (1) The mobile hosts have computing abilities to monitor nearby queries. (2) Mobile hosts are able to locate their positions and have synchronized clocks, e.g. using GPS. (3) The mobile hosts can set up connections to the server through mobile or WLAN network. Figure 3-1 shows the system architecture. In this section, we will describe the solutions of the query insertion, query result update and query assignment problems.

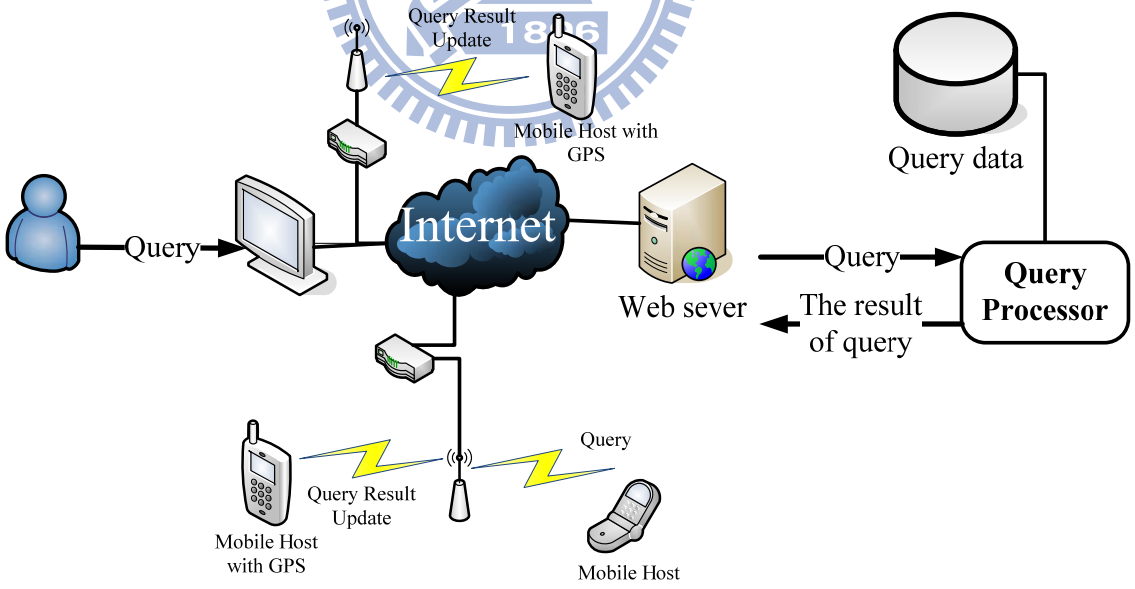

Figure 3-1 System Overview

3.2 Query Insertion

3.2.1 Overview

In this section, we present our method of query insertion. In our system, the domain area is a geographical area, as a city or a district. We presented the domain by a rectangle. The query is also represented by a rectangle that overlaps on the domain area. Figure 3-2 shows five queries on the map, we can think the domain as a city map and the queries are small areas that the users interest in on the map. We index the domain and queries on the tree data structure to support domain searching and query assignment.

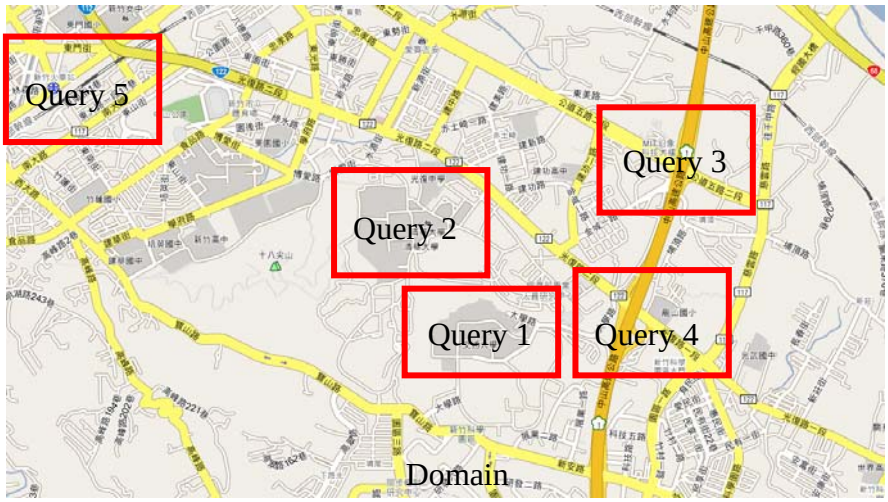

Figure 3-2 Queries on the map. (the map is from maps.google .com)

3.2.2 The method of Query Insertion

We use and modify the BP-Tree and algorithms provided by MQM. Figure 3-3 shows 6 queries overlap a small area on the domain. The query rectangle edges that cross the borders of domain rectangle, and the edges of the query that inside the domain will be stored in the data node. Figure 3-4 shows the domain node and data node structure. The domain node D_i has stores its domain rectangle D_i.domain which contains the lower left and upper right coordinate of the rectangle, a pointer to its

parent domain D_i.parent, a pointer to its child domain or data node D_i.child and D_i.size records the total number of query rectangles stored descending from D_i. The data node d_i stores the total number of query rectangles in D_i.size, and it contains two ordered lists that stores the edges of query rectangles. d_i.X_list stores ordered edge list of query rectangle ascendantly by X-coordinate, and d_i.Y_list stores ordered edge list of query rectangle ascendantly by Y-coordinate. Figure 3-5 shows the value of d_i.X_list and d_i.Y_list in data node of domain D_i.

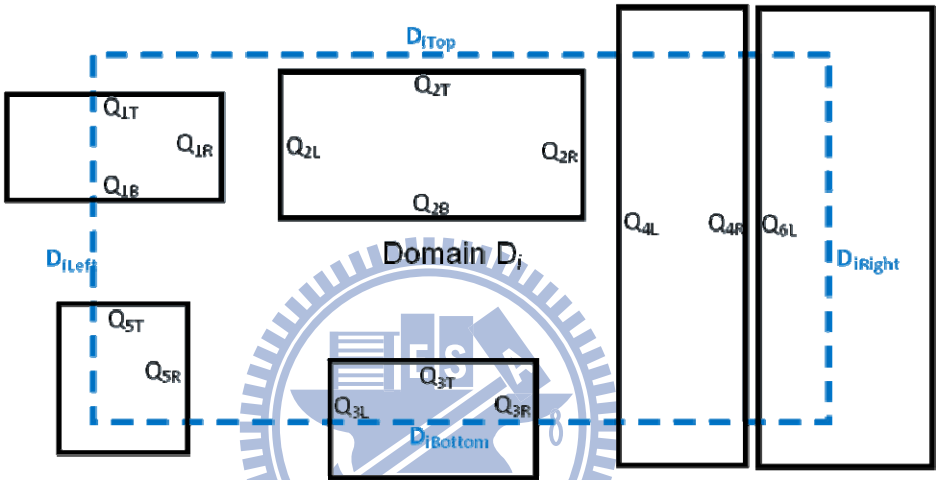

Figure 3-3 Queries that overlaps the domain.

