2009

Managing motion triggered executables in distributed mobile databases

Kihwan Kim
Iowa State University

Managing motion triggered executables

in distributed mobile databases

by

Kihwan Kim

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Ying Cai, Co-major Professor
Wallapak Tavanapong, Co-major Professor
Daji Qiao
Johnny S. Wong
Wensheng Zhang

Iowa State University

Ames, Iowa

2009

DEDICATION

I am extremely grateful to my major advisors Dr. Ying Cai and Dr. Wallapak Tavanapong for their guidance, patience, and support during the last few years. I would also like to thank Dr. Johnny Wong, Dr. Wensheng Zhang and Dr. Daji Qiao for serving on the committee and providing valuable feedbacks on this dissertation.

Over the past years, I am fortunate to have the opportunity to work with a group of talented and friendly schoolmates (DongHo Hong, Kung-En Dean Lin, Danyu Liu, Cumin Liu, Sean Stanek, Minh Tran, Yi Wang, etc) in the Department of Computer Science at Iowa State University. I enjoyed every moment that we have worked together. I appreciate the friendships and all their encouragements to finish this dissertation.

Lastly, I am forever indebted to the love and support of my mother during all these years I am far away from home.

TABLE OF CONTENTS

LIST OF TABLES v

LIST OF FIGURES vi

CHAPTER 1. INTRODUCTION 1

1.1 Concept of Geotask 1

1.2 Contributions 4

CHAPTER 2. LITERATURE REVIEW 6

2.1 Location-dependent Information Services 6

2.2 Database Management for Moving Objects 8

2.3 Data Storage and Retrieval in Distributed Wireless Networks 11

CHAPTER 3. BASIC GEOTASK MANAGEMENT 12

3.1 Overview 12

3.2 System Model 13

3.3 Geotask Management 14

3.4 Analytical Model 17

3.4.1 Cost of Retrieving Relevant Geotasks 18

3.4.2 Cost of Implanting or Removing Geotasks 21

3.4.3 Overall Cost Per Time Unit 21

3.4.4 Validation of the Analytical Model 22

3.5 Performance Study 24

3.5.1 Effect of Grid Size 25

3.5.2 Effect of Mobile Object Density 27

3.5.3 Effect of Mobile Object Movement 29
3.5.4 Effect of Geotask Update Frequency 30
3.5.5 Effect of Geotask Ranges 31
3.6 Conclusion 32

CHAPTER 4. TYPES OF GEOTASK 33
4.1 Mobile Object Correlation 33
4.2 Geotask Mobility 37
4.3 Geotask Dependency 38

CHAPTER 5. ADVANCED GEOTASK MANAGEMENT 41
5.1 Introduction 41
5.2 Handling Stationary Range Monitoring Queries 43
5.2.1 Detailed Design 43
5.3 Handling Other Types of SMQs 45
5.3.1 Mobile Range Monitoring Query 45
5.3.2 Stationary KNN Monitoring Query 47
5.3.3 Mobile KNN Monitoring Query 48
5.4 Performance Study 50
5.4.1 Effect of the Number of Mobile Objects 51
5.4.2 Effect of Mobile Object Movement 52
5.5 Conclusion 55

CHAPTER 6. CONCLUSION AND FUTURE WORKS 56

APPENDIX

SAFE-TIME: AN ALTERNATIVE APPROACH FOR GEOTASK MANAGEMENT 58

BIBLIOGRAPHY 84

LIST OF TABLES

Table 3.1 Simulation Parameters for Basic Geotasking 25

Table 4.1 Types of Geotask 33

Table 5.1 Simulation Parameters for SMQs 51

Table A.1 Simulation Parameters for Safe-Time 75

LIST OF FIGURES

Figure 1.1 Geotask Examples 2

Figure 3.1 Examples of Geotasking 12
Figure 3.2 Geotask Management 15
Figure 3.3 Possible Regions for a Geotask's Center Position 22
Figure 3.4 Validation of the Analytical Model 23
Figure 3.5 Effect of Grid Size 26
Figure 3.6 Effect of Mobile Object Density 28
Figure 3.7 Effect of Mobile Object Movement 29
Figure 3.8 Effect of Update Frequency 30
Figure 3.9 Effect of Geotask Area Size 31

Figure 4.1 Safe Boundary 35
Figure 4.2 Converting m-class to s-class 36
Figure 4.3 Inner and Outer Safe Boundaries 38
Figure 4.4 Converting Mobile Geotask to s-class Geotasks 39
Figure 4.5 Interrelated Geotasks 39

Figure 5.1 Safe Boundaries for an M-RMQ 47
Figure 5.2 Safe Boundary for an S-KNNMQ 48
Figure 5.3 Safe Boundaries for an M-KNNMQ 49
Figure 5.4 Effect of the Number of Mobile Objects 53
Figure 5.5 Effect of Mobile Object Movement 54

Figure A.1 Safe-Time Algorithm: Distance from the Query Point 61
Figure A.2 Circular Band Broadcast Area in a 2D Space 64
Figure A.3 Relationship among the Query Point, k^{th}, and $k + 1^{th}$ NNs 67
Figure A.4 Two Overlapping Critical Circles 68
Figure A.5 Extend-Safe-Time 72
Figure A.6 Shaded Triangle Shows the Condition when o_3 and o_4 Changing Their Order. 73
Figure A.7 Effect of Number of Queries 76
Figure A.8 Effect of Number of Mobile Objects 78
Figure A.9 Effect of Mean Speed of Mobile Objects (Maximum Speed Is Fixed at 4 m/sec) 80
Figure A.10 Effect of Maximum Speed of Mobile Objects 81

ABSTRACT

Mobile devices have brought new applications into our daily life. However, efficient management of these objects to support new applications is challenging due to the distributed nature and mobility of mobile objects. This dissertation describes a new type of mobile peer-to-peer (M-P2P) computing, namely *geotasking*, and presents efficient management of mobile objects to support geotasking. Geotasking mimics human interaction with the physical world. Humans generate information using sensing ability and store information to geographical locations. Humans also retrieve this information from the physical locations. For instance, an installation of a new stop sign at some intersection in town is analogous to an insertion of a new data item into the database. Instead of processing regular data as in traditional data management systems, geotasking manages a collection of geotasks, each defined as a computer program bound to a geographical region. The hardware platform for geotasking consists of popular networked position-aware mobile devices such as cell phones, personal digital assistants, and laptops. We design and implement novel system software to facilitate programming and efficient management of geotasks. Such management includes inserts, deletes, updates, retrieval and execution of a geotask triggered by mobile object correlations, geotask mobility, and geotask dependency. Geotasking enables useful applications ranging from warning of dangerous areas for military and search-and-rescue missions to monitoring the population in a certain area for traffic management to informing tourists of exciting events in an area and other such applications. Geotasking provides *a distributed and unified solution* for supporting various types of applications.

CHAPTER 1. INTRODUCTION

1.1 Concept of Geotask

Our physical world can be seen as a natural computing system. Each individual person is like a mobile computing unit which moves freely on the ground, processes a sequence of jobs, or decides one solution among several choices. The ground is like a hard disk which contains valuable information such as a historical place or stop sign. A person moves on the surface of this natural disk and saving information to this disk or retrieving information from this disk to execute this information. Installing a stop sign at some intersection in town can be seen as saving a new program on the ground. The program is a set of instructions bounded to a geographic region for a person to follow. As we approach this intersection, the stop sign instructs a person to stop before crossing the intersection. A person (mobile computing unit) communicates with other persons, cooperates with them, and gets information from them. For example, getting a friend's current place requires communication with the friend to get his/her position. Since a person can move from one place to the other place, getting a friend's current place requires periodic asking the friend his/her current position or periodic reporting his/her position to a requestor.

This dissertation investigates a new type of mobile peer-to-peer (M-P2P) computing, namely *geotasking*, which mimics the above natural computing. The hardware resource of a geotasking system is simply a set of position-aware mobile objects like many of today's cellular phones. These mobile objects are connected to each other by means of wireless networks, and together they facilitate the system with computing and storage capabilities. The main feature of a geotasking system comes from its executables, each being a *special program called a geotask*. Unlike regular program, a geotask is associated with a user-defined geographical

area (referred to as a binding region), and its execution is triggered by movements of mobile objects.

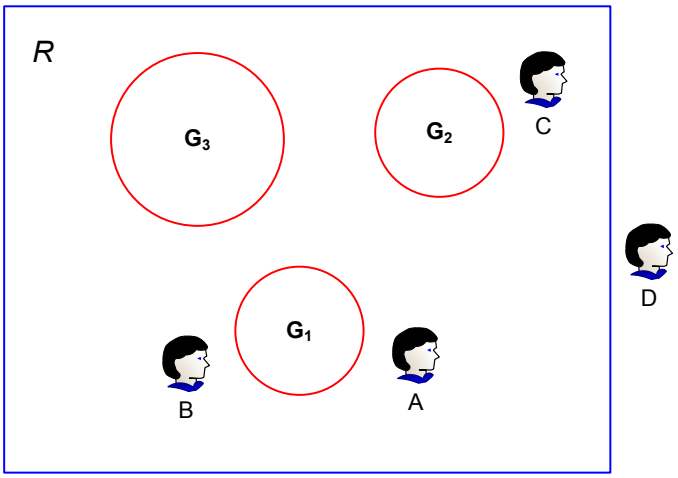


Figure 1.1 Geotask Examples

Figure 1.1 shows three geotasks implanted in a battlefield. When a soldier moves into the region where G_1 is bound, his handset needs to execute the geotask, thus notifying his commander of his arrival automatically. Geotask G_2 is implanted by a soldier who finds mines in his area. As soon as a soldier moves into the area, he will then be alerted with a warning message. The execution of these two geotasks is triggered when a mobile object moves into a certain geographic region. The third geotask G_3 is implanted to request help from the nearest soldier. In this case, G_3 is executed by a soldier when he becomes closest to the site.

Geotasking is conceptually similar to existing computation models where a set of processors shares on hard disk. However, here the processors are mobile devices and the storage disk is our physical ground – all programs (i.e., geotasks) are virtually stored on the ground, and in particular, their execution is triggered by the movement of the processors. To our best knowledge, the concept of geotasking has not been investigated in the literature. This dissertation explores the scope of geotasking applications and develops a cost-effective platform that supports all these applications. We classify geotasks along three dimensions:

- *Mobile object correlation:* Geotasks can be classified based on whether or not the movement of mobile objects is correlated in triggering geotask execution. We say a geotask

is an *s-class* geotask if its execution is triggered by the movement of each mobile object independently. Otherwise, the geotask is an *m-class* geotask. Both G_1 and G_2 in Figure 1.1 are s-class. To determine when to execute them, a mobile object just needs to monitor its own movement with respect to their binding regions. In contrast, G_3 is an m-class geotask. In this case one mobile object's movement may cause another mobile object to execute G_3. To determine whether it is now nearest to G_3, a mobile object would need to know the position of other mobile objects.

- *Geotask Mobility:* Geotasks can also be classified based on their mobility. A stationary geotask is fixed at one place during its life time, while a mobile geotask may keep changing its binding region. For instance, a geotask bound on a vehicle moves as the vehicle moves. An example of such a geotask is an emergency vehicle that demands other vehicles to yield.

- *Geotask Dependency:* Another way to classify geotasks is based on their dependency. A geotask is *independent* if its execution is determined by the movement of mobile object(s) with respect to the geotask itself. The three geotasks in Figure 1.1 are all independent. A geotask G is *dependent* on another geotask G' if the execution of G is determined by the movement of mobile object(s) with respect to both G and G'. For example, we may want a mobile object to execute G whenever its distance to G is less than its distance to G'.

The three-dimension classifications create eight categories of geotasks – a geotask can be s-class/m-class stationary/mobile independent/dependent. To enable geotasking, the key issue is geotask management. A straight-forward implementation is to use a central server to manage all geotasks, and let mobile objects periodically report their position to the server. When the server detects that the execution of a geotask is triggered, it sends the geotask to a corresponding mobile object for execution. This approach presents a single point of failure and may require frequent location updates from mobile objects to ensure real-time execution of geotasks. Relying on a central server also prevents geotasking from being used in sev-

eral application scenarios (e.g., military actions, children camping) where such a server is not available.

This dissertation explores geotasking in the context of a M-P2P networking environment where mobile objects are the only computing devices. In the absence of any central server, efficient geotask management is challenging. The geotasks must be stored among mobile objects, and loaded for execution as soon as triggered by the movement of mobile objects. Different geotasks also demand different management techniques. As a part of this dissertation, we have devised a novel technique for efficient management of s-class stationary independent geotasks, which we refer to as *basic geotasks*. While our scheme allows the geotasks to be executed as soon as their triggers are fired, it minimizes the management costs by storing geotasks among mobile objects dynamically to keep them close to where they are implanted. We have analyzed the performance of this technique with a detailed mathematical model and verified its accuracy through simulations. We extended basic geotasks to support other categories of geotasks. Our research goal is develop a generic geotasking platform for the development and deployment of all types of geotasks.

1.2 Contributions

Contributions of this dissertation are following.

1) Geotasking provides *a new type of computing in M-P2P*. Our geographic environment is a natural storage disk where we, humans store and share data; and wherever we go, we load and process data either consciously or subconsciously, the data implanted around us. Existing computation models have not yet captured the essence of this natural data processing. Geotasking provides a natural solution for persistent storage and efficient execution of geotasks in a distributed manner.

2) Geotasking provides *a distributed and unified solution* to handle various types of queries. Geotasking supports various queries such as a range-monitoring query and a nearest neighbor query. Existing techniques rely on one or more central/stationary severs in order to support such queries; different types of queries also require vastly different solutions. Geotasking

supports a wide range of queries under one platform in a distributed manner.

3) We provide the basic geotask management platform. We partition a network area into disjoint grids. Each mobile object stores relevant geotasks and executes these geotasks when some conditions are satisfied. The basic geotask management platform supports several types of geotasks such as s-class/m-class and stationary/mobile geotasks.

The rest of the chapters are organized as follows. Chapter 2 provides discussion of the state of art. In Chapter 3, the basic geotask management platform using the grid approach is presented. In Chapter 4, types of geotask are discussed. In Chapter 5, an advanced geotask management platform is discussed to support various types of geotask management. Finally, conclusion and description of future work are presented in Chapter 6. Appendix provides an alternative geotask management platform.

CHAPTER 2. LITERATURE REVIEW

A geotasking system is formed by a set of position-aware mobile objects that can communicate with each other through wireless networks.[1] Without assuming any stationary server, these mobile objects collaborate in data storage and processing. Database management systems assuming such a M-P2P environment have been investigated in some aspects (e.g., transaction management [10, 15, 16], skyline query processing [20]). To our best knowledge, no existing work has considered the management of motion-triggered executables, i.e., geotasks. From the perspective of some of its potential uses, geotasking is relevant to location-dependent information services and query processing in moving object database management. Existing research investigates these two types of services separately and generally assumes a centralized environment with the presence of stationary servers. In contrast, geotasking presents a distributed and unified solution. From its implementation perspective, geotasking is related to data storage and retrieval in M-P2P networks. We discuss these works as follows.

2.1 Location-dependent Information Services

A data item is location dependent if its value is determined by the location to which it is related [58]. Examples of such data include local yellow pages, local events, hotel and restaurant information. Location-dependent information services refer to services that answer queries based on where the queries are issued [87]. One type of work in this area aims at real-time delivery of location-dependent information, in which a message pops up on a mobile object's screen as soon as the mobile object moves into a specific place. Existing systems for

[1] The mobile objects can form a mobile peer-to-peer network (M-P2P) by themselves and communicate through packet relaying using protocols such as LAR [30], DREAM [59], GPSR [27] and so on. It is worth mentioning that some applications (e.g., PeopleNet [46]) have been proposed recently on top of a large-scale M-P2P network.

such services include ActiveCampus [14], and GeoNotes [51]. Other research works investigate Location-dependent queries (LDQs) such as range query (e.g., "retrieve the hotels within one mile") and K-Nearest Neighbors (KNN) search (e.g., "find me three nearest hotels"). At the core of location-based applications, efficient processing of LDQs has been investigated intensively in the context of cellular networks, where one or more stationary servers are used as information repositories. In general, a LDQ can be processed on demand or by broadcast, and client caching can be applied in each mode to improve performance [37]:

- In on-demand access, a mobile client submits a request, associated with a location, to a server. To minimize disk I/O incurred in query processing, the server usually indexes the information using spatial data structures such as R-tree [17].

- In broadcast mode, the server broadcasts the information periodically without explicit requests from clients. A client handles its own query by tuning into the broadcast channel and filtering out the data according to the query. A major challenge here is to minimize client access latency and their battery consumption. A number of techniques (e.g., [80], [87], [86], [38]) have been proposed for this purpose. These schemes extended the early concept of air indexing [22, 23] to support location-dependent queries in broadcast environment.

- Caching location-dependent data at mobile clients for future LDQs was investigated in [58, 83, 82, 79, 41, 19] and more recently in [35, 34]. While other schemes assume a client caches data just for its own use, the techniques in [35, 34] allow clients to share their caching results. That is, a client can check its nearby clients for query results before submitting its query to a server.

An LDQ is a continuous query if it stays active for some time period. A continuous LDQ is often associated with a mobile object's current position. As the mobile object moves, the query results keep changing and require update. Two indexing techniques were proposed in [64] and [67] for on-demand processing of continuous range queries and nearest neighbor search, respectively. Continuous search of nearest neighbor in broadcast environment was investigated

in [84, 85]. These works are different from a geotasking system. First, they all rely on a central server for message management. To determine if a message needs to be delivered, the server has to track every movement of mobile objects. Frequent location updates will not only quickly drain the battery of mobile objects, but also create both communication an processing bottlenecks at the server side. Second, the existing systems associate a message with a point location. A mobile object receives a message when its distance to the message is less than a threshold, which is fixed at system wide. In contrast, geotasking associates a message with a region that can be user-defined. In particular, geotasking implements each message as a program, and thus allows much more flexible messaging.

2.2 Database Management for Moving Objects

A moving object database management system records the movement of mobile objects and allows users to track and query these objects, say, retrieving all mobile objects within a geographic region at some time point. Such queries are also location-dependent; however, they retrieve mobile objects and are therefore different from the LDQs discussed in the previous subsection which query stationary objects such as hotels. Existing research assumes a centralized architecture where all mobile objects report their location to some stationary server, and most of the work has focused on two problems that are interrelated, i.e., minimizing mobile communication cost and server processing cost. Excessive location update from mobile objects will not only exhaust their battery quickly, but also create a processing bottleneck at the server side when a large number of mobile objects are involved. To avoid continuous location updates, an effective solution is location estimation [73, 74, 36]. The idea is to let a mobile object report its current position and velocity and update the server with such information only when the deviation between the object's actual location and the estimated location exceeds some threshold. At the server side, the approximated moving trajectory of mobile objects can then be indexed for efficient query processing. Many spatial-temporal indexing techniques have been proposed for this purpose. These schemes allow users to query either the past positions of mobile objects [52, 33, 53, 66, 18], or their movements in near future [1, 61, 2, 60, 68, 25, 49],

with a few exceptions that are capable of both [65, 40, 50].

LDQs over mobile objects can also be continuous and various types of such queries have been studied. One is *range-monitoring queries*. Such a query requires retrieving the mobile objects that are currently inside a geographic region and, before the query is terminated, providing real-time and continuous update as the objects move into and out of the region. The techniques that support range-monitoring queries include *Q-index* [54, 77] and MQM technique [7, 8, 9]. Another type of continuous queries is *moving range queries* studied in [12, 45, 13]. An example of such queries is "retrieving all mobile objects that are within one mile of my current position." As the user moves, the query region changes accordingly. Thus, in addition to the mobility of mobile objects, the movement of the user who issues the query may also cause the query results to change. Finally, continuous queries for neighbor monitoring was studied in [81, 78, 47]. Continuous K nearest neighbors (cKNN) query allows one to continuously retrieve moving objects that are nearest to a reference position. Studies in [47, 81, 78, 64, 39, 24] focused on efficient methods of cKNN queries by tracking the change of k nearest mobile objects. Most of these approaches assume that every mobile object updates its location repeatedly to a centralized server. The server can learn the trajectory of a mobile object and to process a cKNN query. However, due to the location update interval, the result of cKNN can be changed between two consecutive location update period (i.e., the query processing might fail to report a correct result of a cKNN query). A study in [64] proposed an approach in which KNN query is reevaluated at each time when the moving query object location is updated. Works in [47, 78, 81] extended the approach of [64] to reduce communication cost by skipping some KNN query evaluation using positions of mobile objects.

We focus on the two distributed solutions for cKNN as they are most relevant to our work. Both solutions support moving queries, but also work for stationary queries. Wu et al. [70] assumes a cellular network with base stations. Their solution consists of two phases. In the initial phase, a snapshot KNN of an initial location of the query is computed. The query location, the location of the k^{th} NN (termed *critical object*), velocity vectors (speed and direction) of the moving query and the critical object, and the query result are broadcast to

the mobile objects in relevant cells (which we call a monitoring area hereafter). In the second phase, the mobile object in the monitoring area determines *critical distance* defined as *the distance from the current location of the query to the critical object.* If the mobile object is in the query result but its distance to the query point has become larger than the current critical distance, this object could become a candidate critical object. On the other hand, if the object is not in the query result and its distance to the query point becomes smaller than the current critical distance, this object could also be a critical object. The object cannot be sure whether it is the new critical object or not because it does not know the velocity vectors of other objects. Hence, candidate critical objects report to the server. The server selects the correct critical object. It then broadcasts the velocity vector of the critical object to the monitoring area. The server also broadcasts any changes in the velocity of the query object and the critical object.

Liu et al. [42] investigate cKNN for road networks. In this environment, object distance depends on the structure of the road segments. Hence, the monitoring area is computed accordingly. For each query object, the server initially computes the query result and the monitoring area. The server sends the information about the query object and the query result to only objects moving in the monitoring area. These mobile objects inform the server in the following conditions. 1) The object was not in the query result, but it moves closer to the query object than the k^{th} NN in the current query result. 2) Other objects in the query result move further away from the query object than the k^{th} NN. The server recomputes the query result and the process repeats.

Both the solutions have been shown to reduce server load and communication cost compared to a centralized approach. Nevertheless, the server is still very involved in receiving reports for the new k^{th} NN candidates, computing the new k^{th} NN, and updating velocity of the k^{th} NN to all mobile objects in the monitoring area. Hence, it is not most favorable for M-P2P since message relay among mobile objects is the means for communication in that environment.

Existing work for moving object management focuses on one or few types of queries. Different types of queries require different solutions. However, geotasking has the potential to

provide a unified solution for the processing of such queries.

2.3 Data Storage and Retrieval in Distributed Wireless Networks

Efficient data storage and retrieval have been studied intensively in distributed wireless environments. The techniques most related to this research are those investigated in [57, 56, 6, 76, 46], and we will hereafter refer to them as *location-based storage*. Despite vastly different application scenarios, these schemes share a common feature: they all map a data item into a geographic location and store the data at the mobile/stationary object(s) near to the location. The technique proposed in [57, 56] was intended for wireless sensor networks. In this scheme, each data item is associated with a name, which is hashed into a location that is within the boundary of the network domain. A data item is stored at the sensor that is nearest to the data's corresponding hash location. The techniques in [6, 76] adopt a similar idea for location services in M-P2P networks. These schemes associate each mobile object with a virtual home region (VHR), a circular area that is centered on a fixed location (e.g., generated by hashing a mobile object's ID). When a mobile object moves, it periodically geocasts its current location to its VHR, and the mobile objects inside the VHR are responsible for storing this location information. In this way, the mobile objects within a mobile object's VHR act as its position servers, from which other mobile objects can obtain its current position. The technique investigated in [46] was for information sharing among the users in cellular networks. The proposed scheme partitions the network domain into a fixed number of non-overlapping regions called bazaars, each composed of several cells. Each bazaar handles some specific types of data. That is, the mobile objects within a bazaar are responsible for caching the bazaar's related data and answering queries related to these data. Since the location of each bazaar and the types of data it handles are known to all mobile objects, searching for a specific type of data is to simply route the query to the mobile objects in the related bazaar. While the techniques mentioned above provide cost-effective solutions for data storage and retrieval in mobile environment, they do not guarantee storage persistency: a data item will be lost if its corresponding VHR or bazaar has no mobile objects.

