博士学位申請論文

A STUDY OF PEER-TO-PEER SYSTEMS FOR SPATIAL DATA SHARING

(空間データ共有のためのピアツーピアシステムに関する研究)

指導教員：瀬崎　薫　助教授

東京大学大学院　情報理工学系研究科
電子情報学専攻

47406　魏　新法
(Xinfa WEI)

A STUDY OF PEER-TO-PEER SYSTEMS FOR SPATIAL DATA SHARING

by

XINFA WEI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Department of Information and Communication
Engineering, Graduate School of Information Science and Technology
The University of Tokyo, 2006

Tokyo, Japan

ABSTRACT

This thesis is about peer-to-peer systems for sharing spatial data. In recent years, tremendous improvements in data gathering techniques have contributed to an unprecedented growth of available spatial data at geographically distributed locations. This has created a strong motivation for the efficient sharing of such data. While peer-to-peer systems becomes an important approach of massively distributed systems not only for file transfers but also for searchable data network, in this thesis we study the methods of sharing geographically distributed data with P2P networks. By now, there are a number of P2P protocols proposed, but most of them are based on distributed hashing tables, such as CAN and Chord which support exact match only and have limited number of search predicates support. When we consider a set of peer nodes as a massively geographically distributed database, several types of search predicates should be provided in addition to exact match search. So in order to efficiently support spatial data sharing peer-to-peer application, we need to design new peer-to-peer systems that broaden the types of query processing, improve the performance and are of fault-tolerance.

This thesis first presents the design and evaluation of GNet, an early work of exploring the possibility of geographical peer-to-peer protocol that targets supporting wide area location-based service. The GNet protocol uses hierarchical geographic address as the identifiers of peer nodes. By combining domain-progressive routing mechanism like plaxton mesh with geographical domain hierarchy, this protocol has the advantages of efficient routing, locality preserving, etc. It supports position-based and especially geographically scoped operations efficiently. Though implementation prerequisites limits its application area, analysis and evaluation results demonstrate its scalability, query efficiency and load balancing features, which makes it adaptable to certain applications.

As the main contribution of this thesis, DHR-Trees peer-to-peer protocol is presented with its structure design, collaborative multidimensional query method, maintenance method and the cost analysis, and methods to strengthen fault-tolerance of structure and

query execution under dynamic network environment. Essentially, DHR-Trees structure is the first peer-to-peer structure that has semi-independent R-Trees structure and supports region-based multidimensional search predicates, such as range queries and nearest neighbor queries as in R-Trees structure, while dealing with network dynamism efficiently as well.

The thesis presents the structure details of DHR-Trees. Instead maintaining a global centralized R-Trees index, each peer owns a semi-independent partial region tree structure, which makes it possible to keep correctness of structure even under dynamic network changes. Each geographically distributed peer node is identified by its Hilbert value on a Hilbert space filling curve, by which the peer's two-dimensional geographical location is mapped to one-dimensional identifier. Peer nodes self-organize into a virtual ring topology, sorted by the identifiers. As the core part of the DHR-Trees' protocol, each peer maintains a routing table, which contains two principal parts: For routing purpose, it holds pointers to a number of nodes in the network; for supporting spatial query purpose, the region information of sub-trees in the DHR-Trees is contained.

Spatial queries are executed in a distributed fashion by collaborative efforts among peers. DHR-Trees mainly provides three spatial query functions: point query, range query and nearest neighbor query. By exploiting region information in routing table, the spatial query evaluation results show that DHR-Trees can execute spatial queries much more efficiently than its competitor, the Squid P2P protocol. Furthermore, the nearest neighbor query, one of most important spatial queries which is unsupportable in Squid, can also be efficiently executed.

DHR-Trees faces network churn problem as well as other P2P systems. To keep the system working properly while nodes join, leave, and fail on their own agenda, each peer node is required to maintain both the ring structure and routing table. To maintain ring structure, it uses similar ring stabilization approach as in Chord protocol. For routing table maintaining, processes includes ping, stabilization, and notification process are run periodically or triggered by routing table change events. Our analysis and evaluation result shows that the overhead of updating routing tables when a new node joins or fails increases nearly logarithmically to the network size. This demonstrates the scalability of DHR-Trees peer-to-peer system.