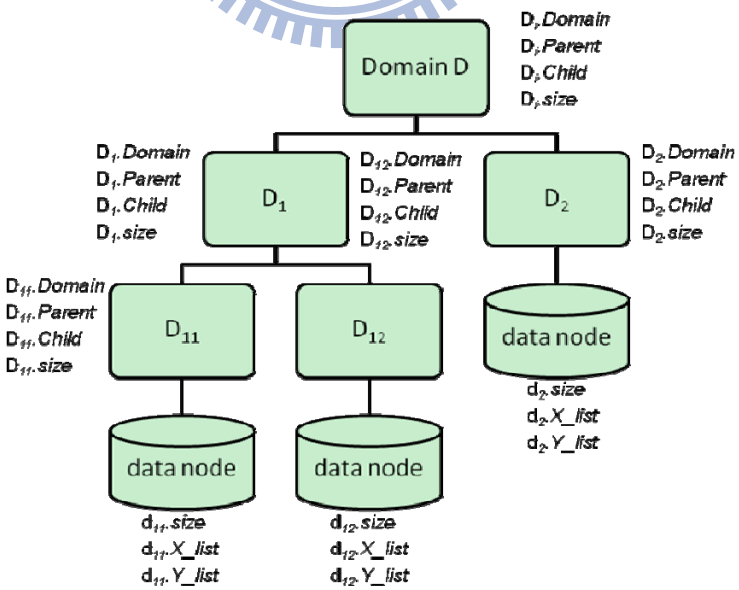

Figure 3-4 Domain node and data node in modified BP-Tree.

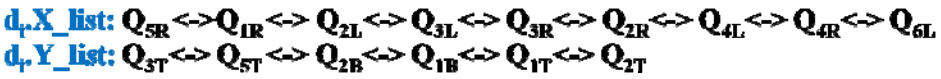

Figure 3-5 The edges that are stored in the data node of Domain D_i.

The query rectangles that overlap the domain rectangles have the property below:

The four edges of the domain rectangle D_i : D_{iLeft}, D_{iRight}, $D_{iBottom}$, D_{iTop}

The four edges of the query rectangle Q_i : Q_{iLeft}, Q_{iRight}, $Q_{iBottom}$, Q_{iTop}

For all edges that are overlap the domain rectangles,

$D_{iLeft} < Q_{iLeft}$ (1)

$D_{iRight} > Q_{iRight}$ (2)

$D_{iBottom} < Q_{iBottom}$ (3)

$D_{iTop} > Q_{iTop}$ (4)

The query rectangle edges that satisfy these inequalities will be stored on the X_list or Y_list of date node. To find the query rectangle edges, we may refer to the rectangle intersection problem[18]. In our system, we check the query rectangle edges and store the edges that satisfy these inequalities. The AddQuery and InserList algorithms are below.

AddQuery(D_i, Q_j)
{
//Descend BP-tree to find the data nodes to store query.

- If D_i is a domain node, then for each entry (D_i, P), call AddQuery(**D_i.P**, Q_j) if D_i overlaps with Q_j.
- If D_i has a data node, then:
  InsertList(D_i, Q_j);
- If D_i is full,
  SplitDataNode(D_i); //Provided by MQM

}

```
InsertList(Di, Qj)
 {
     // Insert Qj's point to Di's data node(di) if it overlaps Di
    If (QjLeft > DiLeft)
        Insert (QjLeft, Di.X_list);
    If (QjRight < DiRight)
        Insert (QjRight, Di.X_list);
    If (QjTop < DiTop)
        Insert (QjTop, Di.Y_list);
    If (QjBottom > DiBottom)
        Insert (QjBottom, Di.Y_list);
}
```

3.3 Query Result Update

3.3.1 Overview

In this section, we present our method of query result update strategy which is a solution of the location update problem. In this distributed architecture, server allocates some queries to mobile host to relieve its load. We leverage the mobile host's tiny computing power to share the load with the central server. We also observe that the mobile host's position in resident domain will affect the number of location update.

3.3.2 The method of computing the Rectangle Area

We use and modify the resident domain concept and algorithms provided by MQM. In our system, the Rectangle Area is a rectangle that around the mobile host. It also includes n queries that can fit the mobile host's capability. When the mobile host is just registered to our system or leaves its current rectangle area, it sends a request for rectangle area. The information contains its current location and current time by GPS on the mobile host. And its capability is also reported to the server. The server then computes the rectangle area based on the current location and capability of the

mobile host. The rectangle area is extended from the current position of mobile host to include n queries.

After receiving its rectangle area and queries, the mobile host checks whether it is still in the rectangle area when it moves. It also checks if it is cross over any query boundary. If it is in the rectangle area and not crosses over query boundary, it does not need to report to the server for query results update. Thus, the location update is avoided. The server load and communication costs are saved. When the mobile host detects that it has cross over some query boundary, it contacts the server to updates the query results. Or when a mobile host leaves it rectangle area, it sends a request to the server to get the new rectangle area and queries. Figure 3-6 shows the concept of rectangle area, the mobile host in the figure is A with capability 2.

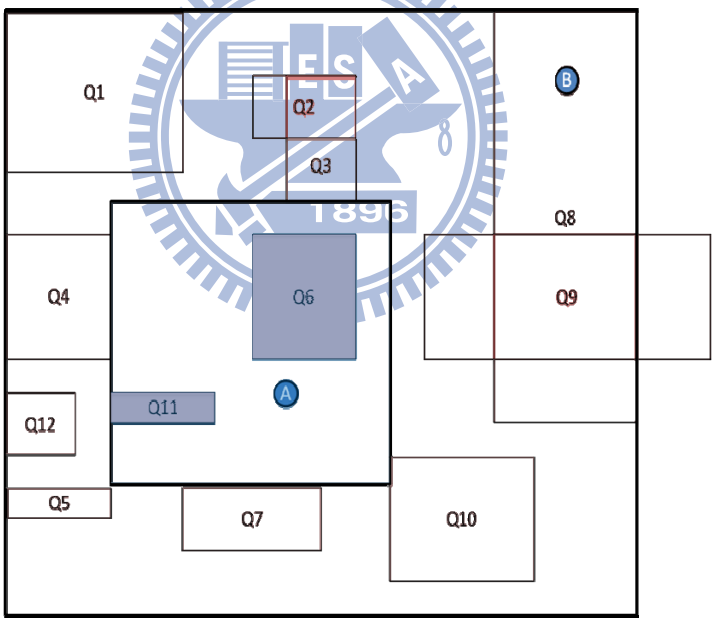

Figure 3-6 The rectangle area of mobile host A.

3.4 Query Assignment

3.4.1 Overview

The domain represents a rectangle area in geography. Query is also a rectangle

that overlaps the domain area. Their relationship is managed in the BP-Tree. The server searches the BP-Tree down to the data node to find the mobile host's position in the query rectangle edges lists. Then it computes rectangle area for mobile hosts by extending its border to include n queries to fit mobile host's capability. The detail design and algorithms are introduced below.

3.4.2 The method of Query Assignment

The AssignQuery algorithm input Domain Root D_i and mobile host O_i to assign rectangle area and query for the mobile host. Initially, it calls FindDomain to find the leaf domain where the mobile host in it. LoadList loads the data node inside the leaf domain to retrieve the edge lists in it, and set two pointers point to the border of domain rectangle. Then it calls ComputeRECT to compute the rectangle area.