CHAPTER 3. BASIC GEOTASK MANAGEMENT

3.1 Overview

A geotasking system is simply formed by a set of position-aware mobile objects, without assuming any stationary server. These mobile objects are connected to each other by means of wireless networks, and together they facilitate the system with computing and storage capabilities. Each data item managed in the geotasking system is *a special program called a geotask*. Unlike regular programs, each geotask is associated with a geographical region and its execution is triggered by movements of mobile objects. As mentioned early, geotasks are classified into three dimensions which are mobile object correlation, geotask mobility, and geotask dependency. We focus on a basic geotask management in this chapter. (i.e., s-class, stationary, and independent geotask management). We will use term "geotask" as "basic geotask" in this chapter.

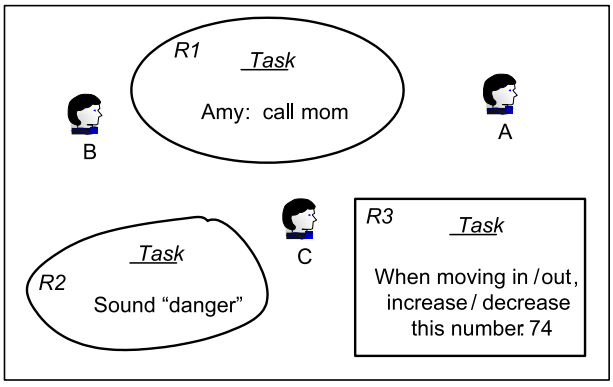


Figure 3.1 Examples of Geotasking

Figure 3.1 shows examples of geotasking. For instance, a geotask can be implanted in Amy's school; as soon as Amy arrives at the school, her cellular phone will automatically

notify her mom as instructed by the geotask. A soldier can leave a geotask in a dangerous area (e.g., one containing mines) to sound a warning when other soldiers approach. Similarly, a brake sensor detecting an icy patch can automatically leave a geotask on the road to alert every oncoming vehicle. We can also define a geotask that increases/decreases the value of a counter when a mobile object moves into/out of an area to monitor the traffic conditions within that area. Data management systems like geotasking have not been investigated in the literature. Geotasking manages *programs (geotasks)* instead of *data* as in traditional databases. The programs are associated with triggers. Unlike typical database triggers whose execution is triggered by inserts, updates, or deletes of data items [55], the execution of geotasks is triggered by movements of mobile objects. A straight-forward design of geotasking is to use a central server to manage all geotasks, and let each mobile object periodically report its position to the server. This approach presents a single point of failure and may require frequent location updates from mobile objects to ensure real-time execution of geotasks. In particular, relying on a central/stationary server prevents geotasking from being used in application scenarios (e.g., military actions, children camping) where such a server is not available. This project is to address the challenges of enabling geotasking and explore its potential in a M-P2P environment where the system computing and storage capabilities are facilitated by mobile objects themselves.

3.2 System Model

Our research assumes a fully distributed system which we will refer to as a M-P2P networking environment. The system is made up by a set of position-aware mobile objects such as GPS-enabled PDAs or cellular phones. Without causing ambiguity, we will use the terms mobile objects and users interchangeably. We assume the mobile objects can communicate with each other through wireless networks. Before proceeding, let us examine more carefully what it means for these mobile objects to communicate through means of wireless. The mobile objects can form a M-P2P network by themselves. In this case, the communication among the mobile objects is done through packet relaying; and many efficient protocols (e.g.,

LAR [30], DREAM [59], GPSR [27]) have been developed for this purpose. Alternatively, the mobile objects may take advantage of fixed infrastructures, if available, and use them as a communication backbone. For instance, a set of wireless access points can create a *mesh network* [44, 63, 28, 11], allowing mobile objects to communicate as long as they are within the joint coverage of these access points. A mesh network can cover a very large geographic terrain[1], and therefore may accommodate a large number of mobile objects. In any case, we simply assume the existence of communication primitives such as broadcast and geocast [48][32] and will not concern ourselves with how they are actually implemented.

3.3 Geotask Management

Geotasking manages *geotasks*. Each geotask has three attributes: program, binding region, and execution trigger. A program is a set of instructions that are executable to mobile objects. Without loss of generality, we assume each geotask is a serializable Java object that includes a *task*() method. A mobile object executes a geotask by calling its *task*() method. Each geotask is bound to a geographic area and associated with an execution trigger that specifies how it needs to be executed. In this paper, we consider only a simple *step-in* trigger – a mobile object executes a geotask when the object moves into the region where the geotask is bounded.

Geotasking maps our physical environment into a database; as mobile objects move, they load and execute the geotasks implanted in their surroundings. From the users' perspective, the storage device of a geotasking system is the physical environment and one can store programs in any area. In reality, however, the physical storage is facilitated by mobile objects. Thus, a major challenge is geotask management, i.e., how the mobile objects can cache the geotasks and retrieve them for timely execution. As a simple solution, one can replicate each geotask among all mobile objects. When a mobile object moves, it checks its movement against each geotask and when moving into its binding region, launches a new thread to execute it. This approach is obviously not scalable and may incur significant communication overhead. Whenever a geotask is implanted or removed, a network-wide broadcast is required in order to update all mobile

[1]The entire town of Daytona Beach Shores, FL. is now covered by mesh networks [72]. Some predict that mesh networks may replace the existing cellular infrastructure in near future [62][43].

objects.

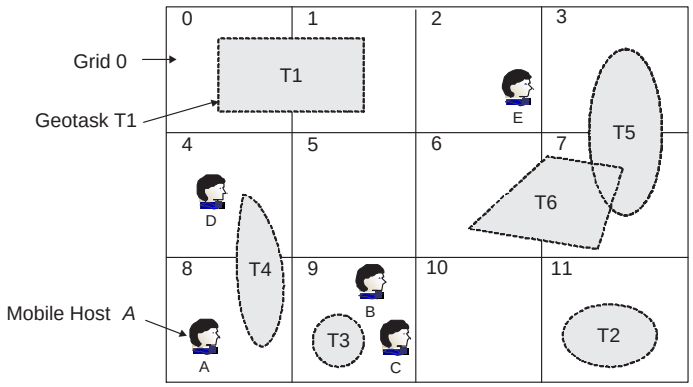


Figure 3.2 Geotask Management

For cost-effective geotasking, we propose to partition the network domain into a set of disjointed grids (or subdomains), each having an equal size, as showed in Figure 3.2, and such partitioning is made known to all mobile objects. We will model and analyze the factors in choosing grid size in the next section. We say a geotask and a grid are *relevant* to each other if the range of the geotask overlaps with the grid. A geotask may be relevant to more than one grid since its range may span over several grids. For instances, geotask T_3 is relevant only to grid 9 while T_5 is relevant to both grids 3 and 7. We also say a grid is a *home grid* to a mobile object if the mobile object is in the grid. Each mobile object caches all geotasks that are relevant to its home grid and does the following as it moves:

- When the mobile object moves inside its home grid, it checks its movement against each relevant geotask and executes it when necessary.

- When a mobile object is about to move out of its home grid, it checks if it is the last mobile object in the grid. If there are some other mobile objects in the grid, the mobile object can simply delete the geotasks that are relevant to this grid. Otherwise, it keeps these geotasks which will be retrieved by other mobile objects when they move into the grid. In this case, the mobile object is called the grid's *retaining mobile object*. For example, if mobile object A in Figure 3.2 is moving out of grid 8, it will become the

retaining mobile object to this grid. Note that a grid can have at most one retaining mobile object; and this happens only when there is no mobile object inside the grid.

- When a mobile object moves into a new grid, it needs to retrieve the grid's relevant geotasks which can be provided by any other mobile objects in the grid. For instance, if mobile object A in Figure 3.2 moves into grid 9, it can get T_3, the geotask relevant to grid 9, from either B or C. It is possible that the grid does not have any other mobile object. In this case, the mobile object searches the grid's retaining mobile object for the relevant geotasks. The search starts from the neighboring grids and then sequentially expands until the retaining mobile object is located. The retaining mobile object can delete the relevant geotasks after delivering them to the searching mobile object.

Installing a geotask is done by first determining its relevant grids and then geocasting [48, 32] the geotask to these grids. For instance, to implant geotask T_4 in Figure 3.2, a user sends it to all mobile objects in grids 8 and 4. A relevant grid may not have any mobile object. In this case, we locate the grid's retaining mobile object using the approach discussed earlier and send the geotask to this mobile object. Similarly, removing an existing geotask can be done by notifying all mobile objects in its relevant grids to delete the geotask from their caches. If a relevant grid does not have any mobile object, then its retaining mobile object is located and notified instead.

In the above technique, each geotask is cached by either the mobile objects in its relevant grid(s) and/or a grid's retaining mobile object. Thus, it guarantees storage persistency. This scheme also supports the timely execution of geotasks since each mobile object is made aware of the geotasks relevant to its home grid. As a mobile object moves, it can monitor its movement against the regions bound with the geotasks it caches and execute one whenever necessary. When a mobile object moves into a new grid, it can retrieve the grid's relevant geotasks from any other mobile objects in the grid. However, when the grid has no other mobile objects, it would need to search for the grid's retaining mobile object. This cost can be reduced by keeping a grid's retaining mobile object close to the grid. Suppose mobile object H is the retaining mobile object to grid i, and H is moving from its current grid j into a new grid

k. If the distance from k to i is farther than that from j to i, then H can send the geotasks that are relevant to grid i to another mobile object, if any, in grid j. This mobile object then becomes the new retaining mobile object to grid i. As an example, suppose mobile object A in Figure 3.2 moves from grid 8 to grid 9, and then to grid 10. If A is the retaining mobile object to grid 8, it may unload the grid's relevant geotasks to either B or C. Thus, a grid's relevant geotasks can be stored within the grid or as close as possible to the grid.

3.4 Analytical Model

The above geotask management partitions the network domain into grids. The grid size has profound impact on the overall system performance. To one extreme, the entire network domain can be one grid. In this case, each geotask is replicated among all mobile objects. Whenever a geotask is installed or removed, a network wide broadcast is needed. On the other hand, setting grid size too small can also result in overwhelming communication cost because a mobile object may move in and out of grids very frequently; and whenever this happens, the mobile object needs to query for new relevant geotasks. In this section, we analyze the impact of grid size on the system performance.

Suppose the grid size is $d \times d$ and the network domain $h \times w$ is partitioned into M grids. For simplicity of analysis, we assume every mobile object has the same transmission range r and the network is dense enough that all mobile objects are connected. Recall that when searching for the relevant geotasks, regional broadcast (e.g., broadcasting within a fixed region or a number of hops) is used. Many techniques can be used for such broadcast and they can result in very different communication costs. In fact, the cost itself can have different measures, e.g., the number of packets sent and/or received, or the size of total network traffic. To avoid assuming some particular technique and/or some specific metric, we simply assume the cost of 1-hop broadcast is c and the cost of broadcasting a fixed region is proportional to the area of the region. Thus, the cost of broadcasting within a range of i hops is $C_i = i^2 \cdot c$.

In geotask management, the communication cost mainly comes from two scenarios: 1) when a mobile object moves into a new grid, and 2) when a geotask is implanted or removed,

we analyze these two costs as follows.

3.4.1 Cost of Retrieving Relevant Geotasks

Recall that when a mobile object moves into a grid g, it needs to query for g's relevant geotasks. We call this mobile object a *requesting mobile object*. The search scope can be sequentially expanded, first within 1 hop, then 2 hops, ..., until some mobile object, called *supplying mobile object*, that caches the grid's relevant geotasks is located. Suppose the supplying mobile object is k-hop away from the requesting mobile object. Then it will take k rounds of search to locate this supplying mobile object. The total cost can be calculated according to this formula:

$$cost(k) = \sum_{i=1}^{k} i^2 \cdot c = \frac{k(k+1)(2k+1)}{6} \cdot c \tag{3.1}$$

Now the challenge becomes to identify where the supplying mobile object is actually located. Suppose there are N mobile objects and they are uniformly distributed in the network domain. Let p_0 be the position of the requesting mobile object, which must be on the boundary of grid g. If there is any mobile object in the grid, the one that is the nearest to the requesting mobile object will be the supplying mobile object. Otherwise, the grid's retaining mobile object is the supplying mobile object, which must be outside of grid g. Thus, the search cost can be analyzed based on where the supplying mobile object is found, within or outside of grid g.

Case I. The supplying mobile object is in grid g if there is at least one mobile object in the grid. Suppose that there are n mobile objects in the grid, where $n > 0$, and their positions are p_1, p_2, ..., and p_n, respectively. Then the number of hops from the requesting mobile object to the supplying mobile object is

$$H_n = \left\lceil \frac{\min(dist(p_0, p_1), \cdots, dist(p_0, p_n]))}{r} \right\rceil, \tag{3.2}$$

where $dist(x, y)$ denotes the Euclidean distance between two positions x and y. Hence, the average cost of locating the supplying mobile object can be calculated as

$$\overline{C_n} = \frac{1}{4d}(\frac{1}{d^2})^n \int_B \underbrace{\iint_A \cdots \iint_A}_{n} cost(H_n) \underbrace{dA \cdots dA}_{n} dB, \tag{3.3}$$

where B and A are grid g's boundary and area, respectively. Note that this cost, i.e., $\overline{C_n}$, can be computed using numerical approaches, say, *Monte Carlo* method [4].

Given a grid and a mobile object in the network domain, the probability that this mobile object is outside of this grid is $1 - \frac{1}{M}$. Given a total of N mobile objects, the probability of having n mobile objects (excluding the requesting mobile object) in grid g can be calculated as

$$P_n = \mathrm{C}_{N-1}^{n}(\frac{1}{M})^n(1 - \frac{1}{M})^{N-1-n} \tag{3.4}$$

Thus, in average, the cost of searching within grid g is

$$Cost_{inside} = \sum_{i=1}^{N-1} P_i \cdot \overline{C_i} \tag{3.5}$$

Case II. If the supplying mobile object is found outside grid g, there must be no other mobile object inside g. Based on Equation 3.4, the probability of having no other mobile object in grid g is $P_0 = (1 - \frac{1}{M})^{N-1}$. According to [69], the time duration that a randomly moving unit may stay in an area can be approximated as an exponential distribution and the mean staying time is

$$\bar{t} = \frac{\pi A}{E[v]L}, \tag{3.6}$$

where A is the area, L is the perimeter of the area, and $E[v]$ is the average speed of the mobile unit. Thus, the average time duration that a mobile object stays inside a grid is $\bar{t}_g = \frac{\pi d}{4E[v]}$. More generally, the average time duration that a mobile object stays inside a square of $q \times q$ is

$$\bar{t}_q = \frac{\pi q}{4E[v]}, \tag{3.7}$$

and the probability density function of the corresponding exponential distribution is

$$f_{t_s}(t, q) = \frac{1}{\bar{t}_q} e^{-\frac{t}{\bar{t}_q}} \tag{3.8}$$

Since the mobile objects are uniformly distributed and they move randomly, given a specific close region, the average frequency of the event that some mobile object moves into this region is equal to that of the event that some mobile object moves out of this region. Since the average number of mobile objects in a grid is $\frac{N}{M}$, the average time interval of two consecutive

move-in events on a specific grid is

$$\bar{t_e} = \frac{\pi d \cdot M}{4E[v]N}, \tag{3.9}$$

and the probability density function of the corresponding exponential distribution is

$$f_{t_e}(t_e) = \frac{1}{\bar{t_e}} e^{-\frac{t_e}{\bar{t_e}}} \tag{3.10}$$

Suppose the interval from the time when grid g's retaining mobile object moves out of g to the time when the requesting mobile object moves into g is t_u. Let p_r be the position of the retaining mobile object when the requesting mobile object queries for g's relevant geotasks. Then the number of hops from the requesting mobile object to the retaining mobile object is $H(p_r, p_0) = \lceil \frac{dist(p_0,p_r)}{r} \rceil$. Let S_i denote the square that centers on grid g and consists of $(2i-1) \times (2i-1)$ grids; and let D_i denote the geographic region that is inside of S_{i+1} but outside of S_i. Then the probability that p_r locates inside D_i is

$$\begin{aligned} p_i(t_u) &= \int_0^{t_u} f_{t_s}(t, (2i-1)d) - f_{t_s}(t, (2i+1)d)\, dt \\ &= e^{-\frac{t_u}{(2i+1)\bar{t_g}}} - e^{-\frac{t_u}{(2i-1)\bar{t_g}}} \end{aligned} \tag{3.11}$$

The average cost of locating the retaining mobile object in D_i is

$$\overline{C}_{D_i} = \frac{1}{4d} \frac{1}{8i^2d^2} \int_B \iint_{D_i} cost(H(p_r, p_0)) dB dD_i, \tag{3.12}$$

where B is the boundary of grid g as in Equation 3.3. Similar to $\overline{C}_n$, $Cost_{outside}$ can be computed numerically.

Note that the retaining mobile object can be anywhere outside of grid g, t_u can be from 0 to infinity, and the size of q can range from d to $\max(h, w)$. Let $Q = \lfloor \frac{\max(w,l)}{d} \rfloor$. Then the expected search cost incurred in case II is

$$\begin{aligned} Cost_{outside} &= P_0 \cdot \sum_{i=1}^{Q} \int_0^{\infty} f_{t_e}(t_u) \cdot \overline{C}_{D_i} \cdot p_i(t_u)\, dt_u \\ &= P_0 \cdot \sum_{i=1}^{Q} \left(\frac{(2i+1)\bar{t_g}}{\bar{t_e}+(2i+1)\bar{t_g}} - \frac{(2i-1)\bar{t_g}}{\bar{t_e}+(2i-1)\bar{t_g}} \right) \cdot \overline{C}_{D_i} \end{aligned} \tag{3.13}$$

Thus, the total cost of for a mobile object to retrieve the geotasks relevant to its new home grid is

$$Cost_{retrieval} = Cost_{inside} + Cost_{outside} \tag{3.14}$$

3.4.2 Cost of Implanting or Removing Geotasks

We now analyze the communication cost incurred in geotask updates, i.e., implanting or removing a geotask. Suppose the range of the geotask is a square of $l \times l$. The grid that contains the center position of this geotask can be seen as a concatenation of a number of rectangles. Let $a = \lceil \frac{l}{d} \rceil$ and $b = \lceil \frac{l}{d} \rceil - \frac{l}{d}$, where d is the grid size. Figure 3.3 shows these rectangles and their sizes. These rectangles can be classified into three categories: R_{1X} (including R_1 along), R_{2X} (including R_{21}, R_{22}, R_{23}, and R_{24}), and R_{3X} (including R_{31}, R_{32}, R_{33}, and R_{34}). The number of grids that overlaps with the range of the geotask depends on where the center position of the geotask is located:

- Case 1: If the position is inside R_1, the geotask overlaps with $(a+1)^2$ grids;
- Case 2: If the position is inside some R_{2X} rectangle, the geotask overlaps with a^2 grids;
- Case 3: If the position locates in some R_{3X} rectangle, the number of grids that overlap with the geotask is $a \cdot (a+1)$.

Suppose the geotasks are uniformly distributed in the network domain. Then the probability of each case is proportional to the size of its corresponding area. Specifically, the probability of case 1 is b^2; the probability of case 2 is $(1-b)^2$; and the probability of case 3 is $2b(1-b)$. Thus, the average number of grids that overlap with this geotask is

$$\begin{aligned} n_o &= a^2 \cdot b^2 + (a+1)^2 \cdot (1-b)^2 + a(a+1) \cdot 2b(1-b) \\ &= (a-b+1)^2 \\ &= (\tfrac{d+l}{d})^2 \end{aligned} \tag{3.15}$$

When a geotask is installed or removed, all mobile objects in the overlapping grid needs to be informed. Thus, the cost of installing or removing a geotask can be calculated as

$$Cost_{update} = n_o \cdot \frac{d^2}{\pi r^2} \cdot c \tag{3.16}$$

3.4.3 Overall Cost Per Time Unit

Equation 3.14 allows us to determine the cost of one time retrieval of relevant geotasks. We now analyze the frequency of such retrievals. According to equation 3.6, the average time

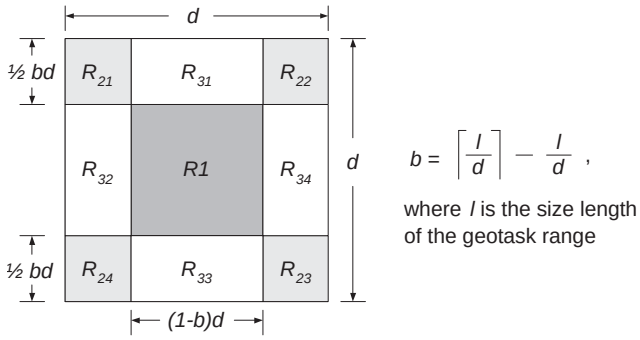


Figure 3.3 Possible Regions for a Geotask's Center Position

duration that a mobile object stays inside a grid is equal to $\frac{\pi d}{4E[v]}$. Thus, given that there are N mobile objects, the frequency of the events that some mobile object moves out its current grid is

$$Freq_{retrieval} = \frac{4E[v]N}{\pi d} \tag{3.17}$$

Each time a mobile object moves into a new grid, it needs to query for the grid's relevant geotasks. Thus, the average cost per time unit in geotask retrieval at the network wide can be calculated as

$$Cost_{retrieval\_ptu} = Cost_{retrieval} \times Freq_{retrieval} \tag{3.18}$$

On the other hand, if $Freq_{update}$ denotes the frequency of geotask updates, then the cost incurred for geotask update per time unit is

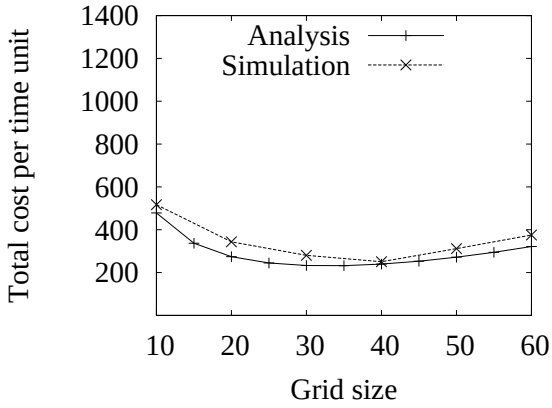
$$Cost_{update\_ptu} = Cost_{update} \times Freq_{update} \tag{3.19}$$

Thus, the overall cost per time unit occurred in the entire system can be calculated as

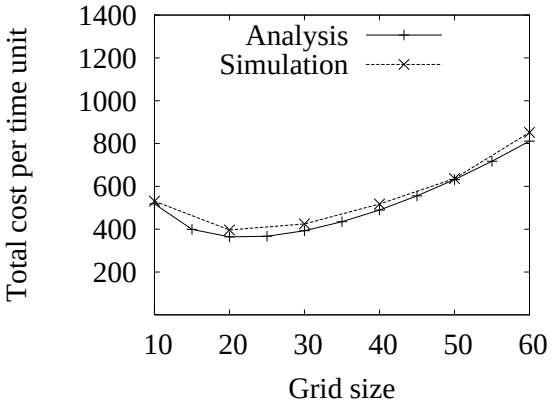
$$Cost_{ptu} = Cost_{retrieval\_ptu} + Cost_{update\_ptu} \tag{3.20}$$

3.4.4 Validation of the Analytical Model

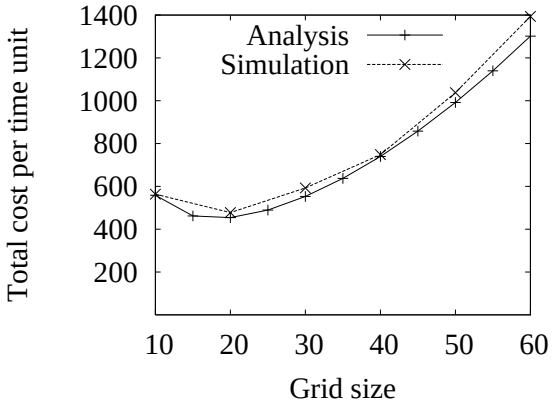
We validated Equation 3.20 using simulation. In this study, we fixed the network domain to be 120×120 $meter^2$ and placed on it a number of mobile objects. The mobile object density is set to be 0.05 $object/meter^2$. These mobile objects are uniformly distributed in the network domain and they move at random speeds, ranging from 0 to 10 $meter/second$



(a) Geotask update frequency = 10



(b) Geotask update frequency = 30



(c) Geotask update frequency = 50

Figure 3.4 Validation of the Analytical Model

and averaged at 5 *meter/second*. The mobile object transmission radius is fixed at $10\sqrt{2}$ *meter*. To broadcast a packet to all mobile objects in some geographic region, we used an existing technique called *Priority Forwarding* [29]. The performance of this technique has been shown to be little sensitive to the mobile object density. As indicated in [29], the number of mobile object participating in forwarding a broadcast message is close to be proportional to the intended broadcast area. In our study, we count the cost of broadcasting a message as the number of mobile objects that rebroadcast this message. Thus, the cost of 1-hop broadcast is $c = 1$.