To improve the fault-tolerance on spatial query support, DHR-Trees proposes two ap-

proaches: entry successor list and adaptive bounding rectangle. By introducing successor lists to the entries in the routing table, robustness and resilience are greatly improved. Moreover, to eliminate the frequent updating requirements of the region information in harsh churn environment, we introduce the usage of adaptive bounding rectangle as the replacement of minimum bounding rectangle. This approach decreases the updating overhead and greatly improves the quality of query result under churn.

Through this thesis, two new novel peer-to-peer protocols are provided. Both GNet and DHR-Trees are designed to be architectures for sharing geographically distributed spatial data. In particular, the DHR-Trees can not only index spatial data as in centralized R-Trees, but also be able to handle dynamism in the peer-to-peer network. We believe our approaches can help realization of certain distributed spatial data sharing applications. We hope our works will stimulate more research interest in both peer-to-peer structures and spatial data sharing applications.

ACKNOWLEDGMENTS

A lot of people have provided social and technical supports to the research work presented in this thesis. First and foremost I'd like to thank my advisor professor Kaoru Sezaki, for his comments about this research and for always encouraging me and supporting me all these years.

I would like to thank Professor Asano, Professor Asami, Professor Morikawa, Professor Matsuura, and Professor Kamijo, for their valuable comments and advises at my doctoral dissertation.

Thanks to all members in the Sezaki laboratory, especially to Mr. Konitoshi Komatsu, Mrs. Kaho Matsumoto and Mr. Konomi from Sezaki laboratory, University of Tokyo for their daily support on my research works.

I owe my deepest thanks to my family - my parents, my wife and children. Words cannot express the gratitude I owe them. Without their love, patience and support, I would not have been finished the thesis.

Table of Contents

List of Figures

List of Symbols and Abbreviations

N	The Network Size, the number of nodes of the network.
d	Order of DHR-Trees, the number of entries at a routing level ranges from $[d, 2d]$.
D	The average fan-out of a routing level; In DHR-Trees structure, $D = \frac{3d}{2} - 1$.
m	The number of non-empty entries in a routing level.
z	The length of successor list. For both ring structure and routing table entries.
$HCode$	The Hilbert value of given multidimensional point. Obtained by mapping through Hilbert Space Filling Curve.
H	The height of the routing table.
n	A peer node n.
p,q	Peer nodes p, q.
M_i	Messages at routing level i.
C_i	Routing table chagnes at routing level i.
P_U	The probability of changing Upper Bound U at above routing level.
P_{U_i}	The P_U at routing level i.
P_{mr}	The probability of changing next routing entry.
P_{mr_i}	The P_{mr} at routing level i.
P_{ru}	The probability of changing Upper Bound U at above routing level, indirectly by influencing next entries.
P_{ru_i}	The P_{ru} at routing level i.
S_{msg}	The sum of messages.
ABR	Adaptive Bounding Rectangle.
DCN	Domain Contact Node.

DHT	Distributed Hash Tables.
GID	Geographical Identifier.
$HSFC$	Hilbert Space Filling Curve.
MBR	Minimum Bounding Rectangle.
DHR-Trees	Distributed Hilbert R-Trees.
GNet	Geographic address-based peer-to-peer Network.

Chapter 1

Introduction

1.1 Motivation

In recent years, tremendous improvements in data gathering techniques have generated an unprecedented growth of available spatial data at geographically distributed locations. Technical progress such as wireless communications, cheap sensor device, and location tracking systems such as GPS, encourage end users to gather data locally and voluntarily share with other users.

Suppose there are many geographically distributed spatial data sources as illustrated in Figure 1.1(a). Such data sources may be video camera monitor, sensor data collection, real estate information, and so on. The volume of the data may be huge, and they are probably updated frequently, i.e. gathering data from physical environment periodically to keep the data freshness. To share these spatial data, using centralized approach is not appropriate because the lack of scalability with the huge volume of data and requirement of frequently updating.

Moreover, the spatial data collecting and sharing may be conducted by end users voluntarily in an ad-hoc fashion. Such service providers have more freedom of decision on data contents to share and sharing agenda. Different from traditional system architecture, the service providers and consumers forms into a loosely-constrained network, in which membership may change frequently and freely. This implies new challenges to traditional system architecture and topology.