AssignQuery(D_i, O_j)
{
- **FindDomain(D_i, O_j);** //Find the leaf-domain D_i where O_j is in.
- **LoadList(D_i);** //Load the D_i.X_list and D_i.Y_list.
- **FindList(D_i, O_j);** //Find O_j's position on the Query List.
  - **D_i.pX** points to the first equivalent position or points to the first element that is higher than O_j on the D_i.X_list.
  - **D_i.pY** points to the first equivalent position or points to the first element that is higher than O_j on the D_i.Y_list.
- RECT = **ComputeRECT(D_i, O_j)**
- Return RECT; //Return the Rectangle area and Queries for MH.

}

In ComputeRECT, it extends its rectangle area to enclose n queries to fit the mobile host's capability. It starts from the mobile host's position. For each query rectangle it reaches, we compute the minimum step from mobile host's X-coordinate to reach the boarder of Q_{iLeft} or Q_{iRight}. And the minimum step from mobile host's Y-coordinate to reach the boarder of Q_{iTop} or $Q_{iBottom}$, respectively. Then we choose the maximum of these two values as the step to extend to reach the query rectangle, stores the query id and this value to the priority queue. Then it checks that the steps of

rectangle area extended are larger than the minimum step in the priority queue or not. If it is true, it represents that the rectangle area has cross the query boundary. The extend process is to adjust the domain edge border pointer that point to edge lists of query rectangle in data node. We extend the rectangle from the position of mobile host. If it reaches the border of current domain, it should load the edge lists in other data node. LoadDomain algorithm checks the subdomain covered by the rectangle area, calls MergeList to merge these ordered edge lists. If ComputeRECT has added n queries, it returns the queries and current position of extended rectangle as the mobile host's rectangle area. Then the AssignQuery procedure completed.

```
ComputeRECT(Di, Oj)
{
    for (step=0; qcount < Oj.n; step++ )
    {       // Oj.n: capability of Oj
        min_x = min( abs(Oj.X - Di.pXL), abs(Oj.X - Di.pXR) )
        min_y = min( abs(Oj.Y - Di.pYT), abs(Oj.Y - Di.pYB) )
        stepToGo=max(min_x, min_y);
        AddtoPriorityQueue(Di.p, stepToGo) //Add query and step to PQ

        if(P_MIN<step)
        { //PQ's smallest item's key < step; Query is bounded.
            Oj.QList = Oj.PQ.pop(step); //Pop items from PQ whose key<step
            qcount++;    //qcount is the total query added to Oj
        }

        Di.pXL  <=NextElementInX_List;
        Di.pXR  <=NextElementInX_List;
        Di.pYT  <=NextElementInY_List;
        Di.pYB  <=NextElementInY_List;
        if( ( Di.pXL.< Di.XL ) or ( Di.pXR.> Di.XR) or ( Di.pYT.> Di.YT) or
                (Di.pXB.< Di.YB) )     { LoadDomain(D, Di); }
    }//End of for
}
```

```
LoadDomain(D, Di)
{
    Ds=Intersect(D, Rect(Di.pXL, Di.pXR, Di.pYT, Di.pYB) );
    //Find the Subdomains covered by Rectangle area
    MergeList(Ds, Di); //Merge all List below D.
}
```

```
MergeList(Ds, Di)
{
    if( Di.pXL.< Di.XL ) Merge( Ds.X_list, Di.X_list );
    if( Di.pXR.> Di.XR) Merge( Di.X_list, Ds.X_list );
    if( Di.pYT.> Di.YT) Merge( Ds.Y_list, Di.Y_list );
    if( Di.pXB.< Di.YB) Merge(Di.Y_list, Ds.Y_list );
    MergeSort( (Di.XL, Di.pXL), (Di.pXR, Di.XR), Ds.X_list(>Di.pXL , <Di.pXR) );
    MergeSort( (Di.YB, Di.pYB), (Di.pYT, Di.YT), Ds.Y_list(>Di.pYT , <Di.pYB) );
}
```

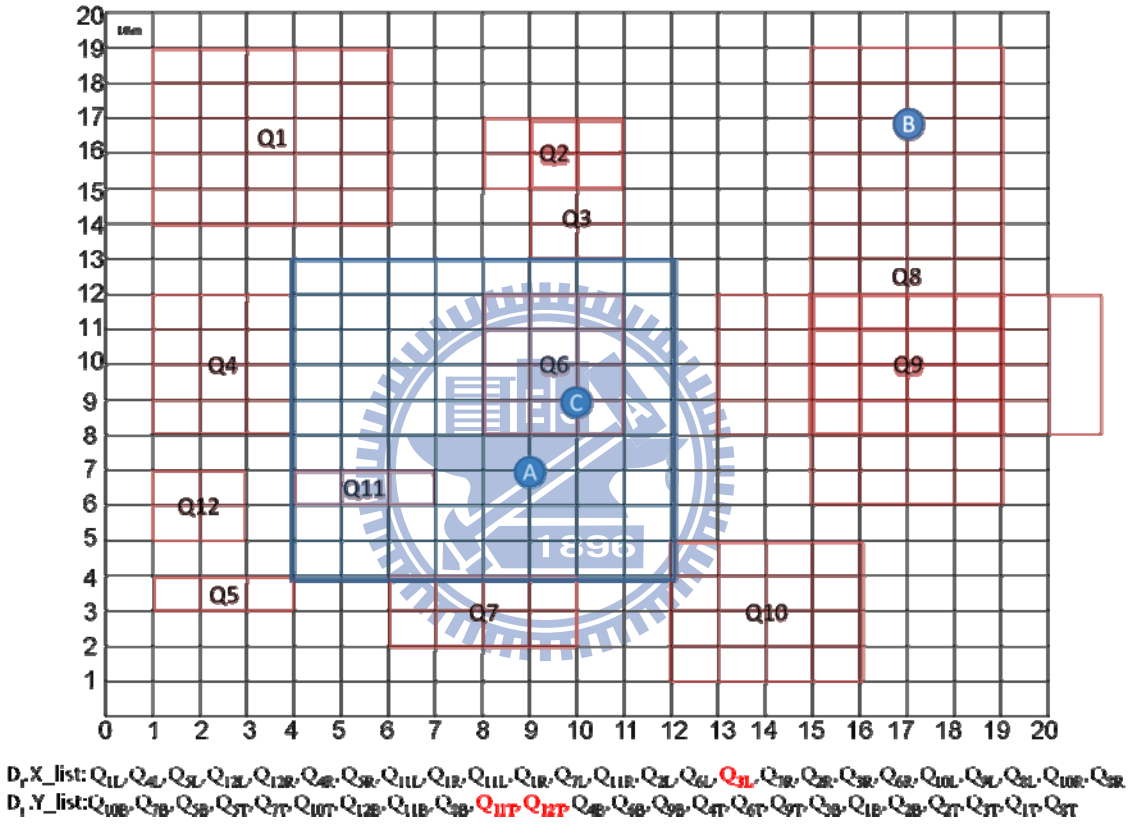

Figure 3-7 The edge lists and the rectangle area of mobile host A.

At the end of Chapter 3, we demonstrate the procedure of Query Assignment. In Figure 3-7, mobile host A's position is (9,7) with capability 2. The grids do not exist in our system. It is just to help us to see the steps clearly. At first, we found A on the edge lists. In X_list of current data node, we start at $Q_{3Left.}$ In Y_list of current data node, we start at Q_{11Top} and $Q_{12Top.}$ Table 3-1 shows the detail procedure of query assignment. The red colored words represent the data changed.