The range of each geotask generated is a square and the size is uniformly distributed from 5 x 5 to 15 x 15. At the network wide, the frequency of geotask updates is fixed at 10, 30, and 50 geotasks/second. The grid size is varied from 10 to 60 meters. During each simulation run, we periodically compute the average communication cost incurred during a fixed time interval, and collect the performance data when the average cost per time unit becomes stabilized. Figure 3.4 shows both the simulation results and the results computed using Equation 3.20. We observe that the formula is quite accurate in predicting the average cost per time unit in each simulation setting. The slight error mainly comes from the assumption that the broadcast cost is proportional to the area of a broadcast region. In our analytical model, the distance between the requesting mobile object and the potential supplying mobile object is round to an integer. In particular, the cost of broadcasting a message in a rectangular region (e.g., within a grid or a combination of a number of grids that overlap with a geotask) is also approximated by its area. In simulation, however, the area covered by each broadcast retransmission is actually a circle.

3.5 Performance Study

In this section, we illustrate the cost incurred in geotask management using the analytical model presented in the previous section. Table 3.1 summarizes the parameters used in our study. Unless otherwise specified, the default values are used. We are mainly interested in how the average cost per time unit is affected by grid size, mobile object density, geotask update

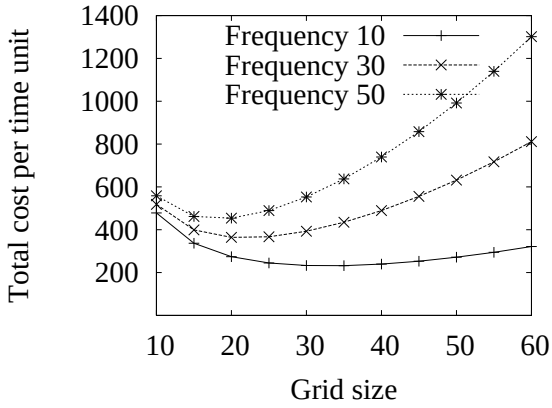
frequency, and the range of geotasks. In all studies, we present the total cost and its two separated costs, geotask retrievals and geotask updates, incurred per time unit.

Table 3.1 Simulation Parameters for Basic Geotasking

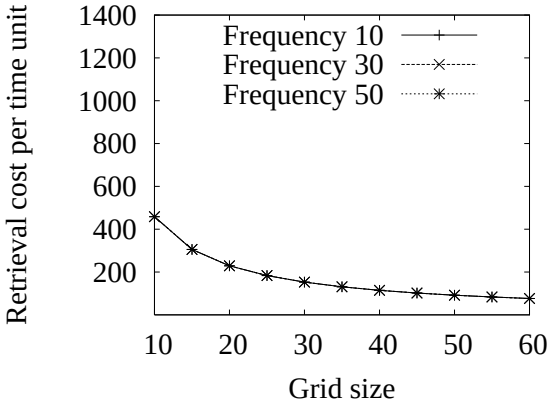
parameter	range	default	unit
network area	120x120	120x120	$meter^2$
grid size	10x10 - 60x60	30	$meter^2$
geotask range	2x2 - 20x20	N/A	$meter$
mobile object density	0.01 - 0.1	0.05	$object/meter^2$
mobile object speed	0 - 10	5	$meter/sec$
broadcast radius	$10\sqrt{2}$	$10\sqrt{2}$	$meter$
update interval	10 - 120	60	$second$
geotask size	5x5 - 15x15	10x10	$meter^2$

3.5.1 Effect of Grid Size

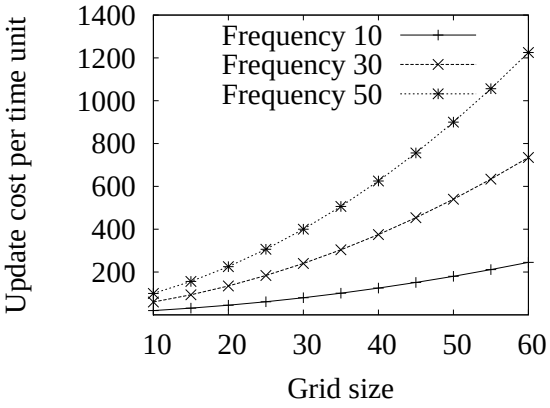
In this study, we varied the grid size from 10 to 60 meters. The performance results under three different geotask update frequencies (i.e., 10, 30, and 50 geotasks/second) are plotted in Figure 3.5. As the grid size increases, the total costs under all three settings initially reduce and then start to climb up. When a grid size results in the smallest per-time-unit total cost, we say the grid size is optimal. Figure 3.5 (a) shows that each setting has its own optimal grid size that minimizes the total costs and a lower update frequency has a larger optimal grid size. Specifically, the optimal grid sizes for the update frequencies of 10, 30, and 50 geotasks/second are 30, 20, and 15 meters, respectively. Note that the frequency of geotask update does not affect the cost of geotask retrievals. Thus, when other parameters are fixed, the cost incurred by geotask retrievals is fixed. Figure 3.5 (b) shows that the costs of geotask retrievals under three different update frequencies are the same. It also shows that the cost of geotask retrievals reduces as the grid increases. This is mainly due to the fact that the



(a) Total cost



(b) Geotask retrieving cost



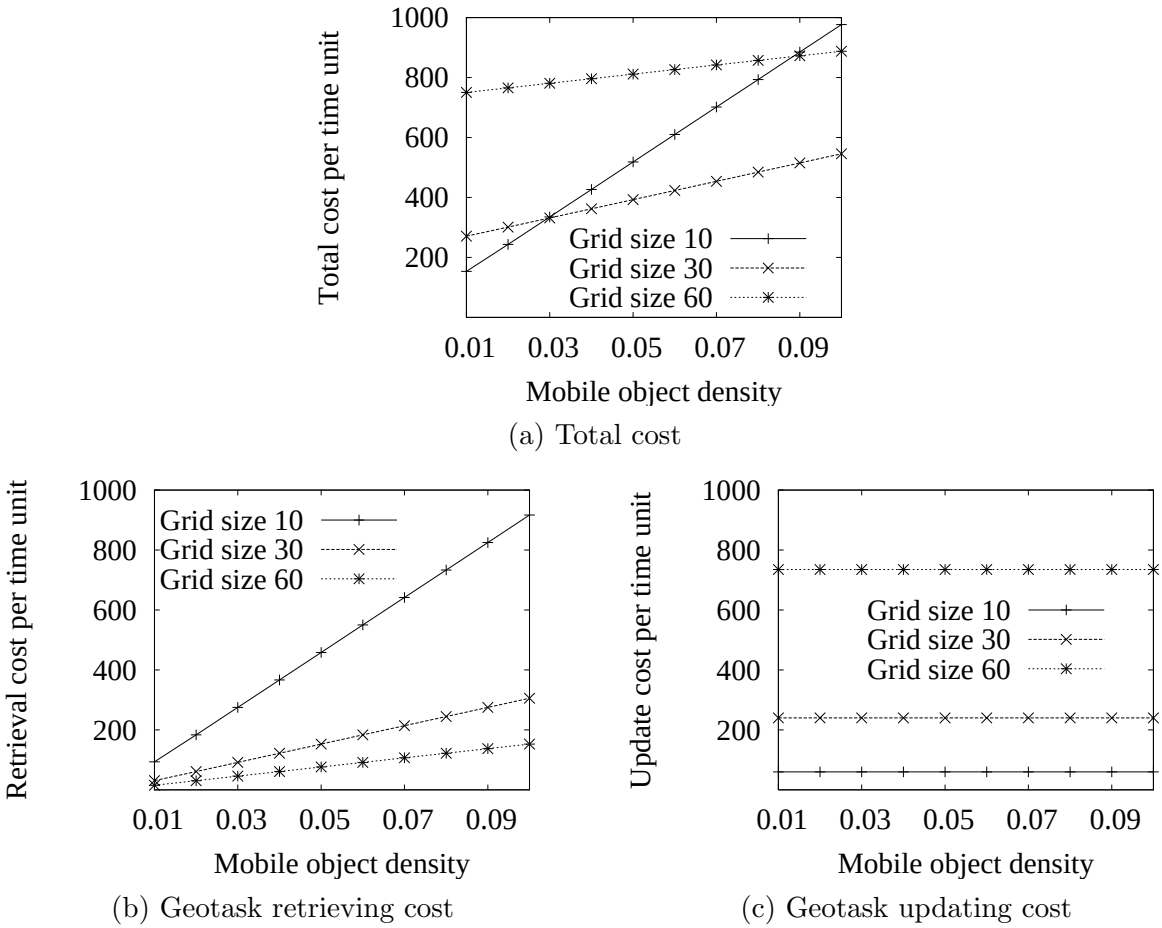
(c) Geotask updating cost

Figure 3.5 Effect of Grid Size

frequency of geotask retrievals is reduced. On the other hand, a larger grid size results in more costly geotask updates, as showed in Figure 3.5 (c). Obviously, this is because installing or removing a geotask needs to inform all mobile objects in the grids that overlap with the geotask. This study shows that the grid size has a profound impact on the overall system performance. In reality, the challenge of choosing a good grid size is to find the balance: A smaller grid size reduces the cost of geotask updates since it minimizes the unnecessary broadcast area; but on the other hand, it increases the frequency of geotask retrievals and thus the total cost incurred in geotask retrievals.

3.5.2 Effect of Mobile Object Density

This study investigated the impact of mobile object density on the system performance. We varied the mobile object density from 0.01 to 0.1 $object/meter^2$ and collected the performance results under three different settings of grid sizes: 10, 30, and 60 meters. The results are plotted in Figure 3.6. Given a fixed network domain, a higher mobile object density means a more number of mobile objects. When the grid size is fixed, this increases the frequency of geotask retrievals. Thus, the total cost of geotask retrievals increases, as confirmed by Figure 3.6 (b). This figure also shows that when the mobile object density increases, a smaller grid size causes a sharper increase of the per-time-unit cost of geotask retrievals. This is because for a same mobile object, its frequency of moving into a new grid increases linearly with respect to the grid size. Figure 3.6 (c) shows that the cost of geotask update is not affected by the mobile object density. This is simply because the cost of installing or removing a geotask is determined by the area of the geographic region that needs to be broadcast, instead of the number of mobile objects inside it. To reduce the cost of geotask updates, it is preferable to set a smaller grid size. However, a smaller grid size causes more frequent geotask retrievals. When the mobile object density increases to a certain level, the total cost incurred by geotask retrievals starts to become dominant and outweigh the savings from the reduction of the total cost incurred by geotask updates. As showed in Figure 3.6 (a), setting the grid size to be 10 meters initially results in the best performance; but as the mobile object density keeps



(a) Total cost

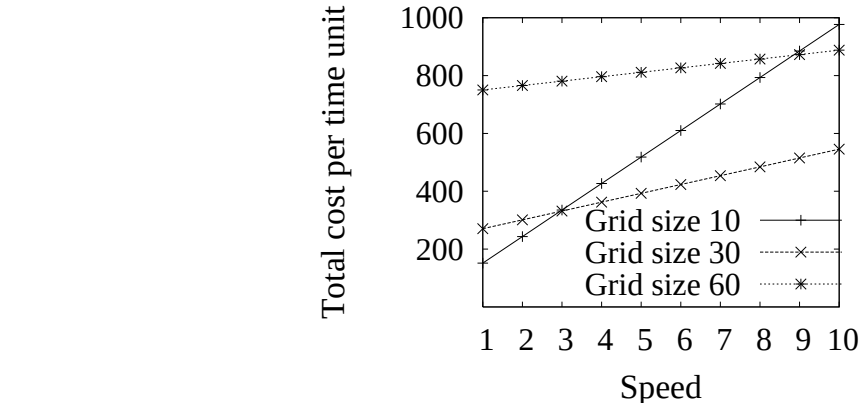
(b) Geotask retrieving cost

(c) Geotask updating cost

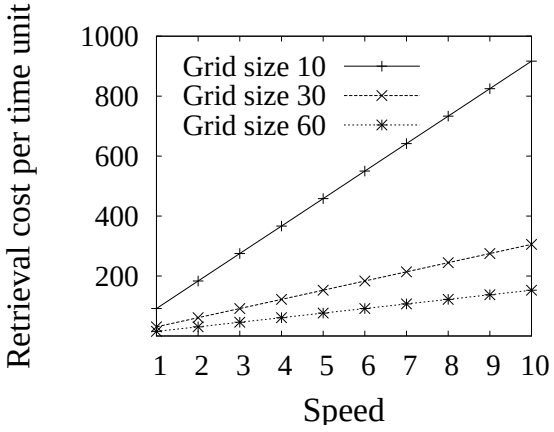
Figure 3.6 Effect of Mobile Object Density

increasing, its performance becomes deteriorated.

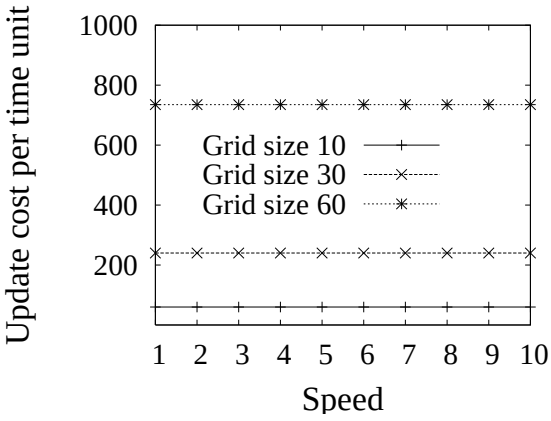
3.5.3 Effect of Mobile Object Movement



(a) Total cost



(b) Geotask retrieving cost



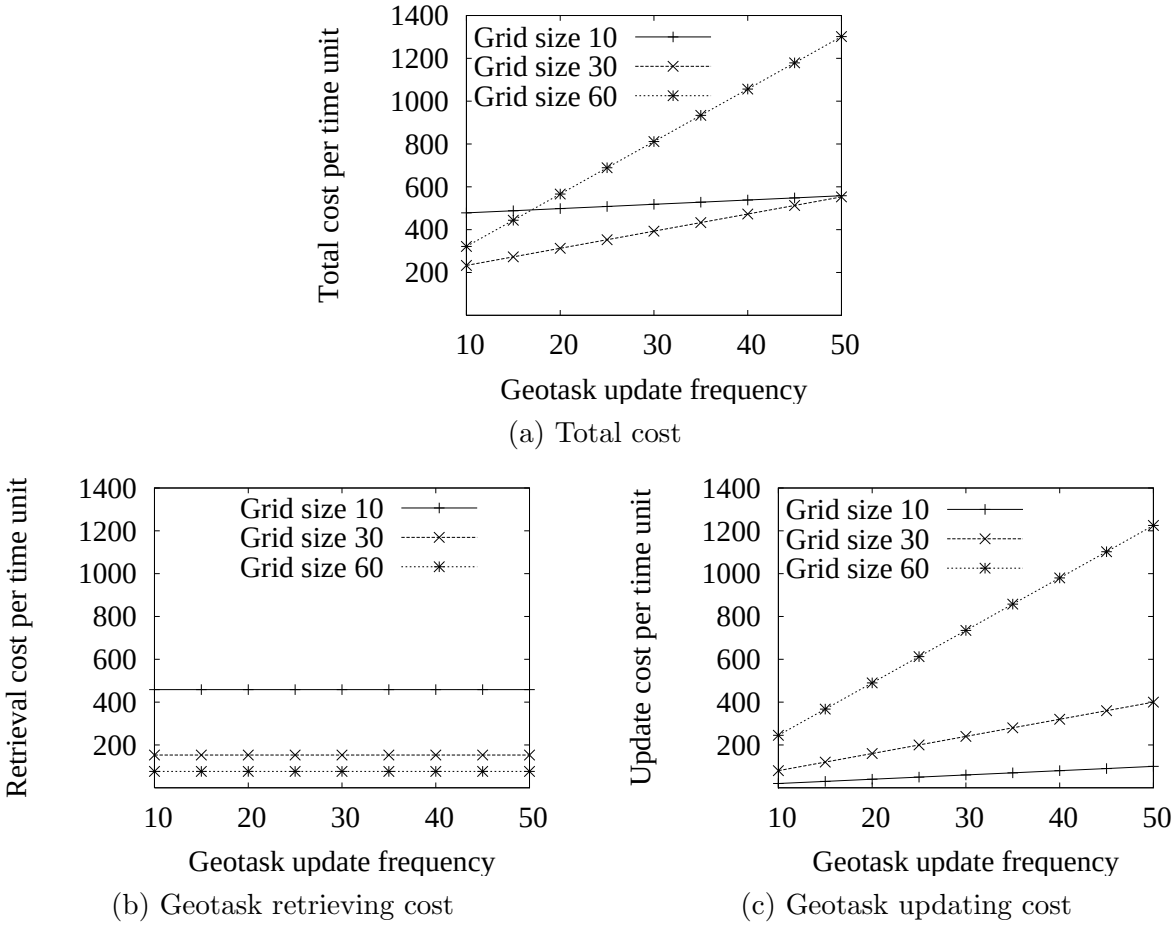
(c) Geotask updating cost

Figure 3.7 Effect of Mobile Object Movement

In this study, we varied the average moving speed of a mobile object from 1 to 10 m/sec. We collected data under three different settings of grid sizes: 10, 30, and 60 meters. Figure 3.7 shows the relationship between transmission cost and the average speed of mobile object. The cost is increased as the movement becomes faster since the number of request summary message is increased. The he smaller the grid size increases cost faster than other grid size with the speed variance. From equation (3.7) we can see that the frequency of a mobile object

leaving its current grid is proportional to the average speed, and reversely proportional to the grid size.

3.5.4 Effect of Geotask Update Frequency



(a) Total cost

(b) Geotask retrieving cost

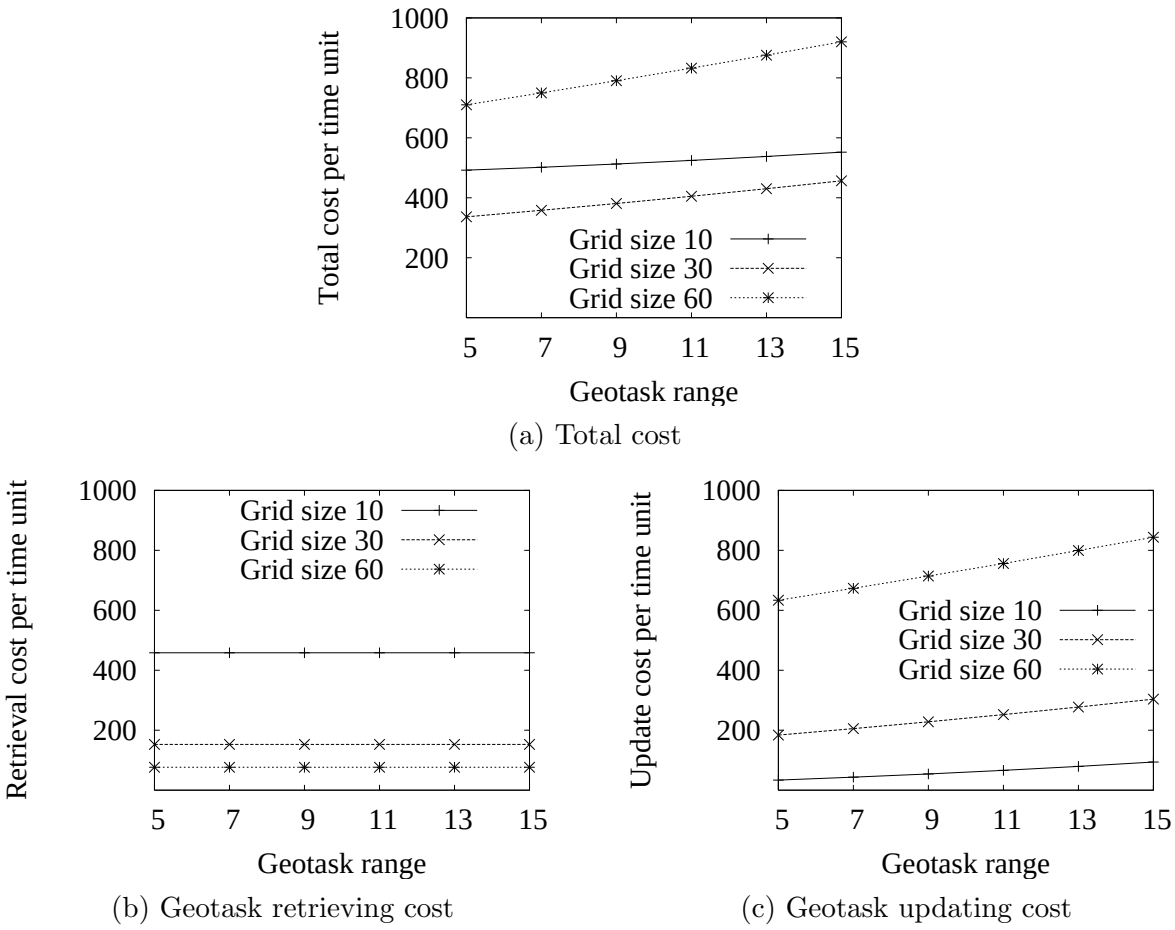
(c) Geotask updating cost

Figure 3.8 Effect of Update Frequency

In this study, we varied the geotask update frequency from 10 to 50 geotasks/second. We collected the performance data under three different grid sizes and plotted them in Figure 3.8. Since the frequency and the cost of geotask retrievals are not affected by the geotask updates, the curves for the total cost of geotask retrievals are flat, as showed in Figure 3.8 (b), although different grid sizes results in different total costs. A higher update frequency causes a high

total cost of geotask updates. This expected increases are confirmed in Figure 3.8 (c). Again, a smaller grid size avoids more unnecessary area that needs to be broadcast when installing or removing a geotask. Therefore, the total cost incurred by the geotask updates is minimized when the grid size is set to be 10 meters, which is the minimal setting since the mobile object transmission radius is $10\sqrt{2}$ meters.

3.5.5 Effect of Geotask Ranges



(a) Total cost

(b) Geotask retrieving cost

(c) Geotask updating cost

Figure 3.9 Effect of Geotask Area Size

In this study, we increased the average ranges of geotasks from 5x5 to 50x50 $meter^2$. The performance data under three different grid sizes are plotted in Figure 3.9. When a geotask

is installed or removed, all grids that overlaps with its range needs to be broadcast. Thus, installing or removing a geotask with a larger range costs more in general. This is showed in Figure 3.9 (c). Since the frequency of geotask retrievals and the cost of per retrieval are not affected by the ranges of geotasks, the three curves are flat in Figure 3.9 (b). Figure 3.9 (a) shows that the performance under the grid size of 30 meters is the best. Again, this is a result of the balance of geotask updates and geotask retrievals, one preferring a larger grid size while the other a smaller one.

3.6 Conclusion

Our geographic environment is a natural storage where we humans store and share data; and wherever we go, we load and process, either consciously or unconsciously, the data implanted around us. Abstracting from this natural computing, we define two fundamental features for geotasking: 1) programs are treated like data and are associated with geographic regions and 2) the execution of these programs are triggered by the movements of their nearby mobile objects. These features enable geotasking to support *a range of new distributed applications under one platform*. We have presented a generic framework for cost-effective management of various types of geotasks. The proposed approach supports persistent storage and timely execution of geotasks in a fully distributed environment without relying on any stationary server. For performance evaluation, we have developed a mathematical model and a detailed simulator.