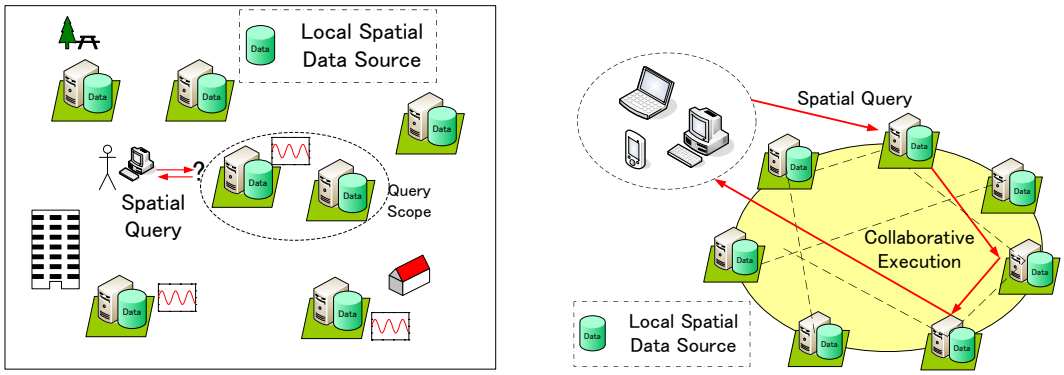
The Peer-to-Peer network has attracted many interest in both Internet users and research community. Rather than requiring the mediation or support of a centralized server

or authority, it is designed for sharing computer resources (data content, CPU cycles) by direct exchanges. There is no official definition for Peer-to-Peer network, we cite a definition from literature [6] as:

. *Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority.*

P2P becomes an important approach of massively distributed systems not only for file transfers but also for searchable data network. In this thesis we study the methods of sharing geographically distributed data with P2P networks. By now, there are a number of P2P protocols proposed, but most of them are based on distributed hashing tables, such as CAN and Chord which support exact match only or have limited number of search predicate support. When we consider a set of P2P nodes as a massively distributed database, several types of search predicates should be provided in addition to exact match search. So in order to efficiently support spatial data sharing P2P application, we need to design new P2P protocols that broaden the types of query processing, improve the performance and fault-tolerance.

Peer-to-peer systems is a new computer network paradigm as described above. It is believed that the peer-to-peer system is not only targeted on sharing music or video file, but also may work in many area like communication, computing, database, etc. As in Figure 1.1(b), we imagine that many distributed data sources are connected together using peer-to-peer network. The computing device, which holds a piece of spatial data source, is a peer node in the systems. Peer nodes can join, leave on their own agenda or sometimes fail suddenly due to power loss etc. In this thesis, we explore the possibility of sharing spatial data with peer-to-peer systems. We try to answer the problems such as: what the structure should be; what queries can be supported; how to implement the robustness, scalability and fault-tolerance under dynamic network environment.



(a) Geographically distributed local data sources

(b) Self-organized into a P2P overlay networks

Figure 1.1: Peer-to-Peer Networks for spatial data sharing

1.2 Goals and Contribution

After carefully investigating related projects and prior peer-to-peer networks, the necessity of designing new peer-to-peer networks is identified. Existing systems are mostly built on DHT-based p2p systems and therefore the query support is limited due to the underlying implementation. The basic requirements of peer-to-peer systems for spatial data sharing we identified are:

Spatial Query Support. Without any centralized indexing service, peers in the system should collaboratively support not only exact query as in other systems, but also range query and nearest neighbor query efficiently. The rich set of query predicates give a chance and stimulate to build more flexible and powerful applications over it. Two query examples are: "find 3 closest available ATM corners **near** location P.", "find the average temperature and dioxide concentration **within** region R.".

Easy Maintenance. In this thesis, we assume the pure type of peer-to-peer system. In dynamic network environment, the peer nodes may join or leave the network frequently. The system must be able maintain its structure by peers themselves with less human intervention. The system must be able to restore to normal state when suffering errors.