At Step 0, edges Q_{3Left}, Q_{11Top}, and Q_{12Top} are processed and their minimum steps to the four edges are compared. The maximum of these two values and its id stored to priority queue. Step 1 to Step 3 is similar. In the ComputeRECT Procedure of step 3, the MAX(X,Y) and Priority Queue's minimum key (P_MIN) is compared. The AssignQuery procedure output query list contains Q6 and Q11 to the mobile host A whose capability is 2. After pop the priority queue, the P_MIN is 3. And the pointers which point to the position of next element in X_List and Y_List are also returned as the rectangle area's boundary. We also adjust the rectangle to extend as large as possible until it is almost reach more than n queries.

Table 3-1 The Query Assignment Procedure

Step	Query	MIN STEP		MAX (X,Y)	Priority Queue	P_MIN	Query List
		X	Y				
0	Q_{3L}	**0**	6	6	$Q_{11}(2)$, $Q_3(6)$, $Q_{12}(6)$	2	-
0	Q_{11T}	2	**0**	2			
0	Q_{12T}	6	**0**	6			
1	Q_{2L}	**1**	8	8	$Q_6(1)$, $Q_{11}(2)$, $Q_7(3)$, $Q_4(5)$, $Q_8(6)$, $Q_9(6)$, $Q_3(6)$, $Q_{12}(6)$, $Q_2(8)$	1	-
1	Q_{6L}	**1**	1	1			
1	Q_{7R}	**1**	3	3			
1	Q_{8B}	6	**1**	6			
1	Q_{9B}	6	**1**	6			
1	Q_{4B}	5	**1**	5			
2	Q_{10T}	3	2	3	$Q_{11}(2)$, $Q_7(3)$, $Q_{10}(3)$,$Q_4(5)$, $Q_8(6)$, $Q_9(6)$, $Q_3(6)$, $Q_{12}(6)$, $Q_2(8)$	2	Q_6
3	Q_{1R}	3	7	7	$Q_7(3)$, $Q_{10}(3)$,$Q_4(5)$, $Q_5(5)$,$Q_8(6)$, $Q_9(6)$, $Q_3(6)$, $Q_{12}(6)$, $Q_1(7)$,$Q_2(8)$	3	Q6,Q11
3	Q_{5T}	5	3	5			

3.5 Summary

In this chapter, we present the algorithm of query insertion, query result update and query assignment algorithms in our system. We use the feature of mobile host's location and extend the rectangle area to fit mobile host's capability. The analysis of the dynamic query assignment is presents in Chapter 4.

Chapter 4 Analysis and Evaluation

In this thesis, we present a Query Result Update algorithm and a Query Assignment algorithm. In this section, we will evaluate our proposed algorithms with other methods. In the end of this chapter, we also show the simulation of our proposed algorithm.

4.1 Query Result Update Algorithm

In the query result update algorithm, we propose the rectangle area concept to reduce the number of location update. And it contains n queries for the mobile host to share the load with server. We will compare our method with Resident Domain concept which proposed by MQM.

In MQM, the subdomain is partitioned by the threshold of the less capable mobile host's capability. The computation of resident domain is down from the root of BP-Tree to the subdomain that the mobile host is in and fits its capability. When the mobile host moves out of its resident domain, it reports to the server to compute a new resident domain. When the mobile host just exits its current domain, its position is near the boundary of multiple resident domains. And the position reported by mobile hosts from GPS may have some error. The mobile host may detect that it is cross over its resident domain boundary. Figure 4-1 shows mobile host A's movement and Table 4-1 shows its position and corresponding resident domains. Because the movement of the mobile host is not always linear and predictable, the mobile host also nears the boundary of resident domain. This makes the mobile host send a lot of request of resident domain request in a short time.

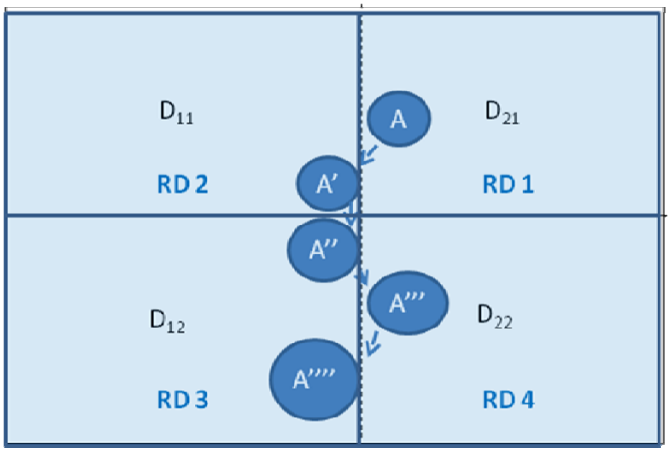

Figure 4-1 The mobile host A and its resident domain.

Table 4-1 The position and resident domain changes in Figure 4-1.

Position	Resident Domain
In D_{21}	RD 1
In D_{11}	RD 2
In D_{12}	RD 3
In D_{22}	RD 4
In D_{12}	RD 3

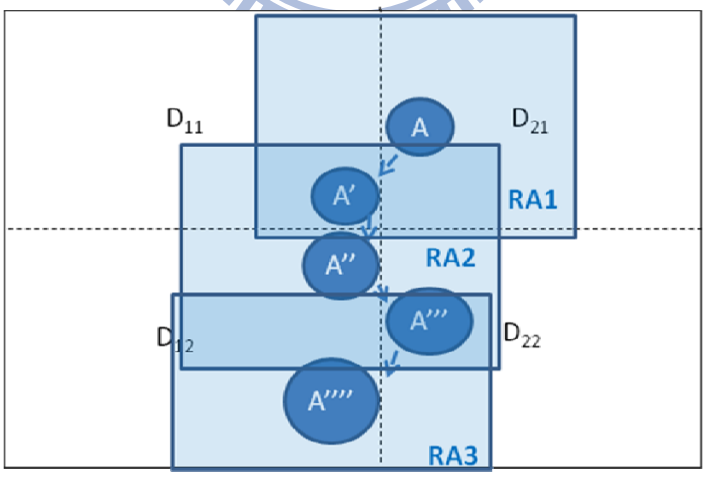

Figure 4-2 The mobile host A and its rectangle area.

In our method, we compute rectangle area based on the mobile host's position. Figure 4-2 shows the concept of our algorithm. We assume the directions the mobile host moves are random. The probability of choose each direction (North, Northeast,

East, Southeast, South, Southwest, West, Northwest) is equal. We extend the rectangle area from the position of the mobile host and try to include n queries to fit the capability of the mobile host.

4.2 Query Assignment Algorithm

The other part of this thesis is query assignment. We use BP-Tree in MQM to search for the data node that corresponding to the mobile host's position. In MQM, the server searches subdomains in BP-Tree for the mobile host as the resident domain that the mobile host is in and the queries in the subdomain may small or equal to the mobile host's capability. The split threshold ensures that every mobile host could be allocated at least one resident domain. Figure 4-2 shows mobile host A with capability 12 is in the subdomain D_{122} , and the size of D_{122} is 8. The size is smaller than mobile host A's capability. So D_{122} is assigned to mobile host A as its resident domain. The server cannot assign D_{12} to mobile host A because the size of D_{12} is 18, as we can see in Figure 4-3. D_{122} fits the mobile host A's capability, but there are better choices.