CHAPTER 4. TYPES OF GEOTASK

Recall that we classified geotasks along three dimensions: mobile object correlation, geotask mobility, and geotask dependency. Table 4.1shows types of geotasks. In the previous chapter, we have considered only the geotasks with simple step-in triggers; they are bounded to fixed geographic regions and also assumed to be independent of each other. We explore other types of geotasks and propose techniques converting other types of geotask to basic geotask (i.e., s-class stationary geotask).

Table 4.1 Types of Geotask

Object correlation	Geotask mobility	Geotask dependency
s-class	stationary	independent
m-class	mobile	interrelated

4.1 Mobile Object Correlation

To fulfill various application needs, different execution triggers need to be defined. For example, we may want a geotask to be executed when a mobile object moves out of its binding region, or moves into the region from a certain direction at some minimum speed. In reality, a stop sign is applicable only to the vehicles approaching from the direction it faces. As another example, a user may want to associate a geotask with a nearest neighbor (NN) trigger – a mobile object needs to execute the geotask whenever it becomes the one that is nearest to the geotask. According to whether or not the movement of mobile objects is correlated in triggering geotask execution, we classify geotasks into two categories.

- *s-class:* A geotask is in this category if its execution is triggered by the movement of each mobile object independently. A geotask with the step-in-trigger is in this category. By monitoring its own movement against the geotask, a mobile object can determine whether or not it has moved into the binding region (and thus needs to execute the geotask).

- *m-class:* a geotask is in m-class if the movement of one mobile object may cause another mobile object to execute the geotask. A geotask with the aforementioned NN trigger is an m-class geotask. In this case, a mobile object may become the nearest neighbor of the geotask because of the movement of other mobile objects.

One trigger in s-class geotask of a particular interest is *overlapping*. A geotask with such a trigger is executed by a mobile object whenever the mobile object's *capable range* overlaps with the geotask's binding region. Here the concept of capable range is broadly defined. A mobile object with a camera has a "vision" and the capable range is the area that it can picture; similarly, a mobile object with a microphone has a capable range in which its sound can be heard. Moreover, a mobile object with multiple sensors can have multiple capable ranges, each on one sensing apparatus. Overlapping triggers can be useful in many applications. For instance, a region can be implanted with a geotask to "monitor enemy tanks and report when detecting one". A mobile object with such capability can perform this task as long as it is capable of monitoring any part of the region, even though it may be outside of region. As another example, a senior person may want to be aware of a stop sign before moving close to it. In this case, a mobile object can be associated with a user-defined capable range so that the user can be alerted from a desired distance. S-class geotasks can be supported by extending our basic geotask management platform.

Compared to s-class geotasks, m-class geotasks are significantly more difficult to support, because the firing of their execution is determined by the movement of multiple mobile objects. Consider a geotask bound to a position P with an NN trigger. To ensure the geotask is executed as soon as its trigger is fired, one solution is to let each mobile object broadcast its every movement to the entire network; whenever a mobile object receives a location update,

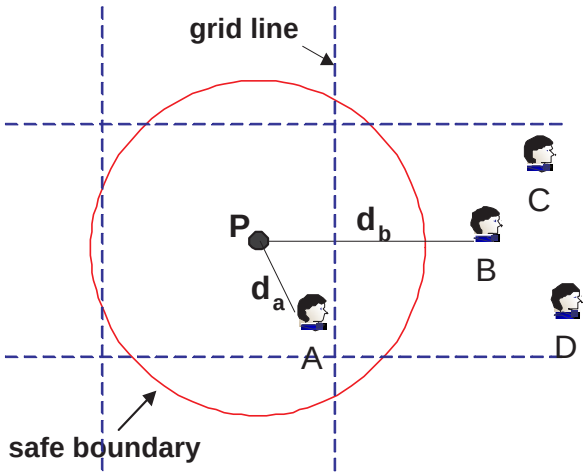


Figure 4.1 Safe Boundary

it checks if it now becomes the one that is nearest to P. This approach will incur excessive network overhead. On the other hand, we investigate the following idea. Figure 4.1 shows two nearest neighbors (A and B) of a query point P. Let d_a and d_b be the distance between A and P, and between B and P, respectively. Our key observation is that A will always be the one that is the nearest to P as long as no mobile objects cross over the *safe boundary*, the circle that centers on P with a radius between d_a and d_b. If a radius of a boundary is set as close to either d_a or d_b, A or B will crosses this boundary with high probability, respectively. The more crossing the boundary incurs the more frequent position update. We set a radius of a safe boundary as $\frac{d_a+d_b}{2}$ which gives equal probability of crossing a boundary for A and B. We just need to check if the NN trigger is fired only when a mobile object crosses this boundary after setting a boundary. By making a mobile object perform such a check whenever it crosses over the boundary, we can support a geotask G with an NN trigger by binding an *auxiliary* geotask G' to the safe boundary with the following instructions for the crossing mobile object to execute: 1) Find the two mobile objects that are currently nearest to P, and compute the new safe boundary; 2) Replace the binding region of G' with the new boundary; and 3) If the nearest mobile objects to G has changed, execute G (i.e., send the NN result to a query requestor). Figure 4.2 shows converting m-class, stationary geotask to s-class stationary geotask with safe boundary concept. Note that, R is a geotask bound to a position P with an

NN trigger and R' is a safe boundary.

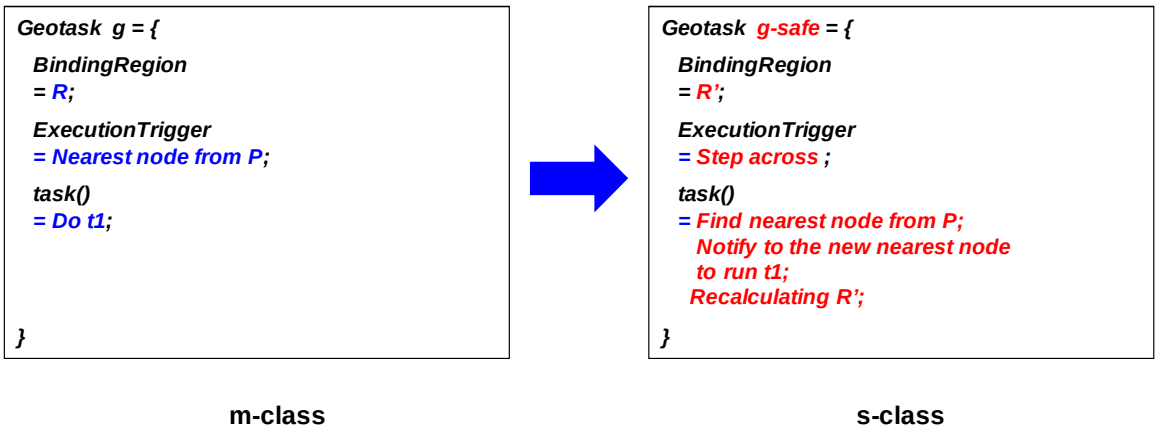


Figure 4.2 Converting m-class to s-class

When a user asks to create a geotask with an NN trigger, the system can compute its initial safe boundary and then install a corresponding auxiliary geotask with an executable that includes the above instructions. The auxiliary geotask is executed whenever a mobile object crosses over its binding region and therefore it is an s-class geotask. This indicates that the geotask with an NN-trigger, which is an m-class geotask, can be converted into an s-class geotask, which can then be managed using the basic geotasking management. In general, not every movement of mobile objects will trigger the execution of a geotask. Given an m-class geotask, we can identify all safe boundaries where a crossing may cause it to be executed; and for each of these boundaries, we can implant an auxiliary geotask with the instructions that we want a crossing mobile object to perform. *This strategy allows an m-class geotask to be converted into one or more s-class geotasks.* Some specific types of triggers to be considered include KNN trigger (a mobile object needs to execute a geotask if it is one of the K mobile objects that are nearest to the geotask) and density-monitoring trigger (execute a geotask as soon as the number of mobile objects inside its binding region exceeds some threshold). For KNN triggers, we investigate what safe boundaries need to be defined and what actions need to be performed when a crossing event occurs in the next chapter.

4.2 Geotask Mobility

Recall that a stop sign is a geotask bound to a geographic region. While the binding region for such a geotask is fixed during its entire life time, many geotasks in our daily life may be bound dynamically. For instance, an advertising message displayed on a vehicle is a geotask that is associated with a moving point; as the point moves, the corresponding binding region changes. A police car moving with its siren on is another example of mobile geotasks that demands attention from its nearby people wherever it goes. Dynamic binding adds a new dimension of challenges to geotask management. When a mobile, referred as a focal mobile object, is bound with a geotask, it would have to advertise its every movement in order for other mobile objects to know the current binding region. One way to reduce the communications cost is to let the focal mobile object broadcast its velocity information to all mobile objects inside the cells where this geotask might need to be executed. This approach, however, is effectively only when the velocity does not change frequently.

Our idea is to convert a mobile geotask into one or more stationary geotask, which can then be supported with the basic geotasking management platform. Consider a mobile geotask with a step-in trigger. The binding region is a circle with a radius of d_f and is centered at the focal mobile object's position P. Let A be the nearest mobile object to P that is outside of the binding region, and let d_a be its distance to P. We define two circular safe boundaries, B_i and B_0, which are illustrated in Figure 4.3. Both circles center on P, but B_i has a radius of $\frac{d_a - d_f}{2}$ and B_0 has a radius of $\frac{d_a + d_f}{2}$. We have the following observation: As long as the movement of the focal mobile object is limited inside B_i, the binding region will always be contained by B_0. In other words, unless the focal mobile object moves out of B_i, or there is a mobile object crossing over B_0, it is impossible that some mobile object steps into the binding region and therefore has to execute the geotask. As a result, we can support the mobile geotask as follows. Whenever the focal mobile object moves out of B_i, we let it broadcast its current position to all mobile objects inside the cells relevant to the geotask (allowing these mobile objects to check if they are now in the binding region); and whenever a mobile object moves into B_0, we let it check with the focal mobile object and execute the geotask if and whenever a mobile object

moves into B_0, In either case, a mobile object which crosses either B_i or B_0 recomputes both boundaries.

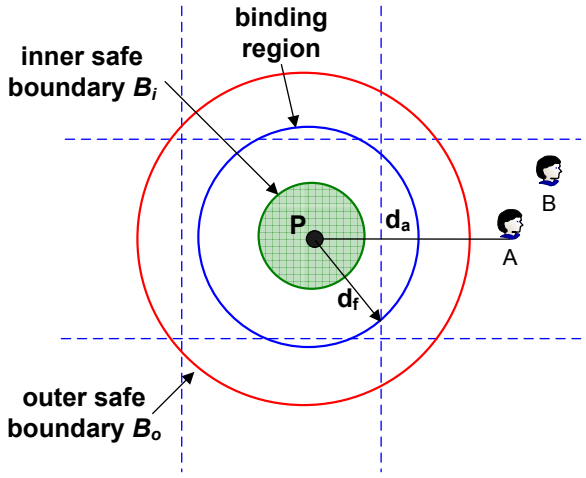


Figure 4.3 Inner and Outer Safe Boundaries

Clearly, the above mechanisms can be implemented by binding two auxiliary geotasks on B_i and B_0, respectively, and associating them with the corresponding actions for a crossing mobile object to perform. Both geotasks are stationary – their binding regions are fixed until a crossing event occurs. In general, to support a mobile geotask, we can find an inner safe boundary for the focal mobile object and all outer safe boundaries where a crossing event may trigger the execution of the geotask; and for each of these boundaries, we bind an auxiliary geotask with some necessary instructions for a crossing mobile object to perform. *This strategy allows us to convert a mobile geotask into a number of stationary geotasks.* Figure 4.4 shows converting mobile geotask to 2 stationary s-class geotasks.

4.3 Geotask Dependency

The geotasks we consider so far are independent in the sense that the execution of a geotask is determined by the movement of mobile object(s) with respect to the geotask itself. We say a geotask G is *dependent* on another geotask G' if the execution of G is determined by the movement of mobile object(s) with respect to both G and G'. A geotask may be dependent on multiple geotasks. As an example, consider a group of geotasks inside a region R and a mobile

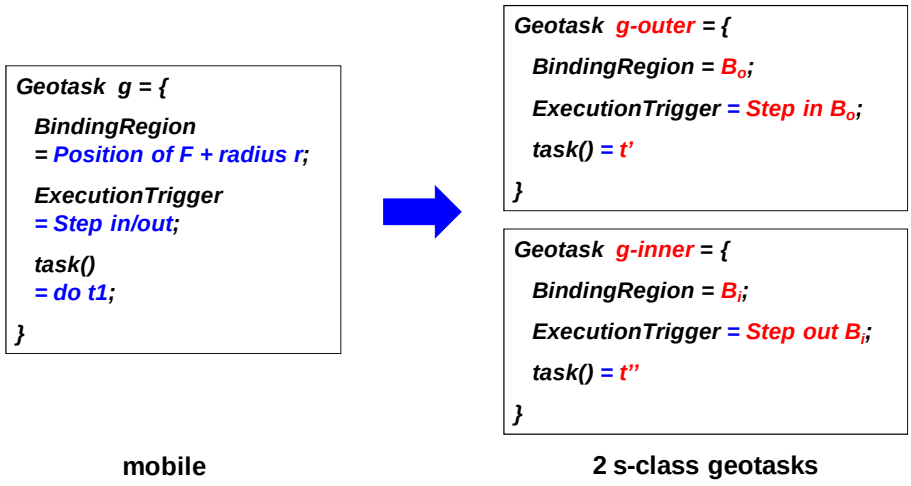


Figure 4.4 Converting Mobile Geotask to s-class Geotasks

node in r. We want to monitor the nearest geotask from the mobile object, and execute it when other geotasks' statuses are changed. An application use of such interrelated geotasks is to remind soldiers in a battlefield of their nearest dangerous spots (each can be implemented as a geotask displaying a warning message).

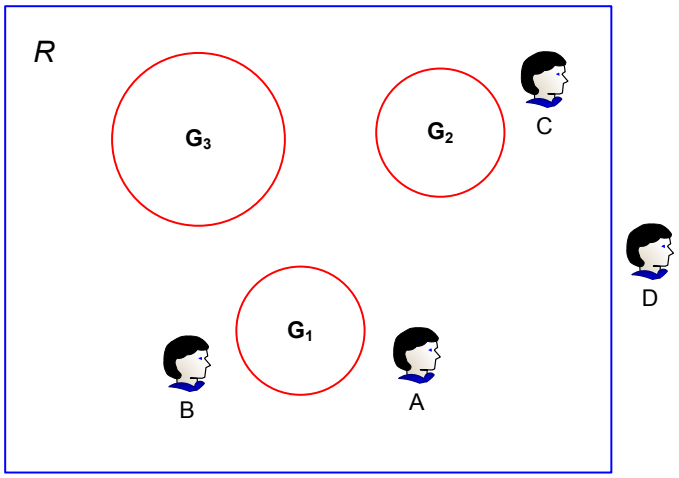


Figure 4.5 Interrelated Geotasks

The above example is a group of s-class interrelated geotasks - a mobile object can determine which geotask to execute by monitoring its own movement against the geotasks in the group. There are also m-class interrelated geotasks. For example, when a mobile object moves inside

R, we may want it to be aware of the geotask that has the least number of mobile objects that take it as the nearest geotask. Figure 4.5 shows three geotasks and three mobile objects in region R. G_1 has two mobile objects (i.e., A and B) that consider it as the nearest geotask. Similarly, G_2 has mobile object C taking it as the nearest geotask and G_3 has no such mobile object. Thus, if mobile object D is moving into R at this time, then it would need to execute G_3. Such m-class interrelated geotasks can be used for load balancing, for instance, each geotask is a supplying station in a battlefield and we want mobile objects to know the station that is least loaded.

Also like independent geotasks, interrelated geotasks can be mobile. While an individual geotask can move, the group binding region may also change as a result of the movement of geotasks.

For s-class stationary interrelated geotasks, we support them by extending our basic geotask management platform. Given a group of such geotasks inside R, we can create a macro geotask that takes R as its binding region. The macro also includes the location information of these geotasks. When a mobile object moves into R, it can then monitor its movement against each individual geotask and executes one when necessary. As such, the group of interrelated geotasks can then be implemented as an independent geotask. For m-class stationary interrelated geotasks, it is significantly more challenging to support because a mobile object needs to track the movement of other mobile objects in order to determine which geotask to execute. Given a group of such geotasks, if its binding region R is small (say, within 1-2 hops wireless coverage), one can let mobile objects inside R broadcast their every movement so that each mobile object knows when and which a geotask needs to be executed. However, if the binding region spans over a large terrain or involves a large number of mobile objects, this approach would incur excessive communication cost.

CHAPTER 5. ADVANCED GEOTASK MANAGEMENT

5.1 Introduction

We introduce various types of geotasks in the previous chapter. We propose a platform for these advanced geotasks in this chapter. Our platform supports s-class/m-class geotask, stationary/mobile geotask, and independent geotask. We introduce a *spatial monitoring query* (SMQ) for management of advanced geotasks since SMQ contains all characteristics of above advanced geotasks.

Given a set of mobile objects, a SMQ retrieves the set of mobile objects that provides real-time updates on the query results whenever the status of mobile objects are satisfied with predefined conditions. SMQ allows users to monitor mobile objects that satisfy some spatial constraints. The four most important types of SMQs are:

- *Stationary Range Monitoring Query* (S-RMQ): An S-RMQ retrieves the set of mobile objects that are inside a user-defined geographic region, and provides real-time updates whenever a mobile object crosses over the border of the region. As mobile objects move, the query results may keep changing.

- *Mobile Range Monitoring Query* (M-RMQ): An M-RMQ also allows one to monitor the mobile population inside a user-defined region, but the query region is associated with a mobile object, referred to as a *focal mobile object*. As the mobile object moves, the query region changes.

- *Stationary KNN Monitoring Query* (S-KNNMQ): An S-KNNMQ retrieves the set of K mobile objects that are nearest to a user-defined query point, and provides real-time updates whenever this set of mobile objects changes.

- *Mobile KNN Monitoring Query* (M-KNNMQ): An M-KNNMQ is the same as an S-KNNMQ, except that the query point is associated with some mobile object. As the focal mobile object moves, the query point changes.

Unlike regular spatial queries, an SMQ is a continuous query and may stay active for a certain time period. As basic operators in moving objects database management systems, efficient processing of the above four types of SMQs has been investigated intensively, and separately, in the context of a centralized environment, wherein mobile objects are assumed to be able to communicate with some central server directly. Our research considers the problems of providing SMQs in a fully distributed mobile networking environment. Specifically, given a set of mobile objects that forms a M-P2P, we want to perform SMQs over these mobile objects. Our work is motivated by the potential uses of SMQs in the application scenarios (e.g., battlefields, emergency management) where fixed communication infrastructures may not exist. In the absence of any central/stationary server, efficient query processing is challenging. A simple solution is to let mobile objects broadcast their every movement to the entire network. Whenever a query creator receives a location update, it computes the query result by itself. This strategy allows one to provide real-time and accurate query results, but is not scalable. Excessive location updates will quickly exhaust mobile objects' battery, and may suspend the whole network.

In this dissertation, we present a novel technique, which we refer to as *Peer-to-Peer Monitoring Query Management* (P2P-MQM), for real-time and cost-effective processing of SMQs. Our technique supports S-RMQs and the three types of SMQs by converting them into S-RMQs. Thus, all queries can be managed with one common platform. To minimize communication costs incurred in query management, our approach stores queries among mobile objects and does so dynamically to make each mobile object aware of its nearby queries. We extend the basic geotask management platform to support SMQs.

5.2 Handling Stationary Range Monitoring Queries

The basic geotask management platform in Chapter 3 can be used for supporting S-RMQs. The query is bound to the area to be continuously monitored. The network is partitioned into disjoint grid cells and each mobile object caches the queries that are relevant to its home grid cell. The mobile object sends its id to a query requestor when it crosses query boundary. In Chapter 3, a mobile object only checks step in trigger condition. However, to support S-RMQ, a mobile object executes a geotask when it steps in and steps out of the binding region.

5.2.1 Detailed Design

We now give a more formal description of our technique. Our design of a mobile object mainly consists of two components: *MessageListener* and *RegionMonitor*. To facilitate our discussion, we define the following notations for the data structures maintained by a mobile object:

- *myID*: The mobile object's unique identifier
- *myPos*: The mobile object's current position
- *myCell*: The mobile object's current home cell
- *myQueries*: The list of queries that are relevant to *myCell*
- *myRetains*: The list of cells to which the mobile object is currently a retaining mobile object and their relevant queries;

MessageListener: The mobile object listens to these messages and processes them as follows:

- *SearchMobileObject(cell)*: If *myPos* is inside *cell*, or *cell* is listed in *myRetains*, reply with a *candidate* message including $myID$ and $myPos$.
- *RetrieveQueries(cell)*:

  - Send all queries relevant to *cell* to the requester;
  - If *cell* is listed in *myRetains*, remove the cell and its relevant queries from *myRetains*.

- $SetRetainingMobleObject(cell, queries)$:
  - Add *cell* and *queries* to *myRetains*.

- *InstallQuery(Q)*:
  - If query Q overlaps with *myCell*, add Q to *myQueries*;
  - For each cell g listed in *myRetains*, if g overlaps with Q, add Q to the list of g's relevant queries.

- *RemoveQuery(Q)*:
  - If query Q overlaps with *myCell*, remove Q from *myQueries*;
  - For each cell g listed in *myRetains*, remove Q from the list of g's relevant queries.

RegionMonitor: When the mobile object moves, it monitors its movement and does the followings:

1. If it crosses over the boundary of any query listed in *myQueries*, notify the query's creator;
2. If it moves into a new cell, say *newCell*, do the followings:
   - Set $CandidateList = NULL$;
   - Broadcast message $SearchMobileObject(myCell)$ within *myCell*;
   - Collect all *candidate* messages and add them to $CandidateList$;
   - If $CandidateList = NULL$, add *myCell* and *myQueries* to *myRetains* (becoming the cell's retaining mobile object);
   - Otherwise, do the following:
     - For each cell c (if any) in *myRetains*, check its distance to *myCell* and to *newCell*;

  - If c is closer to $myCell$ than to $newCell$, then find the mobile object in the $CandidateList$ that is nearest to c, and send a message $SetRetainingMobileObject(g, queries)$ to the mobile object, where $queries$ is the list of queries relevant to c.

- Set $myCell = newCell$, $CandidateList = NULL$, and $TTL = 1$;
- While $CandidateList$ is $NULL$, do the followings:
  - Broadcast message $SearchMobileObject(myCell)$ within a scope of TTL hops;
  - Collect all *candidate* messages and add them to $CandidateList$;
  - Increase TTL by 1.
- Among all mobile objects in $CandidateList$, find the one which is the closest to $myPos$, and send it a message $RetrieveQueries(myCell)$ to retrieve $myCell$'s relevant queries.

5.3 Handling Other Types of SMQs

We now consider how to support other types of queries, including M-RMQ, S-KNNMQ, and M-KNNMQ. We show that given such a query, we can find a set of *safe boundaries*; unless there is a crossing event, the query result does not change. Thus, the query needs to be re-evaluated only when some mobile object crosses over a boundary. Since each boundary can be considered as a stationary range monitoring query, the technique presented in the previous section can then be used to manage all these types of queries.

5.3.1 Mobile Range Monitoring Query

An M-RMQ is associated with a mobile object, referred to as a focal mobile object. As the mobile object moves, the query region changes accordingly. An example use of such queries is for a commander to monitor the soldiers that are within 1 mile of his current position.