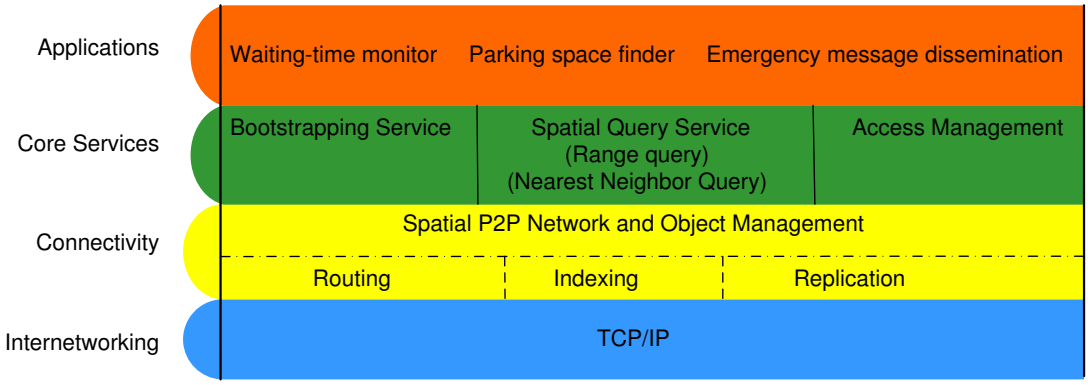


Figure 1.2: The hierarchy of peer-to-peer networks for spatial data sharing

Robustness and Fault Tolerance. The system must be robust to network *churn*. The global consistency should not be a requirement for uninterrupted service provision. The system must be available under massive failures. The system must be able to recover from massive failures without interrupt of service provision.

As in the hierarchy illustrated in Figure 1.2, my research focuses on the peer-to-peer structures, which include routing and indexing in connectivity layer, and spatial query support in core services layer.

This thesis first presents the design and evaluation of GNet, an early work of exploring the possibility of geographical peer-to-peer protocol that targets supporting wide area location-based service. The GNet protocol uses hierarchical geographic address as the identifiers of peer nodes. By combining domain-progressive routing mechanism like plaxton mesh with geographical domain hierarchy, this protocol has the advantages of efficient routing, locality preserving, etc. It supports position-based and especially geographically scoped operations efficiently. Though implementation prerequisites limits its application area, analysis and evaluation results demonstrate its scalability, query efficiency and load balancing features, which makes it adaptable to certain applications.

As the main contribution of this thesis, DHR-Trees protocol is then presented with its design, multidimensional query method and performance, maintenance method and cost analysis, structure fault-tolerance and query fault-tolerance under churn. DHR-Trees structure is the first peer-to-peer structure that supports region-based multidimensional

search predicates, such as range queries and nearest neighbor queries as in R-Trees structure, the predominant indexing structure in spatial databases.

The thesis presents the structure of DHR-Trees and multidimensional query support mechanism. Instead maintaining a global centralized R-Trees index, each peer owns a semi-independent partial region tree structure, which makes it possible to keep correctness of structure even under dynamic network changes. Each geographically distributed peer node is identified by its value on Hilbert space filling curve, which maps Euclidean space into one-dimensional identifier space. Peer nodes self-organize into a virtual ring, sorted by the value of the identifier. For routing purpose and query supporting purpose, each peer maintains a routing table which contains pointers to a number of nodes in the network and region information of underlying entries in the DHR-Trees. The routing cost is logarithmical to network size. The spatial query evaluation results show that queries can be efficiently supported with less traffic than its competitor, the Squid P2P protocol. Furthermore, the nearest neighbor query, which is not supported in Squid, can also be efficiently executed.

The thesis then provides analysis of DHR-Trees peer-to-peer systems under network churn, including scalability, robustness and resilience. To keep the structure correct as nodes join, leave, and fail, peer node in the network periodically run two stabilization processes, i.e. ring stabilization and routing table stabilization. Our analysis and evaluation result shows that the overhead of updating routing tables when a new node joins increases logarithmically to the network size. This demonstrates the scalability of DHR-Trees. By introducing successor lists to the entries in the routing table, robustness and resilience are greatly improved. Moreover, to eliminate the frequent updating requirements of the region information in harsh churn environment, we introduce the usage of adaptive bounding rectangle as the replacement of minimum bounding rectangle. This approach decreases the updating overhead and greatly improves the quality of query result under churn.