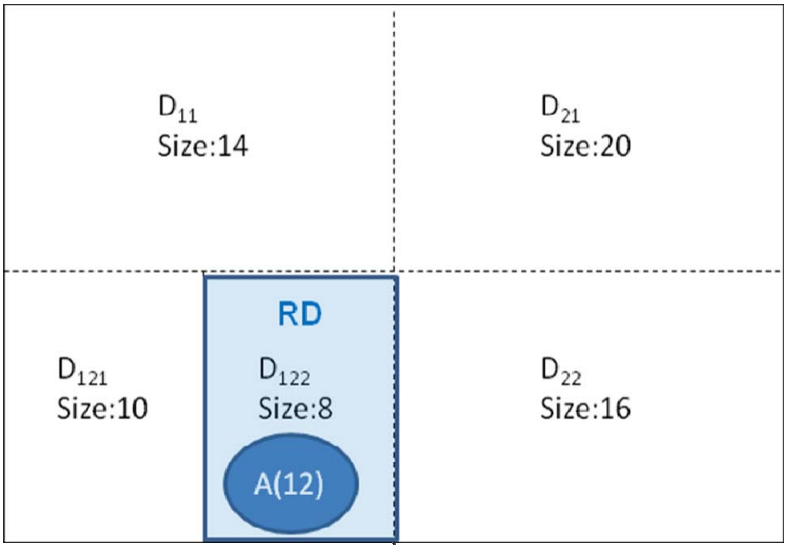

Figure 4-3 The mobile host A with capability 12 and its resident domain.

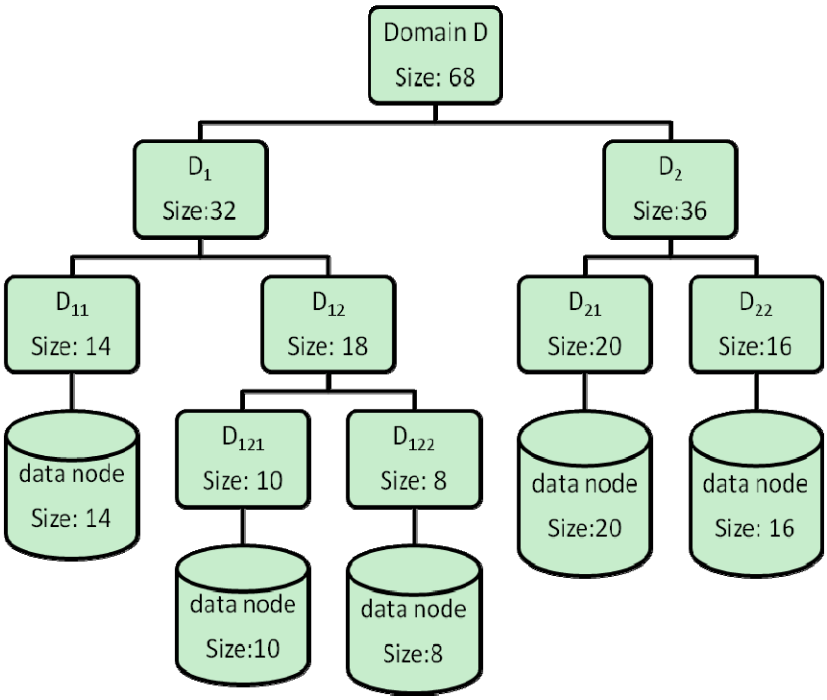

Figure 4-4 The BP-Tree of Figure 4-2.

In our system, the size of the subdomain is the number of queries in it. A query may span over multiple subdomains. Figure 4-4 shows query 1 span over D_{11}, D_{12}, D_{21}, and D_{22}. Query 2 also spans over two subdomains. Figure 4-5 shows the size in each subdomain. We consider size as the different queries in that subdomain. Table 4-2 shows that the comparison of range monitoring process algorithms.

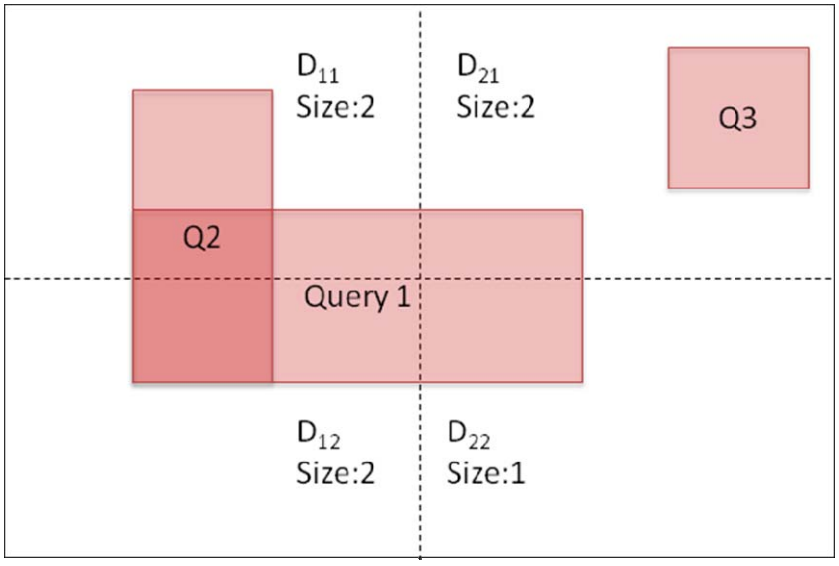

Figure 4-5 The size in subdomains.

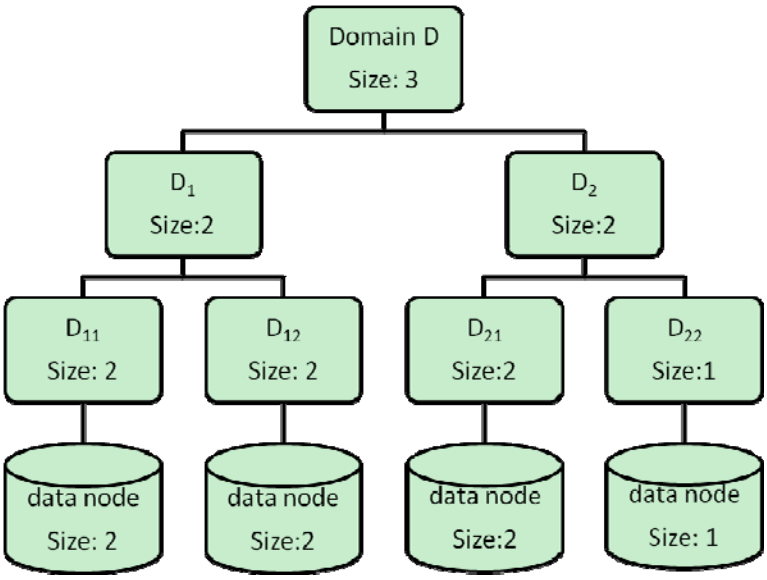

Figure 4-6 The size in BP-Tree.