Figure 5.1 shows a focal mobile object N at position (x_n, y_n) and its current query region $R[(x_q, y_q), (x'_q, y'_q)]$. Given a rectangle R, we will denote it as $R[(x, y), (x', y')]$, where

points (x, y) and (x', y') are its lower-left and upper-right coordinates, respectively. Let $R_i[(x_i, y_i), (x'_i, y'_i)]$ be the minimum bounding rectangle that contains all mobile objects that are currently inside the query region, and let $R_o[(x_o, y_o), (x'_o, y'_o)]$ be the maximum bounding rectangle that contains the query region and does not have any mobile objects that are currently outside the query region. Then we call the following three rectangular boundaries, B_i $[(\frac{x_q+x_i}{2}, \frac{y_q+y_i}{2}), (\frac{x'_q+x'_i}{2}, \frac{y'_q+y'_i}{2})]$, B_o $[(\frac{x_q+x_o}{2}, \frac{y_q+y_o}{2}), (\frac{x'_q+x'_o}{2}, \frac{y'_q+y'_o}{2})]$, and B_f $[(x_n - \min(\frac{x_q-x_o}{2}, \frac{x'_i-x'_q}{2}), y_n - \min(\frac{y_q-y_o}{2}, \frac{y'_q-y'_i}{2})), (x_n + \min(\frac{x'_o-x'_q}{2}, \frac{x_i-x_q}{2}),\ y_n + \min(\frac{y'_o-y'_q}{2}, \frac{y_i-y_q}{2}))]$, as the M-RMQ's *outer safe boundary*, *inner safe boundary*, and *focal safe boundary*, respectively. Our key observation is that, the query result will remain the same unless the focal mobile object moves out of B_f, some mobile object currently inside R crosses over B_i, or some mobile object currently outside of R crosses over B_o. Thus, the query can be processed by making the focal mobile object monitor its movement against B_f, and other mobile objects B_i and B_o. If anyone detects that it crosses over a boundary, it notifies the focal mobile object to re-evaluate the query and provides the latest query result. Note that the above discussion assumes that each query region is a rectangular area. In the case it is a circular area, we can compute the circular safe boundaries in a similar way.

To create a new M-RMQ, a mobile object notifies the selected focal mobile object, which then executes the following M-RMQ(N_p, R_q) procedure, where N_p is the current position of the focal mobile object and R_q is the current query region.

M-RMQ(N_p, R_q)

1. Find all cells that overlap with R_q;
2. Retrieve the position of all mobile objects inside these overlapping cells;
3. Provide the initial query result (i.e., find all mobile objects inside R_q);
4. Compute R_o and R_i;
5. Compute B_o, B_i, and B_f, according to N_p, R_o, R_i, and the position information of the mobile objects inside the overlapping cells, and do the followings:

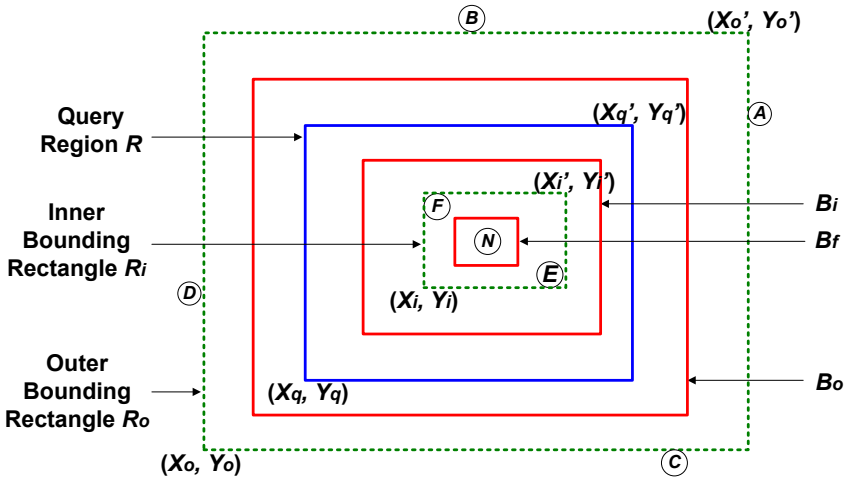


Figure 5.1 Safe Boundaries for an M-RMQ

- Submit B_o and B_i as two S-RMQs with the additional information that tells a mobile object to perform the followings whenever crossing over B_o or B_i,
  - Remove the two S-RMQs created with B_o and B_i;
  - Notify the focal mobile object to re-evaluate the query by calling **M-RMQ**(N_p, R_q).
- Monitor movement against B_f and if moving out of it, re-evaluate the query by calling **M-RMQ**(N_p, R_q);

5.3.2 Stationary KNN Monitoring Query

When a user issues an S-KNNMQ on point p, the system needs to return the k mobile objects that are nearest to the query point p, and provide continuous updates whenever the query result changes. Let N_k be the k^{th} nearest mobile object to p with a distance of d_k, and N_{k+1} be the $(k+1)^{th}$ nearest mobile object to p with a distance of d_{k+1}. Let B be the circular boundary that centers on p with a radius of $\frac{d_k+d_{k+1}}{2}$. These notations are illustrated in Figure 5.2. The set of k mobile objects nearest to p does not change as long as no mobile objects move across B. Thus, B can serve as the query's safe boundary. When a mobile object crosses over the boundary, it computes a new safe boundary by identifying the $k+1$ mobile objects nearest to p and informs the query creator of the new set of k nearest mobile objects.

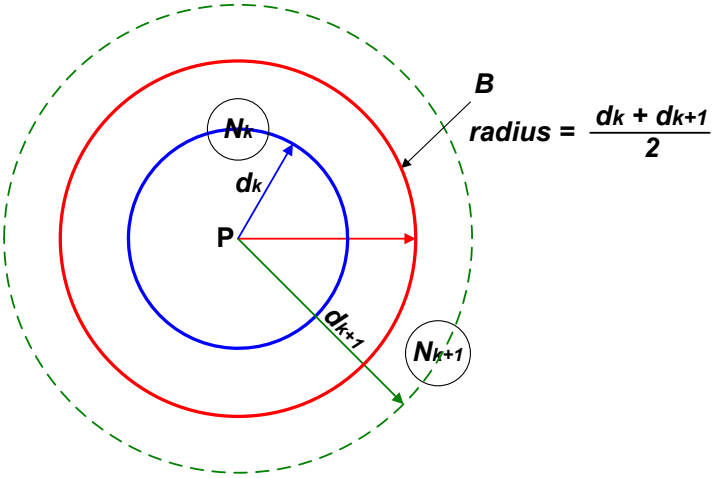


Figure 5.2 Safe Boundary for an S-KNNMQ

The above approach converts an S-KNNMQ into a circular S-RMQ. A more formal description for this algorithm is as follows, where p is the query point and k is the number of nearest neighbors to monitor.

S-KNNMQ(p, k)

1. Find the $k+1$ mobile objects that are nearest to p by searching the cell that contains p (and expanding the search scope if necessary);
2. Notify the query creator the k nearest mobile objects;
3. Compute the safe boundary B according to p, the positions of the k and $k+1^{th}$ nearest mobile objects;
4. Submit B as an S-RMQ with the additional information that tells a mobile object, when crossing over B, to re-evaluate the query by calling **S-KNNMQ**.

5.3.3 Mobile KNN Monitoring Query

An M-KNNMQ, like an M-RMQ, is associated with a focal mobile object; as the mobile object moves, the query point p changes accordingly. Figure 5.3 shows an M-KNNMQ having its current query point at p. Again, let N_k be the kth nearest mobile object to p with a

distance of d_k, and N_{k+1} be the $(k+1)$th nearest mobile object to p with a distance of d_{k+1}. An existing technique of snapshot KNN like DIKNN [75] can be used to get the initial $k+1$ nearest neighbors. We define the following three p-centered circumcisers, B_f, B_i, and B_o, the radius of which are $R_f = \frac{d_{k+1}-d_k}{3}$, $R_i = d_k + \frac{d_{k+1}-d_k}{3}$, and $R_o = d_k + 2 \times \frac{d_{k+1}-d_k}{3}$, respectively. These notations are illustrated in Figure 5.3. Clearly, the set of k mobile objects nearest to p does not change as long as the focal mobile object does not moves out of B_f, and no mobile objects cross over B_i or B_o. Thus, we just need to make the focal mobile object to monitor its movement against B_f, and other mobile objects to monitor their movement against B_i and B_o. If a crossing event occurs, the focal mobile object is notified to re-evaluate the query. A more formal description of this algorithm (executed by the focal mobile object) is as follows, where p is the focal mobile object's current position and k is the number of nearest mobile objects to monitor.

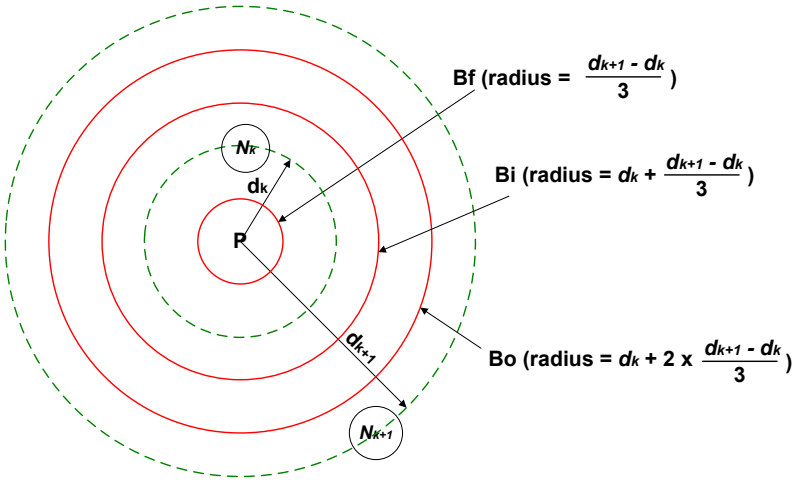


Figure 5.3 Safe Boundaries for an M-KNNMQ

M-KNNMQ(p, k)

1. Find the $k+1$ mobile objects that are nearest to p by searching the cell that contains p (and expand the search scope if necessary);

2. Notify the query creator the k nearest mobile objects;

3. Compute B_f, B_i, and B_o according to p, the positions of the k and $k+1^{th}$ nearest mobile objects;

4. Submit B_i and B_o as two S-RMQs with the additional information that tells a mobile object to do the followings when crossing over B_o or B_i,

   - Remove the two S-RMQs created with B_i and B_o;
   - Notify the focal mobile object to re-evaluate the query by calling **M-KNNMQ**(p, k).

5. Monitor movement against B_f and if moving out of B_f, re-evaluate the query by calling **M-KNNMQ**(p, k);

5.4 Performance Study

Our analysis allows us to conveniently evaluate the performance of our technique under different settings. However, it assumes the network is fully connected, and does not consider the delay in updating query results. In this section, we use our simulator for a more comprehensive performance evaluation. Since our technique supports S-RMQs and the three remaining types of SMQs by converting them into S-RMQs, only S-RMQs are considered in our performance study. The network may be partially disconnected. We simulate a small battlefield where the movement of mobile objects follows the random walk model [21]. The parameters used in our simulation are summarized in Table 5.1. For performance comparison, we have also implemented a baseline approach. In this scheme, a mobile object monitors the mobile population in a region of interest by periodically querying the mobile objects inside the cells that overlap with the region. Our study focuses on the following performance metrics:

- *Communication cost per time unit:* The total number of packets transmitted by each mobile object during the simulation divided by the total simulation time.

- *Miss rate:* The ratio between number of fails to execute queries and the total number of crossing boundaries.

- *Delay:* The period from the time a mobile object actually crosses over a query boundary to the time when the mobile object knows that it has crosses over the boundary. This metric measures the delay in executing a query.

Table 5.1 Simulation Parameters for SMQs

parameter	default	unit
number of mobile objects	300	*objects*
average mobile object speed	5	$meter/sec$
network area	5,000 x 5,000	$meter^2$
cell size	350 x 350	$meter^2$
transmission radius	500	*meter*
number of S-RMQs	100	

In the next subsections, we report how the performance of the two techniques (i.e., proposed and baseline) is affected by the number of mobile objects and mobile object mobility.

5.4.1 Effect of the Number of Mobile Objects

We generated a number of mobile objects, from 100 to 500 and placed them on the network domain. For each simulation, the average of mobile object speed is set to be 5 $meters/second$. In the baseline approach, the result of each query is updated at three different periods (T): 1, 5, and 10 seconds. Figure 5.4(a) shows the communication costs incurred by the proposed approach and the baseline approaches. The baseline approaches are insensitive to the number of mobile objects since the cost of query execution is determined by the frequency of updating query results. However, as the number of mobile objects increases, the communication cost of the proposed approach is initially reduced and then starts to increase. When the number of mobile objects is small (e.g., 100 mobile objects), the network is partially disconnected, and the chance is high that a mobile object moving into a new cell cannot find any mobile object in the cell for relevant queries and therefore has to search for the cell's retaining mobile object. When the number of mobile objects is large, most likely a mobile object will get its new cell's relevant queries within one hop. However, the number of the events of having some

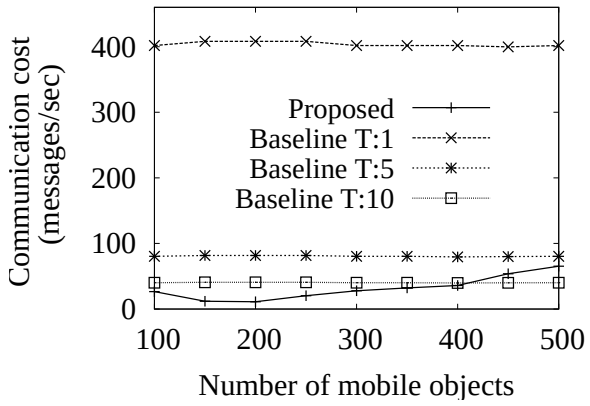
mobile object crossing over a cell boundary increases, which causes the communication cost to increases.

Figure 5.4(b) shows the miss rate of the proposed approach and the baseline approaches. It shows that the miss rate of the baseline approaches is affected by the frequency of updating query results. The more frequent, the more accurate query results will be achieved. However, the baseline approach can never achieve 100% accuracy. In contrast, the proposed technique can guarantee accurate query results when the network is fully connected. When the network is slightly disconnected, it can still achieve nearly 0% miss rate.

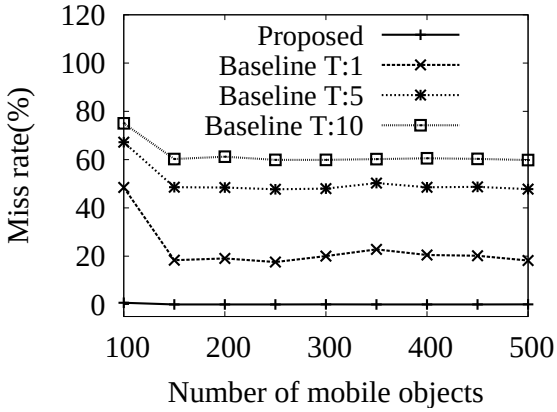
Figure 5.4(c) shows the delays incurred by the two techniques. It shows that the delay caused by the baseline approach initially drops and then becomes stabilized. This is due to the fact that when the number of mobile objects is small, many queries are never actually executed, i.e., the miss rate is very high, as showed in Figure 5.4(b). However, regardless of the number of mobile objects, the delay under the three baseline approaches is always considerably high. In contrast, the proposed technique can achieve almost instant execution of all queries.

5.4.2 Effect of Mobile Object Movement

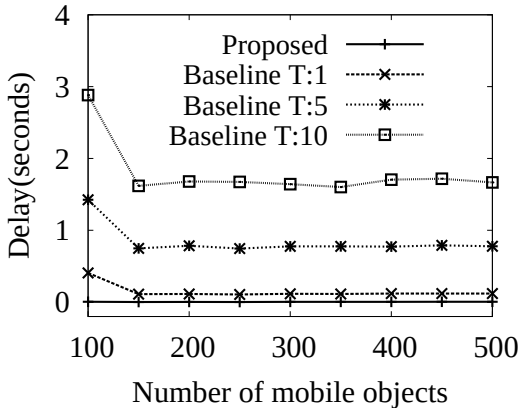
In this study, we generated 300 mobile objects and randomly placed them on the network domain. We vary the average speed of mobile object movement from 1 to 10 $meters/second$. Figure 5.5(a) shows the communication cost incurred by the proposed approach and the baseline approaches. The proposed technique incurs more communication costs as the mobile object mobility increases. This is not surprising because the frequency of crossing cells increases when the average speed of mobile objects increases. Each time a mobile object moves into a new cell, it needs to retrieve the cell's relevant queries. As for the baseline approach, the communication cost is not affected by the mobile object mobility. Figure 5.5(b) shows the miss rate of the proposed approach and the baseline approaches. The proposed approach shows no miss rate regardless of mobile object speeds, because the network is fully connected given 300 mobile objects. As such, each query can be executed as soon as a mobile object crosses over its boundary. As for the baseline approach, the miss rates under the three settings of query frequency



(a) Communication cost

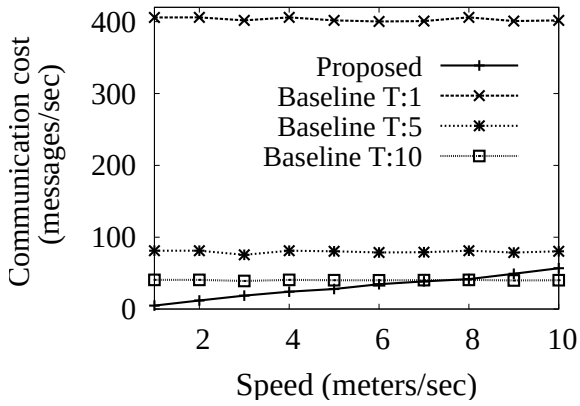


(b) Miss rate

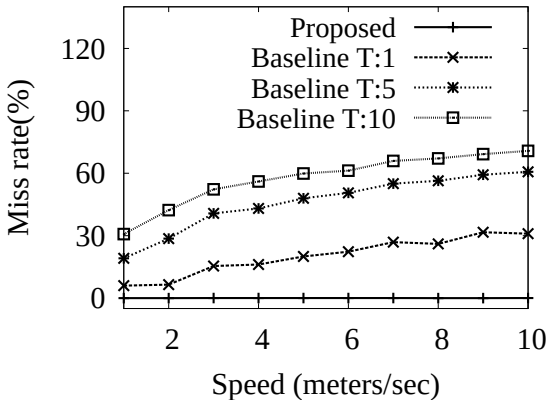


(c) Delay

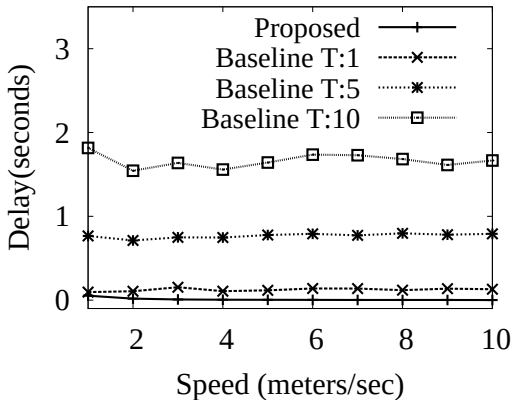
Figure 5.4 Effect of the Number of Mobile Objects



(a) Communication cost



(b) Miss rate



(c) Delay

Figure 5.5 Effect of Mobile Object Movement

increases as the average speed of mobile objects increases. As mobile objects move faster, the chance of crossing a query boundary without knowing the query increases. Figure 5.5(c) shows that the delay in query execution under each technique. Again, the proposed approach is able to guarantee real-time execution while the baseline approaches incur significant amount of latency.

5.5 Conclusion

We have introduced advanced geotasks categorized along the three dimensions: mobile object correlation, geotask mobility, and geotask dependency. We showed how geotasks and the basic management platform can be used to efficiently support stationary range monitoring queries. We showed that other types of mobile range monitoring query and stationary/mobile KNN monitoring query can be converted into a number of S-RMQs. Thus, they can be all supported within one unified platform. Unlike existing research, our technique does not rely on any central server. Instead, mobile objects collaborate in query processing. Our technique is efficient because each mobile object needs to cache only its nearby queries. As a mobile object moves, it can evaluate the queries it knows. As a result, accurate and real-time query results can be provided when the network is fully connected.

CHAPTER 6. CONCLUSION AND FUTURE WORKS

Our geographic environment is a natural storage where we humans store and share data; and wherever we go, we load and process, either consciously or unconsciously, the data implanted around us. Geotasking mimics human interaction with the physical world as humans generate information and store/retrieve information to/from a geographical location. In the proposed works, programs are treated like data and these programs are executed by various types of triggers. We have proposed a generic platform for cost-effective management of various types of geotasks. The proposed approaches support persistent storage and timely execution of geotasks in a fully distributed environment. Geotasking provides a unified solution in a distributed manner.

We classify geotasks into three categories based on mobile object correlation, geotask mobility, and geotask dependency. We have provided a mathematical model of communication cost and evaluated the technique using simulation for basic geotask management. We also provide techniques for converting from various types of geotasks to s-class and stationary geotask using the safe boundary concept. Finally, we propose a generic platform to support these various types of geotasks. We have presented how to use advance geotasks to support stationary range monitoring queries (S-RMQs), and showed that mobile range monitoring query and stationary/mobile KNN monitoring query can be converted into a number of S-RMQs. Thus, they can all be supported within one unified platform.

Safe-Time technique is described in the appendix as an alternative to support one type of geotasks (continuous KNN query). Instead of partitioning a network into grid cells, Safe-Time uses distance information between nodes and a query point to calculate an update interval. United-Safe-Time and Extend-Safe-Time further reduce communication cost.

Our future works are follows. 1) Explore various types of interrelated geotasks. We currently support s-class/m-class in mobile object correlation and support stationary/mobile in geotask mobility. However, current works only support independent geotasks in geotask dependency. Exploring interrelated geotasks and developing corresponding solutions enable the support for more types of geotasks. 2) In our current solution, the network partitioning is pre-determined with a fixed cell size. While this simplifies our technique design, it may not achieve optimal system performance. A larger cell size reduces the chance of a mobile object moving in and out of a cell, and so decreases the communication costs incurred in retrieving relevant queries. However, it increases the cost of installing/removing a query, which needs to be broadcast to all mobile objects inside a cell. Clearly, various factors such as mobile object mobility and query activities need to be considered in order to determine a good cell size. As these parameters may change from time to time, dynamic network partitioning is necessary to ensure consistent good performance.

APPENDIX
SAFE-TIME: AN ALTERNATIVE APPROACH FOR GEOTASK MANAGEMENT

Introduction

The techniques in previous chapters use the grid based approach in which a network is partitioned into disjoint grids In this chapter, we propose an alternative approach for geotask management without grid cells. We consider a continuous k nearest query (cKNN) which is m-class, stationary, and independent geotask for the alternative approach.

To the best of our knowledge, existing solutions for cKNN except two [70, 42] belong to a centralized approach. In the centralized approach, all mobile objects periodically report their locations and other information to a centralized server. The server computes query results and returns them to the query requesters. These schemes are suitable for an infrastructure-based wireless communication environment such as cellular networks with base stations because the communication cost for location updates from all mobile objects is not prohibitively expensive. These solutions, hence, focus on reducing disk I/Os by utilizing index structures at the server to speed up the query execution time.

The two distributed solutions [70, 42] share a common idea to offload some computation from the server to mobile objects in a monitoring area enclosing a query point. The two techniques differ mainly in the calculation of the monitoring area. A cellular network with base stations is assumed in [70] whereas in [42], a road network environment is assumed. In either case, the server informs mobile objects in a monitoring area of a query of the distance (critical distance) between the query point and the k^{th} nearest neighbor (term k^{th} NN hereafter), the

speed, and the direction of the k^{th} NN when the information is changed. Based on the critical distance and the distance between the mobile object and the query object, the mobile object in the monitoring area can determine whether it could change the query result. If so, it reports to the server to compute the correct query result. These two distributed solutions have been shown to reduce server load and communication overhead compared to the centralized solutions.

Nevertheless, in an environment without wireless infrastructures or a server such as pure mobile M-P2P, the requirement to interact with the server to relay information to the monitoring area still incurs expensive communication cost. In addition, the above distributed schemes require the knowledge of directions and speeds of mobile objects.

In this paper, we address the stationary cKNN problem, for M-P2P. Given a set of mobile objects and a set of stationary queries in a 2D geographical space, we are interested in continuously monitoring a set of K nearest objects to each query. Our solution does not require the server, nor the knowledge of current objects' speed and directions. Our contributions are summarized below.