Through this thesis, two new novel peer-to-peer protocols are provided. Both GNet and DHR-Trees are designed to be architectures for sharing geographically distributed spatial data. In particular, the DHR-Trees can not only index spatial data as in centralized R-Trees, but also be able to handle dynamism in the peer-to-peer network. We believe our approaches can help realization of certain distributed spatial data sharing applications. We hope our works will stimulate more research interest in both peer-to-peer structures

and spatial data sharing applications.

Chapter 2

Background

2.1 Peer-to-Peer Systems

Peer-To-Peer networks are distributed systems in nature, without any hierarchical organization or centralized control. Peer-to-peer overlay systems go beyond services offered by client-server systems by having symmetry in roles where a client may also be a server. It allows access to its resources by other systems and supports resource sharing, which requires fault-tolerance, self-organization and massive scalability properties.

In a broad sense, P2P overlay network can be viewed as a kind of communication framework, which specifies a fully-distributed, cooperative network design with peers building a self-organizing system.

2.1.1 Characteristics

There are two distinct characteristics of peer-to-peer architectures than traditional systems are as follows[6]:

- The sharing of computer resources is by direct exchange, without requiring the intermediation of a centralized server. Centralized servers can sometimes be used for specific tasks (system bootstrapping, adding new nodes to the network, obtain global keys for data encryption). As the nodes of a peer-to-peer network cannot rely on a central server coordinating the exchange of content and the operation of the entire network, they are required to actively participate by independently performing tasks such as searching for other nodes, locating or caching content, routing information and messages, connecting to or disconnecting from other neighboring

nodes, encrypting, introducing, retrieving, decrypting and verifying content, as well as others.

- Their ability to treat instability and variable connectivity as the norm, automatically adapting to failures in both network connections and computers, as well as to a transient population of nodes. This fault-tolerant, self-organizing capacity suggests the need for an adaptive network topology that will change as nodes enter or leave and network connections fail or recover, in order to maintain its connectivity and performance.

2.1.2 P2P Structures and Applications

By structure, we refer to whether the P2P overlay network is created non-deterministically (ad hoc) as nodes and content are added, or whether its creation is based on specific rules. We categorize peer-to-peer networks as follows, in terms of their structure:

Unstructured. The placement of content (files) is completely unrelated to the overlay topology. In an unstructured network, content typically needs to be located. Searching mechanisms range from brute force methods, such as flooding the network with propagating queries in a breadth-first or depth-first manner until the desired content is located, to more sophisticated and resource-preserving strategies that include the use of random walks and routing indices. The searching mechanisms employed in unstructured networks have obvious implications, particularly in regards to matters of availability, scalability, and persistence. Unstructured systems are generally more appropriate for accommodating highly-transient node populations. Some representative examples of unstructured systems are Napster, Gnutella[1], Kazaa[2], as well as others.

Structured. These have emerged mainly in an attempt to address the scalability issues that unstructured systems were originally faced with. In structured networks, the overlay topology is tightly controlled and files (or pointers to them) are placed at precisely specified locations. These systems essentially provide a mapping between content (e.g. file identifier) and location (e.g. node address), in the form of a distributed routing table, so that queries can be efficiently routed to the node with the desired content. Structured systems offer a scalable solution for exact-match queries, that is, queries where the exact identifier of the requested data object is known (as compared to keyword queries).

Using exact-match queries as a substrate for keyword queries remains an open research problem for distributed environments. A disadvantage of structured systems is that it is hard to maintain the structure required for efficiently routing messages in the face of a very transient node population, in which nodes are joining and leaving at a high rate. Typical examples of structured systems include Chord[44], CAN[37], PASTRY[40], Tapestry among others.

Category	**Applications**
Communication and Collaboration	Chat/Irc, Aol, Icq, Yahoo, MSN, and Skype
Distributed Computation	Seti@home, genome@home, and others.
Internet Service Support	P2P multicast systems, BitTorrent, and security applications against DoS attacks
Database Systems	PIER, Piazza
Content Distribution	Napster, Gnutella, WinMX and Winny and others.

Table 2.1: The classifications of P2P application

Peer-to-peer architectures have been employed for a variety of different application categories, which are listed in the Table 2.1. Our work for supporting spatial data sharing is considered to be categorized into "Content Distribution".