Table 4-2 The comparison of algorithms

Name Property	Simple Location Update	Safe Region	Resident Domain	Rectangle Area (Proposed)
Distributed	No	No	Yes	Yes
Computing ability on MH	No	Yes	Yes(more)	Yes(more)
Memory Storage on MH	No	Yes	Yes(more)	Yes(more)
Compute a region for MH n: the number of queries	N/A	O(n) to O(nlog3n)	O(logn)	O(n) to O(nlog3n)

4.3 Simulation

In this thesis, we have implemented a simulator for MQM and our technique. We measure the number of request for resident domain messages as the scalability regard to the number of queries and the length of query rectangles. The movement types of the mobile host in our system are random and linear. Figure 4-7 shows random movement. The mobile hosts first start from a random position in the root domain. Then randomly choose eight directions to move. When it moves out of the root domain border, it restarts from origin (0, 0) and continue to move. If the step we set to move is satisfied, it stops. Figure 4-8 shows linear movement. The mobile hosts start from the origin (0, 0) to the Top-Right position of root domain rectangle.

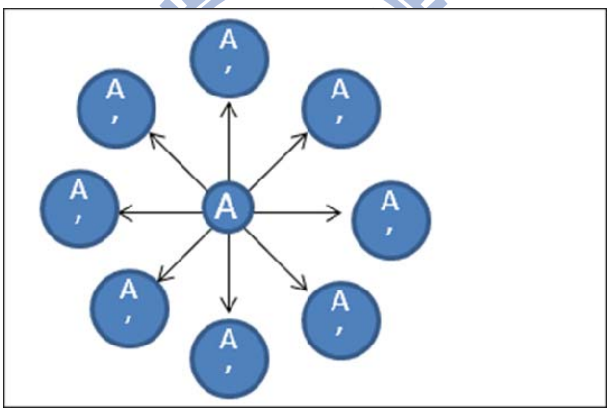

Figure 4-7 Random movement.

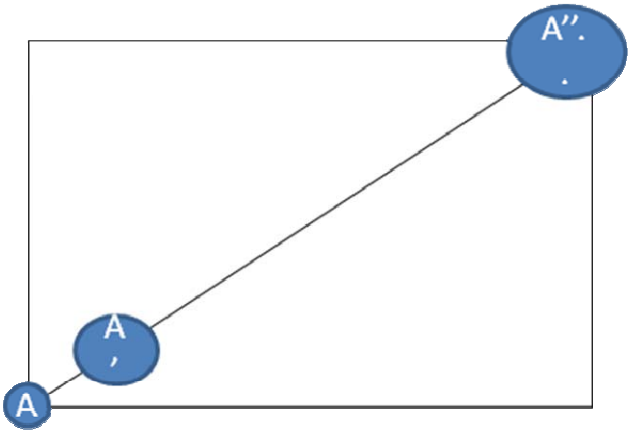

Figure 4-8 Linear movement.

Table 4-2 shows the parameters of simulation 1. Figure 4-9 and Figure 4-10 show the result of simulation 1 under different movement type respectively. Simulation 1 changes the number of queries from 1,000 to 10,000. In simulation 1(a), the mobile host start at (50000, 50000) and randomly choose a direction to go. The mobile host moves out its current resident domain and it sends a request for resident domain message to the server. The server computes the new resident domain for the mobile host. Because the movement of the mobile host is random, it can just enter the resident domain and exit at next step. In MQM, the mobile host is near the border of the new resident domain, so it may exit the current resident domain just computed and request for a new one. This makes the number of request for resident domain messages of MQM in simulation 1 much higher than our method. In contrast, as we can see in simulation 1(b), the movement is linear. In this case, MQM performs better. Because the mobile host of MQM is near the entry border of resident domain, the distance to move out the other border of resident domain rectangle is a little longer.

Table 4-3 The parameters of simulation 1.

Parameter	Value
Domain space	$2^{17}x2^{17}$
Number of mobile host	1
Capability of mobile host	50
Query square length	1000
Start point of random move	(50000, 50000)
Step of random move	500
Distance of random move	5400614
Distance of linear move	$2^{17}x \quad 2^{1/2}$

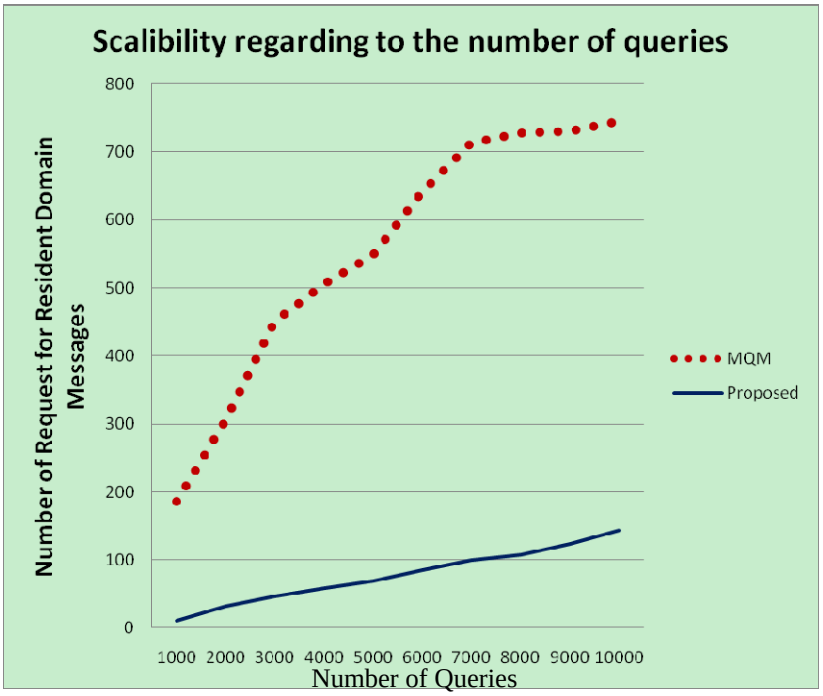

Figure 4-9 Simulation 1(a) (Random movement).

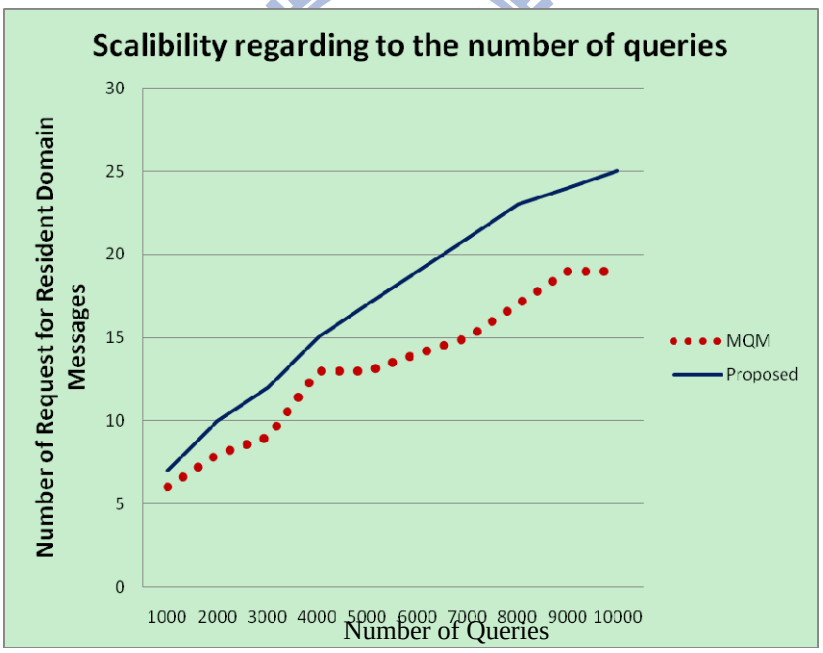

Figure 4-10 Simulation 1(b) (Linear movement).