First, we propose *Safe-Time*—the first serverless distributed solution for stationary cKNN for M-P2P. The two key features of Safe-Time are as follows. 1) During a safe-time period of a given query, the query result is guaranteed to remain the same. Hence, there is no need to re-execute the query. 2) Once the safe-time expires, execution of the query involves only mobile objects in a circular band of width equal to the estimated distance between the k^{th} and the $k+1^{th}$ NNs. The two features provide savings in terms of communication cost over the centralized approach. **Second**, we introduce *Unite-Safe-Time* to further reduce communication cost for dense queries. When queries are sufficiently close to each other in a sufficiently dense network, results of these queries are likely similar. Unite-Safe-Time executes one virtual query derived from a number of nearby queries instead of executing each of them separately. We analyzed for a condition when Unite-Safe-Time is more advantageous than Safe-Time. Unite-Safe-Time is used only in such conditions. **Third**, we introduce Extend-Safe-Time to reduce communication cost. This technique increases the length of the safe-time period by $\alpha\%$ of the original safe-

time period. The technique is suitable for the condition when most mobile objects move much slower than their maximum speed. This is because query results may remain same even some time after the safe-time period. We analyze the tradeoff between the increased safe-time period and percentage of missed query results. **Forth**, we evaluate the effectiveness of the proposed algorithms in terms of communication cost under different conditions. In the best case in our study, Safe-Time saves up to 66% of the cost of the centralized approach. Unite-Safe-Time saves up to 78% of the cost of the centralized approach and Extend-Safe-Time saves up to 80% of the cost of the centralized approach.

Proposed Safe-Time Algorithm

Given a set of mobile objects $O = \{o_1, o_2, \ldots, o_n\}$ and a set of stationary queries $Q = \{q_1, q_2, \ldots, q_q\}$ in a 2D geographical space, we continuously monitor a set of K nearest objects to each query. *We consider two query results to be same if they consist of exactly the same objects regardless of the difference in the order of the objects* [71]. For instance, if the first and the second nearest neighbors switch place in the query results that consist of the same objects, the two query results are considered same.

Assumptions: Mobile objects communicate via message relay (i.e., M-P2P) and they move in any moving patterns. We assume only the knowledge of the maximum speed (termed *MaxSpeed*) of all the mobile objects at all time while they are in the network. This MaxSpeed is known in advance, either from the maximum capable speed of all the mobile devices or a speed limit that all mobile objects agree to comply with while they are in this network. Unlike the above two existing techniques, *our solution does not require the server nor the knowledge of velocity vectors of any objects at any point in time.* We assume that a query requester installs/removes its query by geocasting the installation/removal command to an area covering the desired query point. Geocasting enables delivering of a message to all objects inside a target region that can be specified using a circle, a rectangle, or a polygon around the target point. Several efficient techniques have been investigated [26][31][3].

Without loss of generality, we describe Safe-Time with respect to one query, say q_1. The

same algorithm is independently applied to each query in the query set Q. Let $dst^t(q_i, o_j)$ be the Euclidean distance of object o_j from the query point q_i at time t. Let $d^t_{q_1}(o_i, o_j)$ be the absolute distance between object o_j and o_i measured from the query point q_1 at time t. In other words, $d^t_{q_1}(o_i, o_j) = |dst^t(q_1, o_j) - dst^t(q_1, o_i)|$. Recall that we use the notation i^{th} NN to mean the i^{th} nearest neighbor of a query point.

Safe-Time Algorithm

The Safe-Time algorithm consists of the following steps.

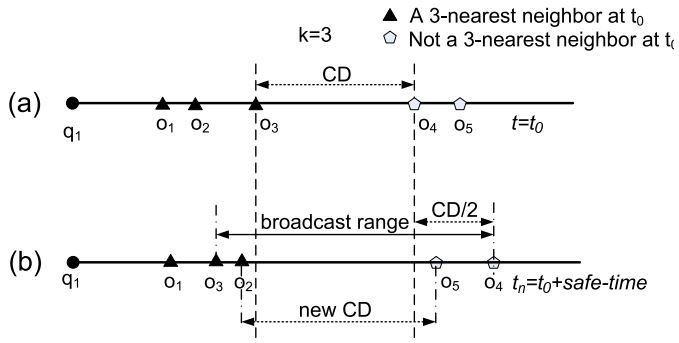


Figure A.1 Safe-Time Algorithm: Distance from the Query Point

- **Initial step:** We adapt a distributed snapshot KNN like DIKNN [75] to get the initial $k+1$ nearest neighbors. The result set is reported to the query requester and the k^{th} NN in this set via greedy forwarding [27]. Let o_k and o_{k+1} denote the k^{th} and the $k+1^{th}$ NNs, respectively. Note that we issue a $k+1$ nearest neighbors query since we also need the $k+1^{th}$ NN in the monitoring step. The $k+1^{th}$ NN is not required by the query requester.

- **Monitoring step:** This step consists of two phases.

  1. **Safe-time period setup:** The k^{th} NN, o_k, uses Equation A.1 to calculate *safe-time period* defined as *a duration after the current time t in which the query result is guaranteed to be the same.*

$$\text{safe-time period} = \frac{\text{cross-distance}}{2 * MaxSpeed}, \tag{A.1}$$

where cross-distance (CD) = $d_q^t(o_k, o_{k+1})$. During the safe-time period, o_k needs not report to the query requester since the query result is the same. This is because at the end of the safe-time period, o_k and o_{k+1} can at most meet only in the middle even though they move toward each other at the maximum speed. In this case, o_k is still a valid k^{th} NN of this query. Note that when they are several objects capable of being the k^{th} NN because they are located at the same distance away from the query, any one of them can be selected as the k^{th} NN. In our case, we choose the one that has already been the k^{th} NN in the previous result. Therefore, during safe-time period, there is no communication cost to update the query result. Once the safe-time period expires, o_k proceeds to the ring broadcast phase to compute the new query result.

Figure A.1(a) illustrates three nearest neighbors ($k = 3$) of q_1 at time t_0, which is $\{o_1, o_2, o_3\}$. Object o_4 is the 4^{th} NN.

2. **Ring broadcast:** The k^{th} NN, o_k, initiates a broadcast to find the new query result and the new k^{th} and $k+1^{th}$ NNs. To reduce the broadcast cost, we compute the *broadcast range* as $d_{q_1}^{t+\text{safe-time period}}(o_k, o_{k+1})$. This is the distance between the current location of o_k and the estimated location of o_{k+1}. Since we do not continuously track the location of o_{k+1}, in the worst case, the furthest o_{k+1} can move is $CD/2$ away from its position in the safe-time period setup phase. Once the broadcast range is (see Figure A.1), o_k broadcasts a *result update request* in a circular band of width equal to this broadcast range. The broadcast message contains the current location of o_k and the estimated location of o_{k+1}.

   Broadcast Technique: Any mobile object in the transmission radius of o_k that is closer to the query point than o_k or is farther than o_{k+1} needs not forward the broadcast message. See Figure A.2 for the broadcast area. For the mobile object in the broadcast range, if it sees the message for the first time, it adds its own location and forwards the message. Otherwise, it does not forward the message. The message with the list of objects in the broadcast ring will arrive back to the

originating o_k. In practice, the mobile object can send redundant messages to cope with message loss.

The object o_k computes the new query result and find the new k^{th} and $k+1^{th}$ NNs. If the new query result is different from the old one, o_k reports the new result to the query requester. The object o_k uses greedy forwarding to tell the new k^{th} NN of the new query result. The new k^{th} NN proceeds to the safe-time period setup phase. The monitoring step repeats for the duration of the query.

Figure A.1(b) shows an example of a broadcast range at time t_0+safe-time period. After the broadcast, o_3 found that o_2 is the new k^{th} NN and tells o_2 that i) the new query result is $\{o_1, o_3, o_2\}$, ii) o_2 is the new k^{th} NN, and iii) o_5 is the new $k+1^{th}$ NN of this query. As a result, o_2, the new k^{th} NN, determines the new safe-time period, based on the cross distance between o_2 and o_5 using Equation A.1.

Note that we can reduce the size of the broadcast message by keeping only information about the k^{th} and $k+1^{th}$ NNs. As the message is being forwarded, the forwarding object updates the information if they can become the new k^{th} and $k+1^{th}$ NN. In Figure A.2, o_3 starts sending a broadcast message to find out new k^{th} NN and $k+1^{th}$ NN. The message contains the query point , k^{th} NN and $k+1^{th}$ NN 's ids and positions (q, o_3, o_4). The message is forwarded to counter clockwise sequentially. During message relaying, if a mobile object's position is closer than the position of k^{th} NN or $k+1^{th}$ NN in the message, the mobile object updates the k^{th} NN or $k+1^{th}$ NN with its id and its position. The message contains only newly updated k^{th} NN and $k+1^{th}$ NN 's ids and positions. o_3 can get the new k^{th} NN and $k+1^{th}$ NN when the message relaying is completed (q, o_3, o_5).

Proofs of Correctness

We show that our Safe-Time generates correct result for a stationary cKNN query.

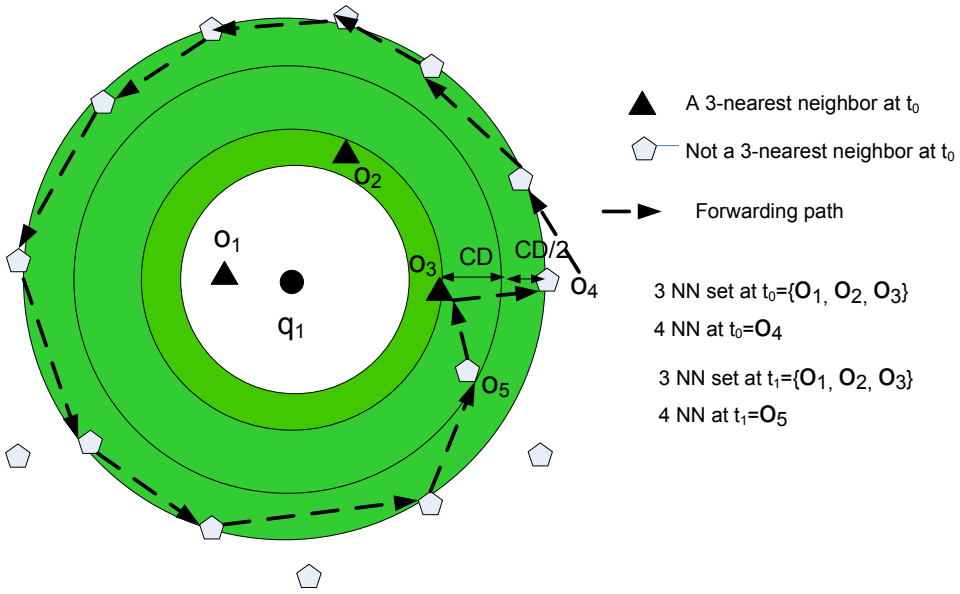


Figure A.2 Circular Band Broadcast Area in a 2D Space

Theorem 1: During the safe-time period, the query result is guaranteed to be the same.

Proof by contradiction: Suppose the query result changes prior to the expiration of the safe-time period. In other words, the k^{th} and the $k+1^{th}$ NNs switch place prior to the expiration of the safe-time period. This cannot happen since the safe-time period is the shortest possible time for these two objects to meet in the middle as they both move toward each other at the maximum speed.

Theorem 2: After the safe-time period has expired, to find the new query result and the new $k+1^{th}$ NN, it is suffice to broadcast in the circular band of width equal to the broadcast range between the k^{th} and $k+1^{th}$ NNs.

Proof: The k^{th} NN knows the current query result. After the safe-time period has expired, this object may become the i^{th} nearest neighbor (where $1 \leq i \leq k$) in the new query result. By Lemma 1, the other $k-i$ objects in the current result cannot move beyond the current position of the $k+1^{th}$ NN. By Lemma 2, other objects not included in the current query result cannot move closer to the query point than the k^{th} NN. Hence, broadcast in the broadcast range between the locations of k^{th} and $k+1^{th}$ NNs in the ring broadcast phase will find the new query result and the new $k+1^{th}$ NN.

Lemma 1: At the end of the safe-time period, objects included in the current query result cannot be farther away from the query point than the $k + 1^{th}$ NN.

Proof by contradiction: Suppose there exists an object in the query result, which have moved farther away from the query point than the $k + 1^{th}$ NN at the end of the safe-time period. This object must originally be farther away from the $k + 1^{th}$ NN by at least the cross distance. To pass the $k + 1^{th}$ NN at the end of the safe-time period, this object must have moved faster than the maximum speed toward the $k + 1^{th}$ NN, which is not possible. Hence, the contradiction.

Lemma 2: At the end of the safe-time period, objects not included the current query result cannot be closer to the query point than the k^{th} NN.

Proof by contradiction: Suppose there exists an object not included in the current query result, which moves closer to the query point than the k^{th} NN at the end of the safe-time period. Since this object is not originally in the query result, its distance from the k^{th} NN must be at least the cross distance. Hence, to be closer to the query point than the k^{th} NN at the time the safe-time period expires, this object must have moved toward the k^{th} NN (which is also moving toward this object) faster than the maximum speed, which is not possible. Hence, the contradiction.

Discussion

Safe-Time relies on the knowledge of the maximum speed of all the mobile objects in the network. If the actual moving speed is much lower than the maximum speed, the safe-time period computed according to Equation 1 will be much shorter than the actual duration in which the query result remains the same. In other words, the safe-time period is too conservative, which incurs unnecessary broadcast cost. If we increase the safe-time period, we can save additional broadcast cost. In section V, we present our Extend-Safe-Time by lengthening the safe-time period. In another case where there are a few objects capable of moving at a much faster speed than the rest of the objects. For instance, in the case of a few tanks versus a large number of moving sensors. We should not consider the speed of these

outlier objects in the computation of the safe-time period. These objects typically have higher communication capability. The query requester can use the result of Safe-Time together with the locations of the outlier objects to obtain the final result.

In M-P2P networks, it is possible that some area of the network becomes disconnected. In this case, it is not possible to have the correct answer regardless of which techniques are used.

Proposed Unite-Safe-Time Algorithm

When queries and objects are sufficiently dense, several nearby queries could generate almost the same query results. In this case, we may save additional communication cost if we execute multiple nearby queries together as one virtual query instead of executing each of them separately. Unite-Safe-Time is proposed to exploit this advantage. Note that the cost referred hereafter is the communication cost.

Analysis of When to Use Unite-Safe-Time

We first analyze the cost of executing m queries separately and compare it with the cost of executing one virtual query that merges these m queries together. To ease the analysis, we assume a uniform distribution for queries and objects in a circular network area. We assume that all the queries require the same number of the nearest neighbors (i.e., same k value). Let N be the total number of mobile objects and $2R$ be the diameter of the network. We focus on the cost incurred due to the monitoring step since this step is performed repeatedly.

Cost of executing one query per time unit

The ring broadcast phase is executed once every time the safe-time period is expired. Hence, the average cost for one query per time unit is the ratio of the broadcast cost to the safe-time period of this query. We define a *critical circle* of a query as the circle centered at the query location with the radius $x = dst(q, o_{k+1})$. In other words, the radius x is the Euclidean distance between the query point and its $k + 1^{th}$ NN. We call this radius *critical radius*. An average broadcast cost is proportional to an average area of a critical circle of a

query. To analyze for this cost, we need to get the average of x or $\bar{x}$. To get $\bar{x}$, we estimate $\bar{r}$—the average of r where $r = dst(q, o_k)$ (the distance from the query point to the k^{th} NN). Equation A.2 assumes the uniform distribution of the objects in the network.

$$\begin{aligned} \frac{k}{\pi \bar{r}^2} &= \frac{N}{\pi R^2} \\ \bar{r} &= R \cdot \sqrt{\frac{k}{N}} \end{aligned} \tag{A.2}$$

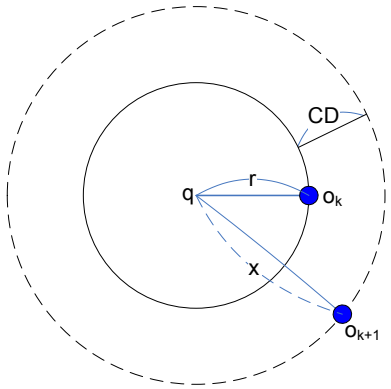


Figure A.3 Relationship among the Query Point, k^{th}, and $k+1^{th}$ NNs

The entire network area is πR^2 and the area of k mobile objects are in πr^2. We know that $k-1$ objects are inside the inner circle in Figure A.3 and the k^{th} NN is on the boundary of this circle. The $k+1^{th}$ NN is on the boundary of the critical circle and the rest of the objects are outside the critical circle. The probability that an object is in the inner circle is $(\frac{1}{\pi R^2}) \cdot \pi r^2$, and the probability that an object is outside the outer circle is $(\frac{1}{\pi R^2}) \cdot (\pi R^2 - \pi x^2)$. The probability that $k-1$ objects among N objects are in the inner circle and the rest of them except k^{th} NN $(N-k-1)$ are outside the outer circle is

$$\binom{N}{k-1} \left(\frac{r}{R}\right)^{2(k-1)} \left(\frac{R^2 - x^2}{R^2}\right)^{N-k-1}. \tag{A.3}$$

Hence, we can compute the average cross distance between the k^{th} and $k+1^{th}$ NNs using Equation A.4.

$$\overline{CD} = \int_r^R (x-r) \binom{N}{k-1} \left(\frac{r}{R}\right)^{2(k-1)} \left(\frac{R^2 - x^2}{R^2}\right)^{N-k-1} dx \tag{A.4}$$

By substituting the average cross distance for the cross distance in Equation A.1, we have

$$\text{safe-time period} \quad = \quad \frac{\overline{CD}}{2 \cdot MaxSpeed}. \tag{A.5}$$

For simplicity, we estimate the area of the broadcast with the area of the critical circle instead of the circular band. Let $\bar{x}$ be the average critical radius, which is equal to $\bar{r} + \overline{CD}$. The average cost per time unit of one query is

$$Cost_{single} = c \cdot \pi \bar{x}^2 \cdot \frac{1}{\text{safe-time period}} \quad , \tag{A.6}$$

where c is the number of messages forwarded per unit area.

Cost of executing a virtual query formed by merging nearby queries

We first consider the simplest case of merging two queries into one virtual query. Intuitively, merging two queries reduces cost when the critical circles of the two queries sufficiently overlap and the safe-time periods of the two queries are about the same. First, we calculate the broadcast area of a merged query which is the union of the critical circles of the two queries. Figure A.4 shows the overlapping critical circles. Since we assume a uniform distribution of the

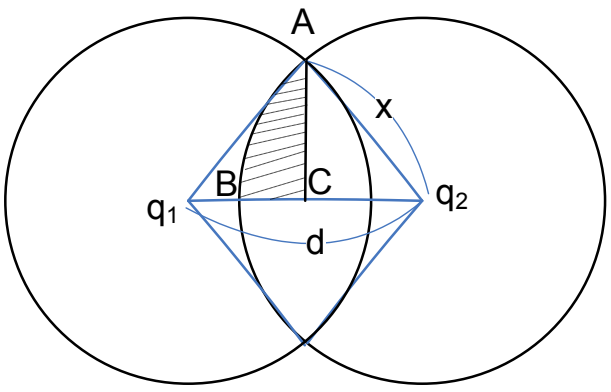


Figure A.4 Two Overlapping Critical Circles

objects, the critical circles of these two queries have the same radius. Let $circle(q_i)$ denote the critical circle of query q_i and x_i be the critical radius of this circle. For the uniform distribution of objects, $x_i = \bar{x}$. Let d be the distance between the two query points. If two critical circles

overlap, we have $d < 2\bar{x}$. The minimum broadcast area is the area of the union of the two critical circles. This area is computed from the sum of the area of the two circles ($2\pi\bar{x}^2$) less the overlapping area as shown in Equation A.7. The overlap is four times the area marked with the dotted cross lines in Figure A.4. The marked area in Figure A.4 is calculated by subtracting the area of the arc (q_2AB) from the area of the triangle ($\triangle q_2AC$).

$$\bigcup_{i=1}^{2} circle(q_i) = 2\pi\bar{x}^2 - 4 \cdot \left(\frac{\bar{x}^2}{2} \cdot \arccos\left(\frac{d}{2\bar{x}}\right) - \frac{d}{4} \cdot \sqrt{\bar{x}^2 - \left(\frac{d}{2}\right)^2} \right) \tag{A.7}$$

Let s_i denote the safe-time period of query q_i. Since a broadcast on the union area is performed once every time the safe-time period expires. The safe-time period of the virtual query is the minimum of the safe-time periods of the two queries. Hence, the average cost per time unit for executing a virtual query is below.

$$Cost_{merge} = c \cdot \left(\bigcup_{i=1}^{2} circle(q_i) \right) \cdot \max(\frac{1}{s_1}, \frac{1}{s_2}) \tag{A.8}$$

Merging condition

We merge two queries as one virtual query when Equation A.9 holds.

$$2 \cdot Cost_{single} - Cost_{merge} > 0 \tag{A.9}$$

The condition for merging m queries is below.

$$c \cdot \left(\sum_{i=1}^{m} \pi x_i^2 - \bigcup_{i=1}^{m} circle(q_i) \cdot \max(\frac{1}{s_1}, \frac{1}{s_2}, \cdots, \frac{1}{s_m}) \right) > 0 \tag{A.10}$$

Unite-Safe-Time Algorithm

We now describe the algorithm for merging two queries.

- **Initial step:** As a new query is added via Geocast, the k^{th} NN of an existing nearby query becomes aware of the new query and the required k. If this existing query is not part of any virtual query and both queries have the same k value, this k^{th} NN considers merging its query with the new query. Greedy forwarding is used as a means of communication between the k^{th} NN of the two queries.

The k^{th} NN of the existing query measures its distance to the new query and sets its wait time directly proportional to the distance. During the wait time, if the object hears another merged request that involves the new query, it discards its own merged request. That is, it does not consider merging with the new query any more. Otherwise, once the wait time expires, this k^{th} NN (also called the merged leader) does the following. It geocasts a merged request to the new query location. The radius of the geocast region is the critical radius of the merged leader, assuming that the new query also has the same critical radius. Therefore, we should find the k^{th} NN of the new query within the geocast region. The k^{th} NN of the new query responds to the merged request with its critical radius and its safe-time period. The use of wait time is to give a chance for the queries closest to each other to be merged.

Once receiving the reply, the merged leader computes Equation A.10 for $m = 2$. If this equation holds, the merged leader sends a commit merged message to the k^{th} NN of the new query. The leader executes the virtual query and sets the safe-time period of the virtual query to be the minimum safe-time period of all the queries that are part of this virtual query.

- **Monitoring step:** This step is performed repeatedly.

  Once the safe-time period is expired, the merged leader broadcasts a *query update* message to the area of the virtual query. The query update message contains a list of the queries that are part of this virtual query and the critical radius of each of the queries. Any object receives this message computes its distance to each of the query locations. The object does not forward the message when its distance is greater than each radius in the list. In other words, this object is not in the union area of all the queries that are merged. The object that is in the union area uses greedy forwarding to send its location to the merged leader and broadcasts the broadcast message to objects in its transmission radius. The merged leader computes the new query result for each of the queries that are part of the virtual query and reports any changes in the query result to the according query requester. The merged leader selects the new k^{th} and $k + 1^{th}$ NNs and tells the

new k^{th} NN to assume the responsibility of the merged leader. The new leader sets the new safe-time period and the monitoring step is repeated.

The initial phase is executed only once. Therefore, after a query becomes a part of one virtual query, it remains part of that query throughout its lifetime and it is not eligible to be merged into another virtual query. Unite-Safe-Time only considers combining two nearby queries. This is because computing the union area of more than two queries is complex.

Extend-Safe-Time

Recall that k^{th} NN updates the query result every safe-time period. In fact, the query result set may continue to remain same after a safe-time period has expired. This is because mobile objects do not always move with their maximum speed. Setting the safe-time period as the original safe-time period plus extra time decreases overall communication cost. However, if the query result is changed during the extra time, k^{th} NN does not have the correct result to update the query requester (i.e. missed update). Lengthening the safe-time period can increase the number of missed updates. To quantify the tradeoff between the safe-time period and the number of missed updates, we define the miss ratio as the total number of missed updates to the total number of actual query result changes in the query. We analyze the relationship between the miss ratio and the safe-time period when we increase the safe-time period by $\alpha\%$ more. Hence, we call this technique Extend-Safe-Time. We first assume that mobile objects have only two movement directions in 1-D space because readers can easily see the change in the order of k^{th} NN and $k+1^{th}$ NN. We then extend our analysis to 2-D space.

Figure A.5 shows positions of mobile objects for different safe-time periods. Let t be the original safe-time period and Δt be $\alpha\%$ of the safe-time period. Let t' be $t + \Delta t$. Let q and o_i denote the position of the query point and the position of i^{th} NN at current time, respectively. We define $P(o_i^t)$ and $P(o_i^{t'})$ to be the position of i^{th} NN at t and the position of i^{th} NN at t', respectively. $P(o_i^t)^+$ and $P(o_i^t)^-$ are the furthest positions from the o_i when i^{th} NN moves with its maximum speed for t time units. If the safe-time period is set as t, the possible positions of 3^{th} NN and 4^{th} NN are $P(o_3^t)^- \leq o_3 \leq P(o_3^t)^+$ and $P(o_4^t)^- \leq o_4 \leq P(o_4^t)^+$, respectively.

$$P(o_i^t)^- = o_i - (MaxSpeed \times t) \qquad P(o_i^{t'})^- = o_i - (MaxSpeed \times t')$$
$$P(o_i^t)^+ = o_i + (MaxSpeed \times t) \qquad P(o_i^{t'})^+ = o_i + (MaxSpeed \times t')$$

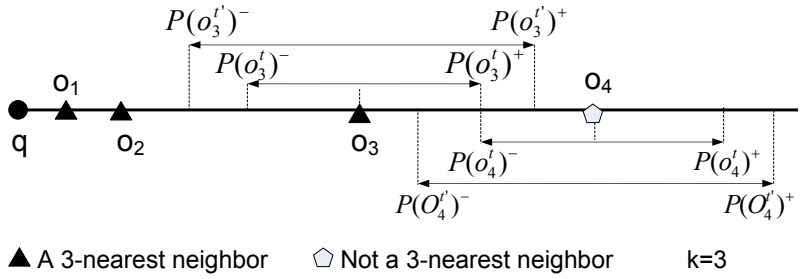


Figure A.5 Extend-Safe-Time

The order of 3^{th} NN and 4^{th} NN is not changed during t (i.e., the miss ratio is 0). If safe-time period is set as t', the order of 3^{th} NN and 4^{th} NN might change during t' since there is an overlapping segment between positions of o_3 and o_4 (i.e., $P(o_4^{t'})^-, P(o_3^{t'})^+$).

To visualize the probability of changing order of 3^{th} NN and 4^{th} NN, we draw a figure in 2-D space. In Figure A.6, X axis represents $P(o_k^{t'})$ and Y axis represents $P(o_{k+1}^{t'})$. Inner rectangles represent areas of $P(o_3^t)$ and $P(o_4^t)$. Outer rectangles represent areas of $P(o_3^{t'})$ and $P(o_4^{t'})$. The shaded triangle represents the area where 3^{th} NN and 4^{th} NN change their order after t'. Let define $Prob(o_k^{t'} > o_{k+1}^{t'})$ be the probability of changing order of k^{th} NN and $k+1^{th}$ NN after t'. The $Prob(o_3^{t'} > o_4^{t'})$ can be calculated by the area of triangle over the union area of the outer rectangles. If the Δt is 0, the area of triangle is 0. Therefore, the probability of changing order of 3^{th} NN and 4^{th} NN is 0. If the Δt is increased, the area of the triangle is also increased and the probability of changing order of 3^{th} NN and 4^{th} NN is also increased.

$$\begin{aligned} UnionArea &= \left(dist(P(o_3^{t'})^-, P(o_3^{t'})^+)\right)^2 + \left(dist(P(o_4^{t'})^-, P(o_4^{t'})^+)\right)^2 \\ &\quad - (2 \cdot MaxSpeed \times \Delta t)^2 \\ &= 2 \cdot (2 \cdot MaxSpeed \times t')^2 - 4 \cdot (MaxSpeed \times \Delta t)^2 \\ &= (8 \cdot t'^2 - 4 \cdot \Delta t^2) \times MaxSpeed^2 \end{aligned} \tag{A.11}$$

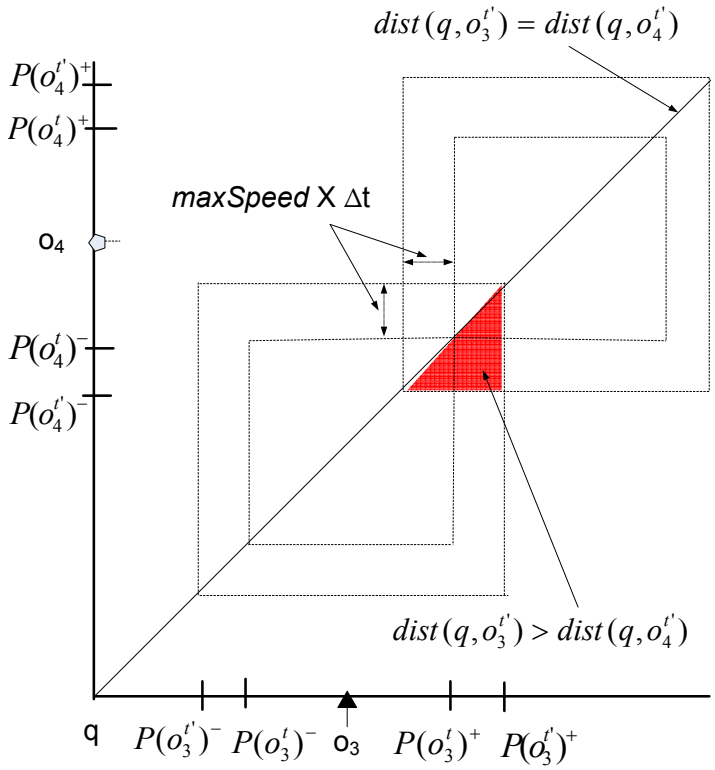


Figure A.6 Shaded Triangle Shows the Condition when o_3 and o_4 Changing Their Order.

The area of the triangle is

$$TriangleArea = \frac{1}{2} \cdot (2 \cdot MaxSpeed \times \Delta t)^2. \tag{A.12}$$

The $Prob(o_3^{t'} > o_4^{t'})$ is

$$\frac{TriangleArea}{UnionArea} = \frac{\Delta t^2}{2 \cdot (2t'^2 - \Delta t^2)} \quad . \tag{A.13}$$

When safe-time period is increased by $\alpha\%$ more (i.e., Δt), the miss ratio is calculated according to Equation A.13.

Performance Study

We evaluate Safe-Time, Unite-Safe-Time, and Extend-Safe-Time techniques in a mobile M-P2P. We use the following two metrics. The communication cost per time unit is calculated as the total number of messages forwarded divided by the simulation time. The miss ratio is calculated as the total number of missed query updates to the total number of actual query result changes during the simulation. We do not compare the proposed schemes with existing server-peers solutions as they were designed for a different network environment. We use *Centralized* as a reference. In this scheme, a central server is located at the center of the network area and every mobile object updates its location to the server every second. *Safe-Time* scheme uses the maximum speed of the mobile object to calculate the safe-time period. In this study, *Unite-Safe-Time* scheme considers merging only two queries. *Extend-Safe-Time* sets the safe-time period as $(100+\alpha)\%$ of the original safe-time period. Queries were randomly distributed in the network area at the start of the simulation. For the movement pattern of objects, we used the random waypoint model [5] in most of our experiments. We generated trajectories of all mobile objects in advance and used these trajectories for evaluation of all schemes for fair comparison. Each data point in our plots is the average of the results from 30 simulation runs. Table A.1 summarizes the parameters used in our performance study. Unless otherwise specified, the default values were used.

We are only interested in relative difference in communication cost among the four schemes under different number of queries, number of mobile objects, and average moving speeds. We

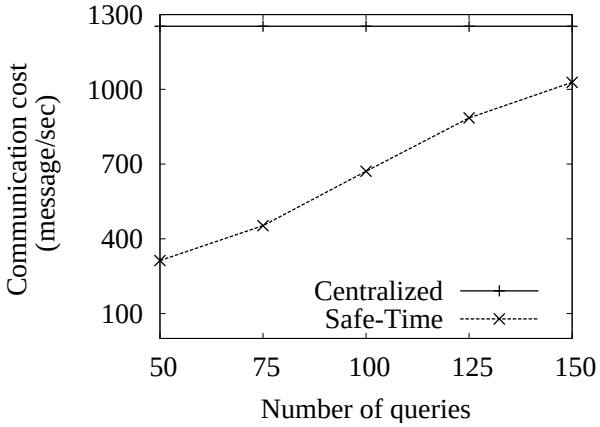
Table A.1 Simulation Parameters for Safe-Time

parameter	range	default	unit
network area	100 x 100	100 x 100	$meters^2$
number of mobile objects	300 - 700	500	*objects*
number of queries	50 - 150	100	*queries*
mean mobile object speed	1 - 3	2	$meter/sec$
max. mobile object speed	2 - 6	4	$meter/sec$
number of nearest neighbors	5	5	*objects*
transmission radius	5	5	*meter*

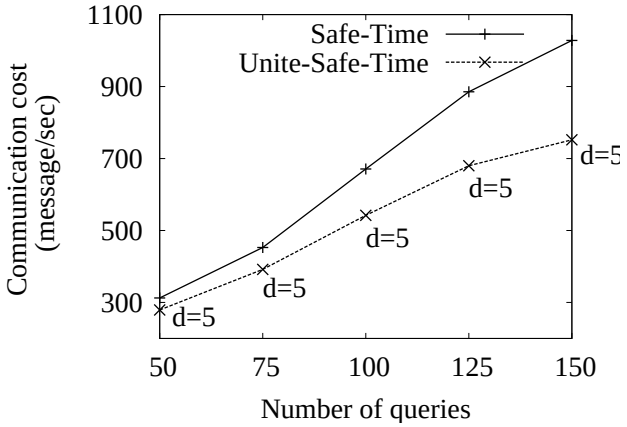
measured the communication cost after all queries were installed in the network. For all the schemes, we did not include the cost of sending the query result to the query requester and the cost of the initial step. In other words, we focus on the repeatedly incurred cost for all the schemes.

Effect of Number of Queries

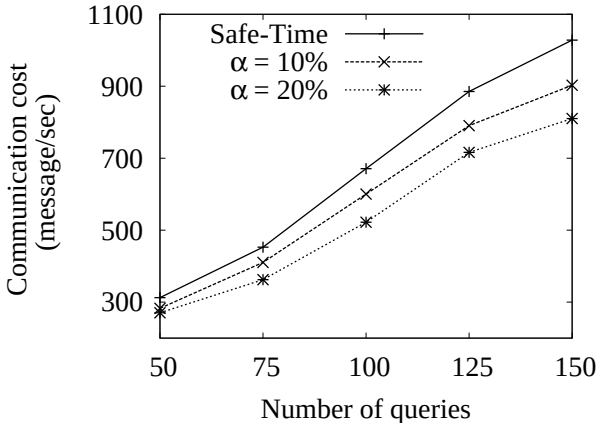
In this study, we varied the number of queries from 50 to 150 queries whereas the other parameters were fixed at their default value. Figure A.7(a) shows that the centralized scheme is not affected by the number of queries. The communication cost of this scheme is mainly due to the frequent location updates of mobile objects. Safe-Time is good for sparse queries. In Figure A.7(a), Safe-Time saves as much as 2/3 of the communication cost incurred by the centralized scheme when 50 queries were distributed in the network area of size about two basketball courts. Note that this scenario is not very sparse. As the number of queries increases, more critical circles of queries overlap. Figure A.7(b) shows the relative performance between Safe-Time and Unite-Safe-Time. In Safe-Time, mobile objects that are in the query results of two or more queries, need to report their locations to the k^{th} NN of each of these queries separately. In Unite-Safe-Time, mobile objects in overlapping critical circles only report their locations to the merged leader. Therefore, the communication cost per time unit of Unite-Safe-Time is less than that of Safe-Time. Figure A.7(b) shows the increasing performance gap between Safe-Time and Unite-Safe-Time with the increasing number of queries. In the figure,



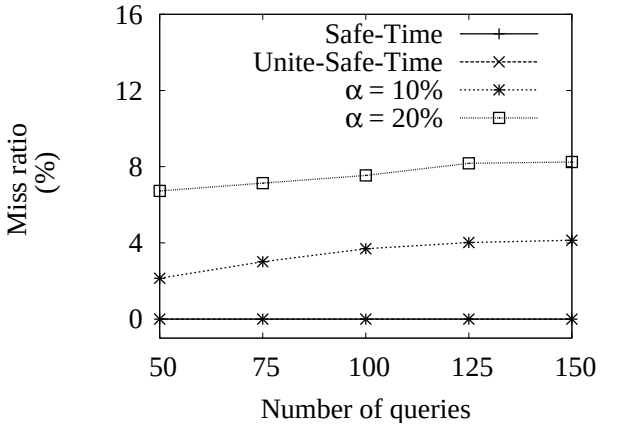
(a) Centralized vs Safe-Time



(b) Safe-Time vs Unite-Safe-Time



(c) Safe-Time vs Extend-Safe-Time



(d) Miss ratio

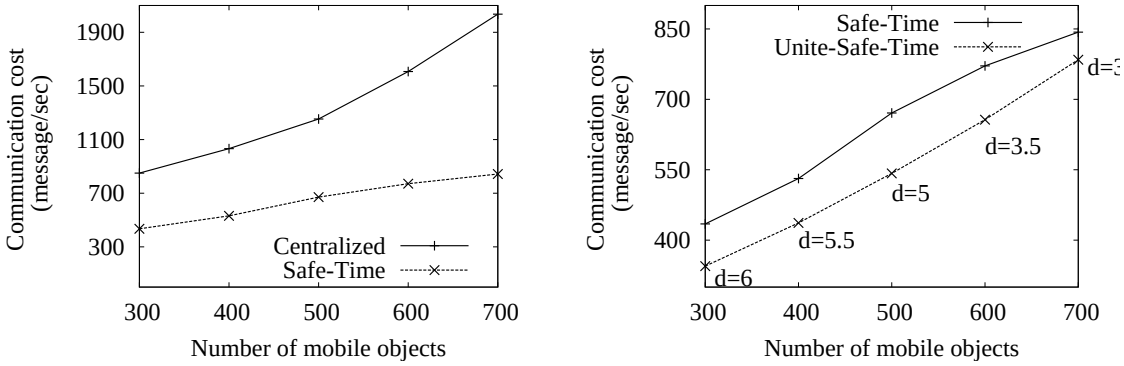
Figure A.7 Effect of Number of Queries

we also show the distance (d) in which two nearby queries are merged using Unite-Safe-Time. As the number of queries increases, the chance of merging two queries also increases. Therefore, the communication cost per time unit is reduced. We can further reduce communication cost further if we merge more than two queries into one virtual query.

Figure A.7(c) shows average communication cost per time unit and the miss ratio of Extend-Safe-Time and Safe-Time. In the best case, 20%-Safe-Time saves 25% and 10% of communication cost incurred by the Safe-Time and Unite-Safe-Time, respectively when 150 queries were distributed in the network area. For comparison with the centralized scheme, 20%-Safe-Time saves between 33% to 80% of communication cost incurred by Centralized. Overall, Safe-Time and its extensions show a large amount of savings in communication cost incurred by the centralized scheme in a sparse network. Extend-Safe-Time incurs less communication cost than Safe-Time does. This is because Extend-Safe-Time has longer safe-time periods than Safe-Time. However, a longer safe-time period incurs a larger miss ratio. In Safe-Time sets the safe-time period based on the maximum speed of all mobile object. Hence, the miss ratio is zero. The miss ratios of both Extend-Safe-Times are small (less than 9%). Extend-Safe-Time shows similar miss ratio regardless of the number of queries.

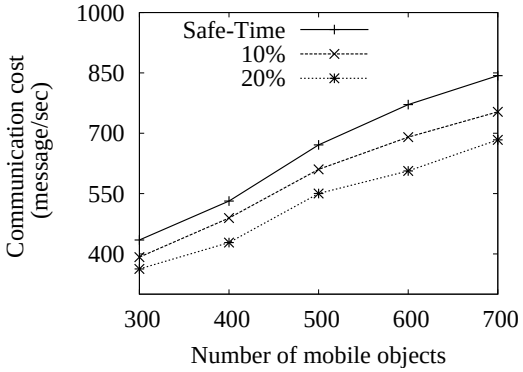
Effect of Number of Mobile Objects

This study investigated the impact of the number of mobile objects on the communication cost. We varied the number of mobile objects from 300 to 700 objects. The results are plotted in Figure A.8. For Centralized, every mobile object reports its location to the server every second. Therefore, the communication cost per time unit increases with the increasing number of mobile objects as shown in Figure A.8(a). In Safe-Time, the safe-time period reduces with the increasing number of mobile objects since the distance between the k^{th} and $k + 1^{th}$ NNs decreases. Therefore, broadcast is done more frequent. However, the broadcast area gets smaller as the critical circle decreases in size. With this trade-off, the cost of Safe-Time slowly increases as shown in Figure A.8(a). Safe-Time saves approximately 66% of communication cost of the centralized scheme in the best case (700 mobile objects) and saves approximately

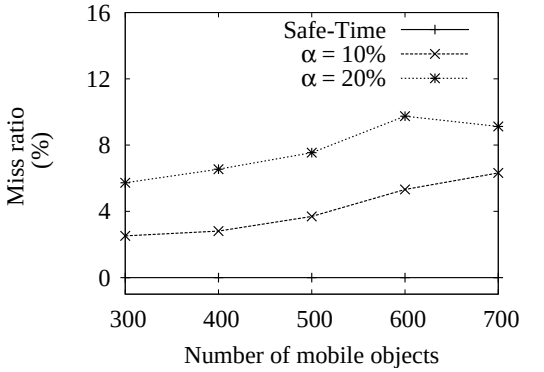


(a) Centralized vs Safe-Time

(b) Safe-Time vs Unite-Safe-Time



(c) Safe-Time vs Extend-Safe-Time



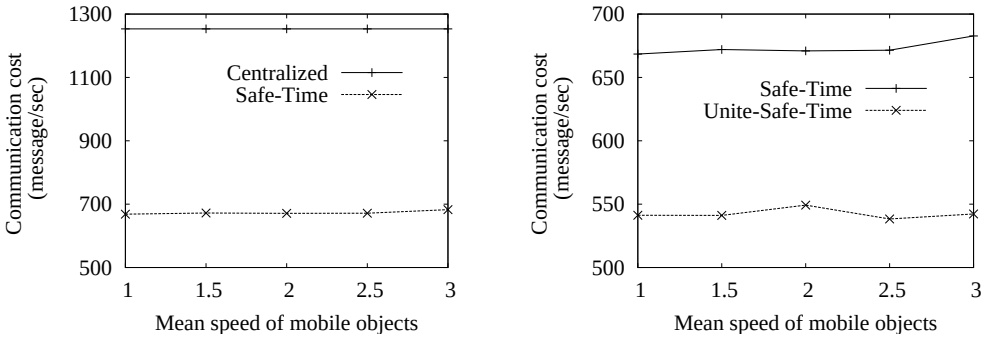
(d) Miss ratio

Figure A.8 Effect of Number of Mobile Objects

50% of communication cost in the worst case (300 mobile objects). Figure A.8(b) shows a slightly decreasing performance gap between Safe-Time and Unite-Safe-Time as the number of mobile objects increases. In fact, when the number of mobile objects is 800 (not shown in the figure), the performance gap is only 34 $message/sec$ with Unite-Safe-Time being slightly lower. The reason is as follows. With the increasing number of objects but fixed number of queries, critical circles get smaller. The overlapped area of two merged queries get smaller, which causes a larger broadcast area. The figure also shows the reduced distance for profitable merging as the network becomes denser. Figure A.8(c) shows decreasing performance gap between Safe-Time and Extend-Safe-Time as the number of mobile objects increases. Extend-Safe-Times incurs less cost than Safe-Time since the longer safe-time period decreases the communication cost per time unit. 20%-Safe-Time saves around 10% to 20% of communication cost incurred by Safe-Time. 20%-Safe-Time saves around 80% of communication cost incurred by Centralized. The miss ratios of Extend-Safe-Time is under 10%. The miss ratio increases as the number of mobile objects increases in Figure A.8(d). This is because the distance between the k^{th} NN and $k+1^{th}$ NN decreases, which increases the chance of changing the result set during the safe-time period.

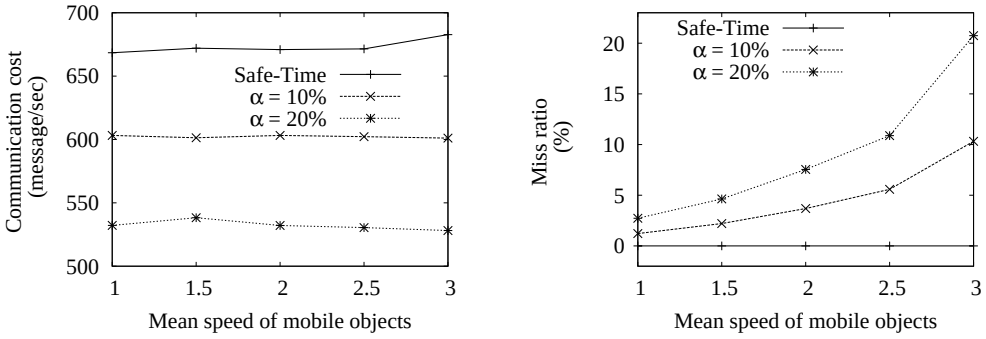
Effect of Mean Speed of Mobile Objects

In this study, we varied mean speeds of mobile objects. we fixed the maximum speed of mobile objects as 4 m/sec. We generated the trajectories of mobile objects based on a normal distribution. We set the mean speed of mobile objects from 1 to 3 m/sec and set the standard deviation to 1 (i.e., $1 \leq \bar{x} \leq 3, \sigma = 1$). If speeds of mobile objects are below 0 or greater than the maximum speed during the generation of trajectories, we set these values to 0 or the maximum speed, respectively. As a result, more than 68% of the speeds of mobile objects are within one standard deviation from the mean speed ($\bar{x} \pm \sigma$). For communication cost per time unit, all schemes are not sensitive to mean speeds. This is because each mobile object in Centralized sends its location information to the server every second and each mobile object in other schemes send its location information to its k^{th} NN based on the maximum mobile



(a) Centralized vs Safe-Time

(b) Safe-Time vs Unite-Safe-Time



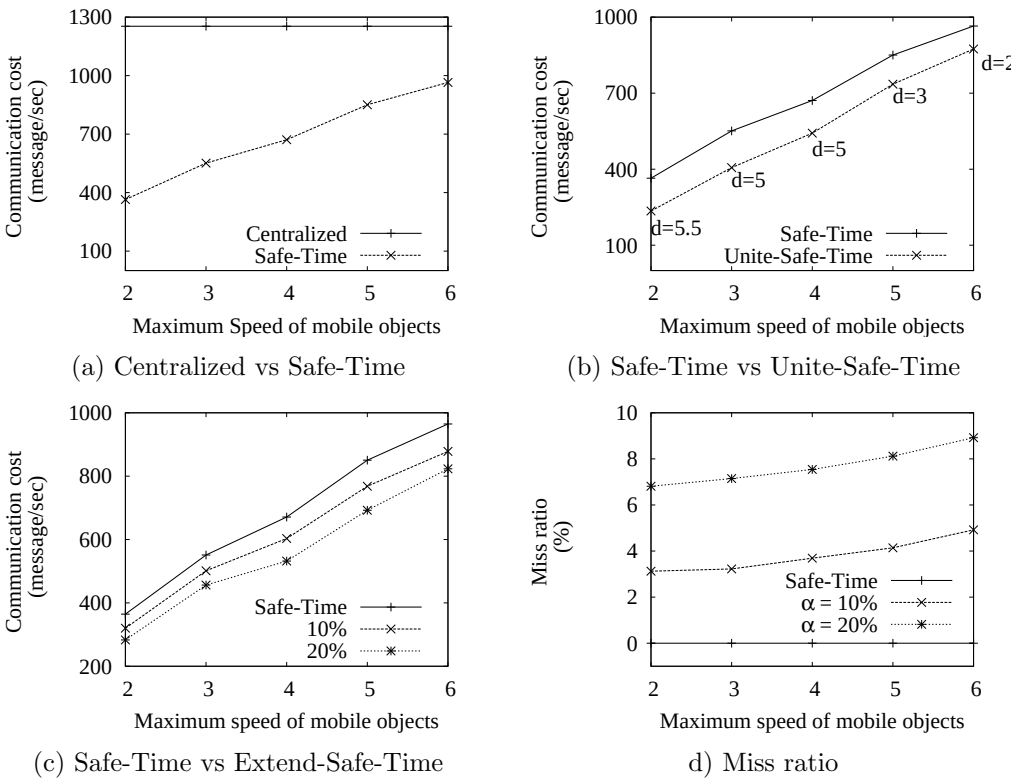
(b) Safe-Time vs Extend-Safe-Time

(d) Miss ratio

Figure A.9 Effect of Mean Speed of Mobile Objects (Maximum Speed Is Fixed at 4 m/sec)

object speed regardless of the mean speed of mobile objects. Safe-Time saves around 50% of communication cost of Centralized and 20%-Safe-Time saves 60% communication cost of Centralized. Figure A.9(d) shows miss ratios of different schemes. Miss ratios of 10%-Safe-Time and 20%-Safe-Time are 1.2% and 2.7% respectively the mean speed of 1. However, miss ratios of those two schemes increase exponentially and are 10.3% and 20.7% respectively the mean speed of 3. Extend-Safe-Time is sensitive to the mean speed of mobile objects. Miss ratios of Extend-Safe-Time increases as the mean speed of mobile objects increases. This is because a result set of a query is easily changed during the safe-time period when a mobile object moves with the speed close to the maximum speed. However, Extend-Safe-Time shows low miss ratio when most mobile objects move much slower than the maximum speed.