Table 4-3 shows the parameters of simulation 2. Figure 4-11 and Figure 4-12 show the result of simulation 2 under different movement type respectively. Simulation 2 changes the length of query square from 1,000 to 10,000. In simulation 2(a), the mobile host also start at (50000, 50000) and randomly choose a direction to

go. The result in simulation 2 is similar to simulation 1. We can observe that the length of query square can also affect the number of request for resident domain message. Because the query square length can also affect the density of query in root domain.

Table 4-4 The parameters of simulation 1.

Parameter	Value
Domain space	$2^{17}x2^{17}$
Number of mobile host	1
Capability of mobile host	50
Number of Query	5000
Start point of random move	(50000, 50000)
Step of random move	500
Query length	1000~5000
Distance of random move	5400614
Distance of linear move	$2^{17}x \quad 2^{1/2}$

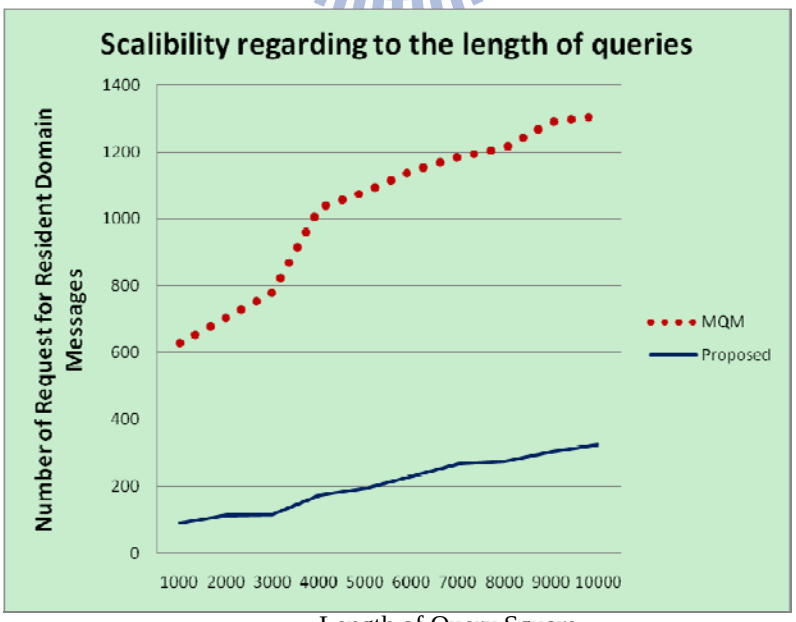

Figure 4-11 Simulation 2(a) (Random movement)

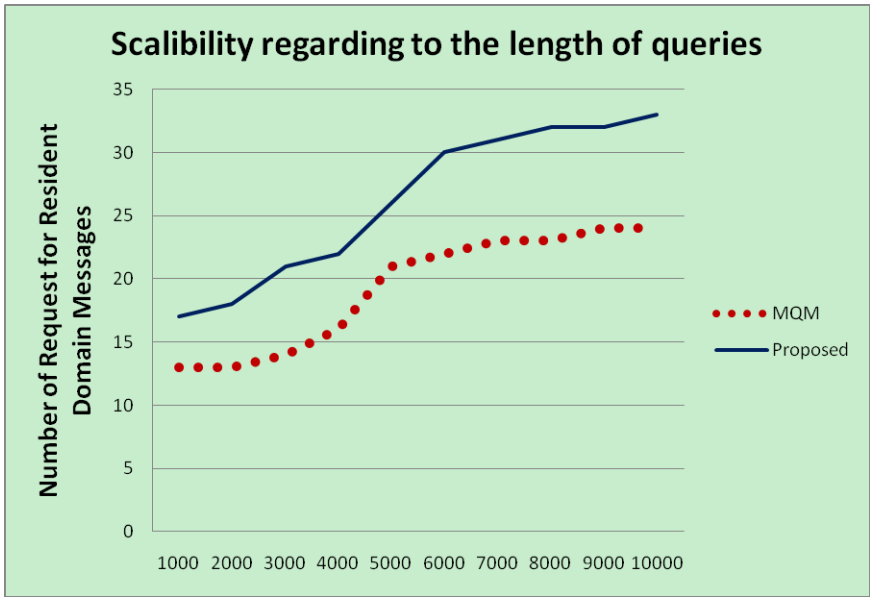

Figure 4-12 Simulation 2(b) (Linear movement)

In simulation 3, the parameter of simulation is the same as Table 4-3. Figure 4-13 and figure 4-14 show the area ratio of rectangle area and resident domain. The size of rectangle area is usually larger than the size of resident domain.

In this chapter, we analysis our method of query location update and query assignment. The mobile host's position in resident domain and its move type will affect the number of request for resident domain message.

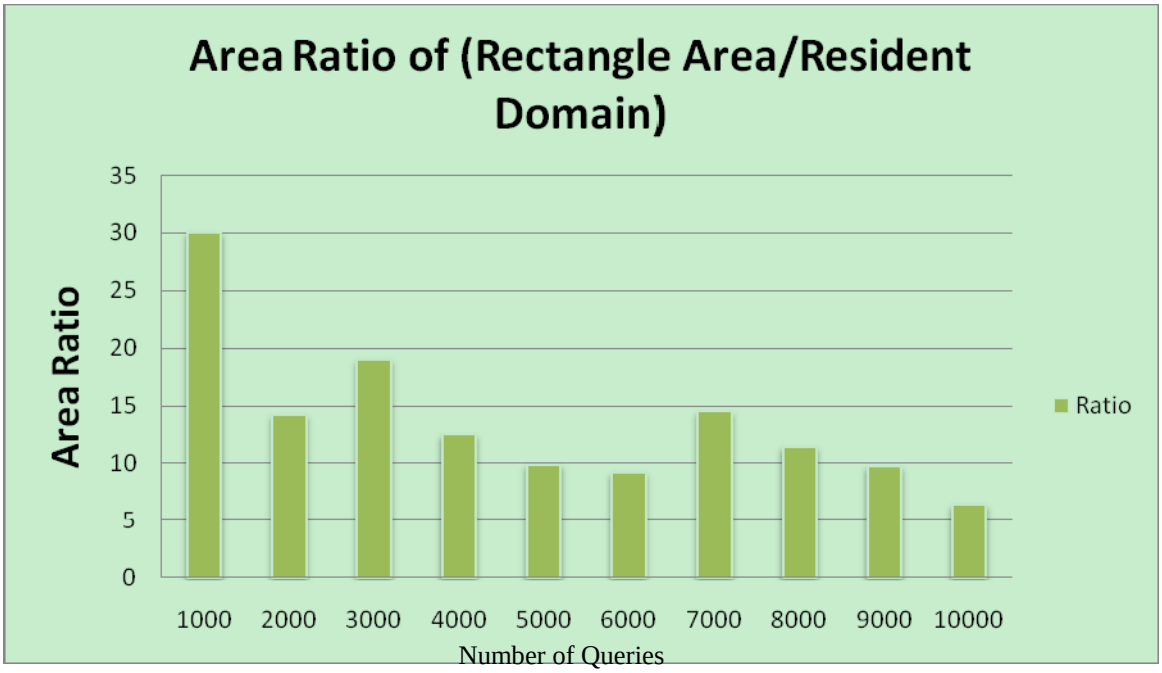

Figure 4-13 Simulation 3(a) (Random movement)