Effect of Maximum Speed of Mobile Objects



(a) Centralized vs Safe-Time

(b) Safe-Time vs Unite-Safe-Time

(c) Safe-Time vs Extend-Safe-Time

d) Miss ratio

Figure A.10 Effect of Maximum Speed of Mobile Objects

In this study, we varied the maximum speed of mobile objects from 2 to 6 m/sec. Figure A.10(a) shows that Centralized is not sensitive to the moving speeds since we fixed the location update rate for mobile objects to be one update per second regardless of their speeds. Safe-Time outperforms Centralized the best for the lowest moving speed (saving 66% of communication cost of Centralized). This is because the safe-time period is the longest. The higher the maximum speed, the shorter the safe-time period. The broadcast is done more frequent, resulting in a small gap in the performance difference. The cost gap between Safe-Time and Unite-Safe-Time reduces when the maximum speed of mobile object increases. This is because Unite-Safe-Time chooses the smallest cross distance among all cross distances between each query's k^{th} and $k+1^{th}$ NNs. Although both Safe-Time and Unite-Safe-Time are sensitive to the maximum speed of mobile objects, they give accurate query results. Figure A.10(c) shows communication cost and Safe-Time and Extend-Safe-Time. All schemes are sensitive to the maximum speed of mobile objects. This is because the calculation of the safe-time period is based on the maximum speed of mobile objects. The higher maximum speed incurs the shorter safe-time period, which produces more communication cost. In Figure A.10(d), miss ratios of Extend-Safe-Times scheme slightly increases as the maximum speed of mobile objects increases. The reason is that high speed of mobile objects increases the chance of the result set during α% of the safe-time period. Figure A.10(d) shows that miss ratios of Extend-Safe-Times do not increase as fast as communication cost as the maximum speed of mobile object increases.

Conclusion

We propose Safe-Time—a distributed solution for continuous K-nearest neighbors queries, aiming for mobile-peer-to-peer networks. The two key features are as follows. 1) Actual execution of a cKNN query is not needed during a safe-time period since the query result is guaranteed to remain the same during this period. 2) Once the safe-time period expires, a limited broadcast is done. We introduce Unite-Safe-Time that provides additional savings in communication cost when queries are dense. We also introduce Extend-Safe-Time that

lengthens the safe-time period to save communication cost. Under Extend-Safe-Time, some query result may be missed. The miss ratio is low when the majority of mobile objets move much slower than their maximum speed. Hence, it should be used in applications where a small miss ratio is acceptable.

Our performance study shows that the proposed safe-time algorithms provide savings in terms of communication cost per time unit compared to Centralized. Up to 80% of the savings is shown in our study even for the case when queries are not so sparse. Unite-Safe-Time saves up to 33% communication cost of Safe-Time for dense queries and Extend-Safe-Time saves additionally 25% of Unite-Safe-Time for dense mobile objects in our performance study.

Safe-Time provides the most saving when queries are sparse but with dense mobile objects. Unite-Safe-Time provides low communication cost when query and mobile objects are sparse. Extend-Safe-Time keeps low communication in most cases. However, applications which use this scheme must be tolerable of missed query result.

BIBLIOGRAPHY

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing Moving Points. In *Proc. of ACM Symposium on Principles of Database Systems (PODS'00)*, pages 175–186, 2000.

[2] P. K. Agarwal and S. Har-Peled. Maintaining Approximate Extent Measures of Moving Points. In *Proc. of ACM-SIAM Symposium on Discrete Algorithms*, pages 148–157, Washington, DC, U.S.A, January 7-9, 2001.

[3] Young bae Ko and Nitin H. Vaidya. GeoTORA: A protocol for geocasting in mobile ad hoc networks. In *Proceedings of ICNP*, pages 240–250, 2000.

[4] Bernd A Berg. *Markov Chain Monte Carlo Simulations and Their Statistical Analysis (With Web-Based Fortran Code)*. World Scientific, 2004. ISBN 981-238-935-0.

[5] Christian Bettstetter. Mobility Modeling in Wireless Networks: Categorization, Smooth Movement, and Border Effects. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(3):55–66, 2001.

[6] L. Blazevic, L. Buttyan, S. Giordano, J.-P. Hubaux, and J.-Y. Le Boudec. Self-Organization in Mobile Ad hoc networks: The Approach of Terminodes. In *IEEE Personal Communications*, pages 166–174, June 2000.

[7] Y. Cai and K. A. Hua. An Adaptive Query Management Technique for Efficient Real-time Monitoring of Spatial Regions in Mobile Environments. In *Proc. of the 21st IEEE Int'l Performance, Computing, and Communication Conference (IPCCC'02)*, pages 259–266, Phoenix, AZ, U.S.A, April 2002.

[8] Y. Cai, K. A. Hua, and G. Cao. Processing Range-Monitoring Queries on Heterogeneous Mobile Objects. In *IEEE Int'l Conference on Mobile Data Management (MDM'04)*, pages 27–38, Berkeley, CA, U.S.A, January 19-22, 2004.

[9] Y. Cai, K. A. Hua, G. Cao, and T. Xu. Real-Time Processing of Range-Monitoring Queries in Heterogeneous Mobile Databases. *IEEE Transactions on Mobile Computing*, 5(7):931–942, July 2006.

[10] R. A. Dirckze and Le Gruenwald. A pre-serialization transaction management technique for mobile multidatabases. *ACM Mobile Networks and Applications (MONET)*, 5(4):311–321, 2000.

[11] R. Draves, J. Padhye, and B. Zill. Routing in Multi-radio, Multi-hop Wireless Mesh Networks. In *Proc. of MOBICOM'04*, pages 114–128, 2004.

[12] B. Gedik and L. Liu. MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In *9th International Conference on Extending Database Technology (EDBT'04)*, volume 2992, Heraklion, Crete, Greece, March 14-18, 2004.

[13] B. Gedik, K-L. Wu, P. Yu, and L. Liu. Motion Adaptive Indexing for Moving Continual Queries over Moving Objects. In *The 13th Conference on Information and. Knowledge Management, (ACM CIKM'04)*, pages 427–436, Washington, D.C., USA, 2004.

[14] W. G. Griswold, P. Shanahan, S. W. Brown, R. S. Boyer, M. Ratto, B. R. Shapiro, and T. M. Truong. Activecampus: Experiments in community-oriented ubiquitous computing. *IEEE Computer*, 37(10):73–81, 2004.

[15] L. Gruenwald and S. Banik. A power-aware technique to manage real-time database transactions in mobile ad-hoc networks. In *Proceedings of the 4th International Workshop on Mobility in Databases and Distributed Systems (DEXA)*, September 2001.

[16] Le Gruenwald, Percy Bernedo, and Prasanna Padbmanabhan. Petranet: A power transaction management technique for real time mobile ad-hoc network databases. In *Proceedings*

of the 22nd International Conference on Data Engineering (ICDE), Atlanta, GA, April 2006.

[17] A. Guttman. R-tree: A Dynamic Index Structure for Spatial Search. In *Proc. of ACM SIGMOD*, pages 47–57, Boston, MA, U.S.A, June 1984.

[18] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos. Efficient Indexing of Spatiotemporal Objects. In *International Conference on Extending Database Technology (EDBT'02)*, page 251268, Prague, Czech Republic, March 24-28, 2002.

[19] H. Hu, J. Xu, W. Wong, B. Zheng, D. Lee, and W-C Lee. Proactive Caching for Spatial Queries in Mobile Environments. In *IEEE International Conference on Data Engineering (ICDE'05)*, pages 403–414, Tokyo, Japan, April 2005.

[20] Z. Huang, Christian S. Jensen, Hua Lu, and Beng Chin Ooi. Skyline queries against mobile lightweight devices in manets. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, Washington, DC, USA, 2006.

[21] B.-D. Hughes. *Random Walks and Random Environments*, volume 2. Oxford University Press, 1996.

[22] T. Imielinski, S. Viswanathan, and B.R. Badrinath. Power Efficiency Filtering of Data on Air. In *Proc. 4th Intl Conf. Extending Database Technology (EDBT 94)*, pages 245–258, Cambridge, U.K, March 28-31, 1994.

[23] T. Imielinski, S. Viswanathan, and B.R. Badrinath. Data on Air: Organization and Access. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):353–372, 1997.

[24] Glenn S. Iwerks, Hanan Samet, and Ken Smith. Continuous K-nearest neighbor queries for continuously moving points with updates. In *Proceedings of the 29th international conference on Very large data bases (VLDB'03)*, pages 512–523. VLDB Endowment, 2003.

[25] C. S. Jensen, D. Lin, and B. C. Ooi. Query and Update Efficient B+-Tree Based Indexing of Moving Objects. In *Proc. of VLDB*, pages 768–779, Toronto, Canada, August 29 - September 3, 2004.

[26] X. Jiang and T. Camp. Review of Geocasting Protocols for a Mobile AdHoc Network. In *Proceedings of the Grace Hopper Celebration (GHC)*, 2002.

[27] B. Karp and H. T. Kung. GPSR: Greedy Perimeters Stateless Routing for Wireless Network. In *Proc. of ACM MOBICOM'00*, pages 243 – 254, Boston, MA, U.S.A, 2000.

[28] R. Karrer, A. Sabharwal, and E. Knightly. Enabling Large-scale Wireless Broadband: The Case for TAPs. *SIGCOMM Computer Communication Review*, 34(1):27–32, 2004.

[29] K. Kim, Y. Cai, and W. Tavanapong. A Priority Forwarding Technique for Efficient and Fast Flooding in Wireless Ad Hoc Networks. In *14th Int'l Conference on Computer Communications and Networks (IC3N)*, pages 223–228, San Diego, CA, U.S.A, October 17-19, 2005.

[30] Y. Ko and N. H. Vaidya. Location-Aided Routing (LAR) Mobile Ad Hoc Networks. In *Proc. of ACM International Conference on Mobile Computing and Networking MOBICOM98*, pages 66–75, Dallas, TX, U.S.A, 1998.

[31] Y. Ko and Nitin H. Vaidya. Geocasting in mobile ad hoc networks: Location-based multicast algorithms. In *Proceedings of IEEE WMCSA*, pages 101–110, 1999.

[32] Young-Bae Ko and Nitin H. Vaidya. GeoTORA: A protocol for geocasting in mobile ad hoc networks. In *Proceedings of the IEEE International Conference on Network Protocols(ICNP)*, pages 240–249, Osaka, Japan, November 2000.

[33] G. Kollios, V. J. Tsotras, D. Gunopulos, A. Delis, and M. Hadjieleftheriou. Indexing Animated Objects Using Spatiotemporal Access Methods. *IEEE Transaction on Knowledge and Data Engineering (TKDE)*, 13(5):758–777, 2001.

[34] W-S. Ku and R. Zimmermann. Location-based spatial queries with data sharing in mobile environments. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006*, Atlanta, GA, U.S.A, April 3-8 2006.

[35] W-S. Ku, R. Zimmermann, C-W. Wan, and H. Wang. MAPLE: A Mobile Scalable P2P Nearest Neighbor Query Model for Location-based Services. In *Proceedings of the 22nd International Conference on Data Engineering, (ICDE'06)*, pages 182–222, Atlanta, 2006.

[36] K. 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 *7th International Conference on Database Systems for Advanced Applications, DASFAA'01*, pages 218–225, Hong Kong, China, April 2001.

[37] D. Lee, B. Zheng, and W-C Lee. Data management in location-dependent information services. *IEEE Pervasive computing*, pages 65–72, 2002.

[38] W-C. Lee and B. Zheng. DSI: A Fully Distributed Spatial Index for Location-based Wireless Broadcast Services. In *25th International Conference on Distributed Computing Systems (ICDCS 2005)*, pages 349–358, Columbus, OH, U.S.A, June 6-10, 2005.

[39] Yifan Li, Jiong Yang, and Jiawei Han. Continuous K-Nearest Neighbor Search for Moving Objects. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04)*, page 123, Washington, DC, USA, 2004. IEEE Computer Society.

[40] D. Lin, C.S. Jensen, B. C. Ooi, and S. Saltenis. Efficient Indexing of the Historical, Present, and Future Positions of Moving Objects. In *International Conference on Mobile Data Management (MDM'05)*, pages 59–66, Ayia Napa, Cyprus, May 9-13, 2005.

[41] B. Liu, W.-C. Lee, and D.L. Lee. Distributed Caching of Multi-dimensional Data in Mobile Environments. In *The Sixth International Conference on Mobile Data Management (MDM'05)*, pages 229–233, Ayia Napa, Cyprus, May 9-13, 2005.

[42] Fuyu Liu, Kien A. Hua, and Tai T. Do. A P2P Technique for Continuous k-Nearest-Neighbor Query in Road Networks. In *Proceedings of the 18th International Conference on Database and Expert Systems Applications (DEXA'07)*, pages 264–276, September 2007.

[43] K. Middaugh. *No more Towers*. http://www.govtech.net/digitalcommunities/story.php?id=90189, 2004.

[44] MIT Roofnet. http://www.pdos.lcs.mit.edu/roofnet/.

[45] M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable Incrementable Processing of Continuous Queries in Spatio-temporal Databases. In *SIGMOD '04*, pages 623–634, Paris, France, 2004.

[46] M. Motani, V. Srinivasan, and P. Nuggehalli. PeopleNet: Engineering a Wireless Virtual Social Network. In *Proceedings of the 11th annual international conference on Mobile computing and networking, (MOBICOM'05)*, pages 243–257, Cologne, Germany, August 28 - September 2 2005.

[47] Kyriakos Mouratidis, Dimitris Papadias, and Marios Hadjieleftheriou. Conceptual Partitioning: an Efficient Method for Continuous Nearest Neighbor Monitoring. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data (SIGMOD'05)*, pages 634–645, New York, NY, USA, 2005. ACM.

[48] J. C. Navas and T. Imielinski. GeoCast – Geographic Addressing and Routing. In *Proc. of ACM MOBICOM97*, pages 66–76, Budapest, Hungary, 1997.

[49] J. M. Patel, Y. Chen, and V. P. Chakka. STRIPES: An Efficient Index for Predicted Trajectories. In *Proceedings of the 2004 ACM International Conference on Management of Data (SIGMOD'04)*, pages 637–646, Paris, France, 2004.

[50] M. Pelanis, S. Saltenis, and C. S. Jensen. Indexing the Past, Present and Anticipated Future Positions of Moving Objects. *IEEE Transactions on Knowledge and Data Engineering TKDE*, 31(1):255–298, March 2006.

[51] P. Persson and P. Fagerberg. Geonotes: A real-use study of a public location-aware community system. In *Technical Report SICS-T2002/27-SE, SICS, University of Goteborg*, Sweden, 2002.

[52] D. Pfoser, C. S. Jensen, and Y. Theodoidis. Novel Approaches in Query Processing for Moving Objects. In *Proceedings of the 26th International Conference on Very Large Data Bases VLDB'00*, pages 395–406, Cairo, Egypt, September 10-14, 2000.

[53] K. Porkaew, I. Lasaridis, and S. Mehrotra. Querying Mobile Objects in Spatio-Temporal Databases. In *7th International Symposium on Spatial and Temporal Databases SSTD*, page 5978, Redondo Beach, CA, U.S.A, July 2001.

[54] S. Prabhakar, Y. Xia, D. Kalashnikov, W. G. Aref, and S. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Transactions on Computers*, 15(10):1124–1140, October 2002.

[55] R. Ramakrishnan and J. Gehrke. *Database Management Systems, Third Edition.* McGraw-Hill, 2002.

[56] S. Ratnasamy, D. Estrin, R. Govindan, B. Karp, and S. Shenker. Data Centric Storage in Sensornets. In *ACM SIGCOMM'02*, Pittsburgh ,PA ,USA, August 19-23, 2002.

[57] S. Ratnasamy, B. Karp, L. Yin, and F. Yu. GHT: A Geographic Hash Table for Data Centric Storage. In *ACM Int'l Workshop on Wireless Sensor Networks and Applications (WSNA)*, Atlanta, GA, U.S.A, September 2002.

[58] Q. Ren and M. H. Dunham. Using Semantic Caching to Manage Location Dependent Data in Mobile computing. In *Proceedings of the MOBICOM Conference*, pages 210–221, Boston, MA, USA, April 2000.

[59] E. Royer and C. E. Perkins. Multicast Operation of the Ad-hoc On-Demand Distance Vector Routing Protocol. In *Proc. of ACM MOBICOM99*, pages 207–218, Seattle, WA, U.S.A, August 1999.

[60] S. Saltenis and C. S. Jensen. Indexing of Moving Objects for Location-based Services. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, page 463472, San Jose, CA, U.S.A, Feb. 26 - Mar. 1 2002.

[61] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *ACM Proc. of SIGMOD'00*, pages 331–342, Dallas, TX, U.S.A, May 16-18, 2000.

[62] D. Sarkar. *City of Ripon goes wireless*. http://www.fcw.com/article89302-06-20-05-Print, 2005.

[63] Seattle Wireless. http://www.seattlewireless.net.

[64] Z. Song and N. Roussopoulos. K-nearest Neighbor Search for Moving Query Point. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD'01)*, pages 79–96, London, UK, 2001.

[65] J. Sun, D. Papadias, Y. Tao, and B. Liu. Querying about the Past, the Present and the Future in Spatio-Temporal Databases. In *20th International Conference on Data Engineering (ICDE'04)*, pages 202–213, Boston, U.S.A, March 30-April 2 2004.

[66] Y. Tao and D. Papadias. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *Proc. of the 27th VLDB Conference*, pages 431–440, Roma, Italy, 2001.

[67] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *Proc. of International Conference on Very Large Data Bases (VLDB'02)*, pages 287–298, Hong Kong, China, August 20-23, 2002.

[68] Y. Tao, D. Papadias, and J. Sun. The TPR*-Tree: An Optimized Spatio-temporal Access Method for Predictive Queries. In *Proceedings of 29th International Conference on Very Large Data Bases VLDB*, pages 790–801, Berlin, Germany, September 9-12, 2003.

[69] R. Thomas, H. Gilbert, and G. Mazziotto. Influence of the Moving of the Mobile stations on the Performance of a Radio Cellular Network. In *Proc. of Third Nordic Seminar*, Copenhagen, Denmark, September 1988.

[70] W., W. Guo, and K. Tan. Distributed Processing of Moving K-Nearest-Neighbor Query on Moving Objects. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE'07)*, pages 1116–1125, April 2007.

[71] W. and K. Tan. iSEE: Efficient Continuous K-Nearest-Neighbor Monitoring over Moving Objects. *International Conference onScientific and Statistical Database Management (SSDBM 2007)*, 0:36–45, 2007.

[72] Wireless Public Safety Data Networks Operating on Unlicensed Airwaves: Overview and Profiles.
http://www.newamerica.net/Download_Docs/pdfs/
Doc_File_2633_1.pdf.

[73] O. Wolfson, S. Chamberlain, L. Jiang, and G. Mendez. Cost and Imprecision in Modeling the Position of Moving Objects. In *Proc. of Intl. Conf. Data Engineering*, pages 588–596, Orlando, FL, USA, February 23-27, 1998.

[74] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In *Proc. of the 10th Int'l Conference on Scientific and Statistical Database Management*, pages 111–122, Capry, Italy, July 1998.

[75] S. Wu, K. Chuang, C. Chen, and M. Chen. DIKNN: An Itinerary-based KNN Query Processing Algorithm for Mobile Sensor Networks. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE'07)*, pages 456–465, April 2007.

[76] X. Wu. VPDS: Virtual Home Region based Distributed Position Service in Mobile Ad Hoc Networks. In *Proceedings on 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 113–122, Columbus, OH, U.S.A, June 06-10, 2005.

[77] X. Xiong, M. Mokbel, W. Aref, S. Hambrusch, and S. Prabhakar. Scalable Spatio-temporal Continuous Query Processing for Location-aware Services. In *Proc. of the International Conference on Scientific and Statistical Database Management (SSDBM)*, Santorini, Greece, June 2004.

[78] Xiaopeng Xiong, Mohamed F. Mokbel, and Walid G. Aref. SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 643–654, 2005.

[79] J. Xu, B. Zheng, W.-C. Lee, and D.L. Lee. Energy Efficient Index for Querying Location-Dependent Data in Mobile Broadcast Environments. In *International Conference on Data Engineering (ICDE'03)*, pages 239–250, Bangalore, India, March 5-8, 2003.

[80] J. Xu, B. Zheng, W.-C. Lee, and D.L. Lee. The D-tree: An Index Structure for Planar Point Queries in Location-Based Wireless Services. *IEEE Transaction on Knowledge and Data Engineering*, 16(12):1526–1542, 2004.

[81] X. Yu, K.Q. Pu, and N. Koudas. Monitoring k-Nearest Neighbor Queries over Moving Objects. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 631–642, 2005.

[82] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based Spatial Queries. In *Proceedings of the MOBICOM Conference*, pages 443–454, San Diego, CA, U.S.A, September 14-19, 2003.

[83] B. Zheng and D. L. Lee. Semantic Caching in Location-dependent Query Processing. In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases*, pages 97–116, Redondo beach, CA, U.S.A, July 12-15, 2001.

[84] B. Zheng, W.-C. Lee, and D.L. Lee. Search Continuous Nearest Neighbors on the Air. In *IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, pages 297–304, Dallas-Fort Worth,TX, U.S.A, March 23-26, 2003.

[85] B. Zheng, W.-C. Lee, and D.L. Lee. Search continuous nearest neighbors on the air. In *the First International Conference on Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous'04)*, pages 236–245, Boston, MA, U.S.A, August 22-26 2004.

[86] B. Zheng, W.-C. Lee, and D.L. Lee. Spatial Queries in Wireless Broadcast Systems. *ACM Wireless Networks Journal (WINET)*, 10(4):411–427, 2004.

[87] B. Zheng, J. Xu, W-C. Lee, and D.L. Lee. Grid-Partition Index: A Hybrid Method for Nearest-Neighbor Queries in Wireless Location-Based Services. *Very Large Data Base (VLDBJ)*, 15(1):21–39, 2006.