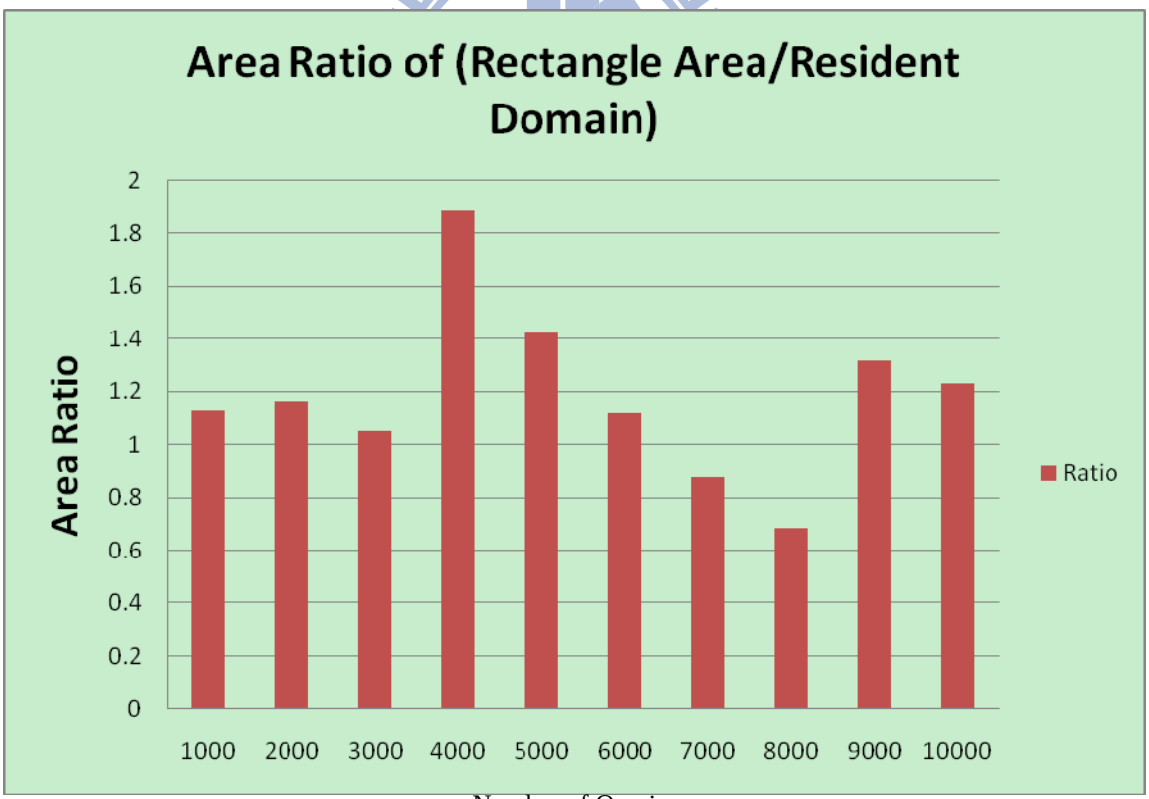

Figure 4-14 Simulation 3(b) (Linear movement)

Chapter 5 Conclusions and Future work

In this thesis, we present an algorithm for query result update and dynamic query assignment. First, query result update algorithm computes a rectangle area for each mobile host. When the mobile host is still in the rectangle area and does not affect any query, it does not need to report to the server about the query result update. Thus the communication cost is saved. Second, in dynamic query assignment, the server also assigns queries to the mobile host based on the position and capability of the mobile host.

In MQM, they proposed resident domain concept to solve query result update problem and used BP-Tree to solve query assignment problem. However, the mobile host may near the border of its resident domain. This makes the mobile host more likely move out of its resident domain. Thus the number of requests for resident domain may be huge. Our method is modified from MQM. We extend the rectangle area from the position of the mobile host. When the movement of the mobile host is random, our method performs better.

In the future, we may extend our system to support processing on moving queries over moving objects. There are many interesting applications based on the LBS. We can modify our system to support various types of queries. The scalability of the system is very important. One may change the distributed architecture to peer-to-peer architecture.

References

[1] Kupper, A., *Location-based Services: Fundamentals and Operation*. 2005: John Wiley & Sons.

[2] D'Roza, T. and G. Bilchev, *An Overview of Location-Based Services.* BT Technology Journal, 2003. 21(1): p. 20-27.

[3] Dao, D., C. Rizos, and J. Wang, *Location-based services: technical and business issues.* GPS Solutions, 2002. 6(3): p. 169-178.

[4] Zhang, J., et al., *Location-based spatial queries*, in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 2003, ACM: San Diego, California. p. 443-454.

[5] Guttman, A., *R-trees: a dynamic index structure for spatial searching.* SIGMOD Rec., 1984. 14(2): p. 47-57.

[6] Song, Z. and N. Roussopoulos, *K -Nearest Neighbor Search for Moving Query Point*. 2001. p. 79-96.

[7] Tao, Y., D. Papadias, and Q. Shen, *Continuous nearest neighbor search*, in *Proceedings of the 28th international conference on Very Large Data Bases*. 2002, VLDB Endowment: Hong Kong, China. p. 287-298.

[8] Shamos, M.I. and D. Hoey. *Closest-point problems*. in *Foundations of Computer Science, 1975., 16th Annual Symposium on*. 1975.

[9] Hu, H., J. Xu, and D.L. Lee, *A generic framework for monitoring continuous spatial queries over moving objects*, in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 2005, ACM: Baltimore, Maryland. p. 479-490.

[10] Rowstron, A. and P. Druschel, *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*. 2001. p. 329.

[11] Yu, K., et al. *A Location-Based Peer-to-Peer Network for Context-Aware Services in a Ubiquitous Environment*. in *Applications and the Internet Workshops, 2005. Saint Workshops 2005. The 2005 Symposium on*. 2005.

[12] Mokbel, M., et al., *Continuous Query Processing of Spatio-Temporal Data Streams in PLACE*. GeoInformatica, 2005. 9(4): p. 343-365.

[13] Ying, C., et al., *Real-time processing of range-monitoring queries in heterogeneous mobile databases.* Mobile Computing, IEEE Transactions on, 2006. 5(7): p. 931-942.

[14] Stojanovic, D., et al., *Continuous range monitoring of mobile objects in road*

networks. Data & Knowledge Engineering, 2008. 64(1): p. 77-100.

[15] Mokbel, M.F., X. Xiong, and W.G. Aref, *SINA: scalable incremental processing of continuous queries in spatio-temporal databases*, in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. 2004, ACM: Paris, France. p. 623-634.

[16] Gedik, B. and L. Ling, *MobiEyes: A Distributed Location Monitoring Service Using Moving Location Queries.* Mobile Computing, IEEE Transactions on, 2006. 5(10): p. 1384-1402.

[17] Prabhakar, S., et al., *Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects.* IEEE Trans. Comput., 2002. 51(10): p. 1124-1140.

[18] Six, H.W. and D. Wood, *The rectangle intersection problem revisited.* BIT Numerical Mathematics, 1980. 20(4): p. 426-433.