2.2 The Most Related Works

There emerged many peer-to-peer protocols and its applications. Among them, most of them are designed for content distribution purpose and support limited query predicates only. For spatial data sharing and multi-dimensional query support purpose, we pick up most related works to ours as following:

GeoPeer. GeoPeer[8] uses physical geographic address as node identifier and nodes self-organize a planar Delaunay triangulation. To reach physically distant node, routing will take too many steps in the mesh of nodes when using greedy forwarding as in ad hoc network[29]. To mitigate this problem, three schemes are proposed to build Long Range

Contacts. But the LRC building process tends to be complicated and some problems such as limitation of unbounded number of LRCs and dynamic operations (node join, leave and fail) will be difficult to solve in large-scale system.

P2PR-Trees. Mondal et al. [35] proposed P2PR-Trees, a variant of R-Trees that targets P2P networks. They showed in their simulation study that P2PR-Trees show better scalability , since they do not suffer from the central server bottleneck. However, maintenance of a dynamic R-Tree, the most important issue in P2P systems, was left unaddressed.

ZNet. ZNet[42] partitions data in the multidimensional space in a way as in the generalized quad-tree. ZNet makes use of Skip Graphs[9] as an overlay network which routes in one dimensional space, therefore multidimensional data space is mapped to one dimensional index space, such that it can be mapped to nodes in the network. ZNet supports range queries, however, they only provide probabilistic guarantees on data availability even when the index is fully consistent due to the underlying Skip Graphs. Moveover, the search performance of the ZNet is $O(d \cdot log_d N)$ while search performance of DHR-Trees is $O(log_d N)$.

Squid. Squid's proposed structure is the closest one to our DHR-Trees. In [41], Schmidt C. et al. presents the design of Squid P2P System and its evaluation . The space is first mapped down into an one dimensional space using a Hilbert space filling curve. The one dimensional data is then range partitioned in one dimension and mapped onto Chord[44] overlay network. For load balancing purpose, however, the original Chord protocol requires that data's identifiers are uniformly distributed in one-dimensional space. This restriction make it inappropriate to use location as data identifier directly, because data can not always be guaranteed of being distributed uniformly in the space. It uses recursive query decomposition to execute range queries, which is not efficient comparing with our proposed DHR-Trees. Furthermore, since routing relies on Chord, which is originally designed for equality search only, the query type is also limited. For example, to realize nearest neighbor query is not easy with Squid since there is no spatial information being preserved.

2.3 Related Projects and Prototypes

The IrisNet[19] is an challenging project by Intel Research. It utilizes distributed server-based approach intending to realize wide-area architectures for pervasive sensing and to enable powerful distributed sensing services. In IrisNet, the sensors—including video cameras (Webcams), microphones, and motion detectors—are connected to the internet via low-cost PCs, which are expected to be deployed globally. They proposed a new concept as *worldwide sensor web*, which is envisioned to provide service such as

- Alert services for notifying users when to head to the bus stop or when water conditions have become dangerous
- Waiting-time monitors for reporting on queueing delays at post offices, food courts, and so on
- Parking-space-finder services for directing drivers to available parking spaces near their destinations
- Lost-and-found services for locating lost objects or pets
- Watch-my-child (or watch-my-parent) services for monitoring children playing in the neighborhood (or elderly parents about town)

The researchers of IrisNet analyzed and concluded that such a service should be able to overcome challenges as: planet-wide local data collection and storage, real-time adaptation of collection and processing, all data as a single queriable unit, queries posed anywhere on the Internet, data integrity and privacy, robustness, ease of service authorship. In the IrisNet architecture, the nodes that can provide generic interface to access sensors are modeled as *sensing agents*(SA) and the nodes that implement the distributed database are called *organizing agents*(OA). The OAs are organized hierarchically by geographical locations to manage local sensor data, such hierarchy can be regarded as mapping of location hierarchy. Such organization is designed to easily facilitate XPATH query, since XML content is organized typically as a tree structure and XPATH is a query language which typically uses hierarchical keyword. The OAs are distributed database and assigned to manage some portion data at certain geographical location. The IrisNet project team has built a parking-space finder prototype application, which promises to provide real-time information about available parking-space near the end user location.

The **SenseWeb**[31] project at Microsoft Research aims to address these challenges by providing a common platform and set of tools for data owners to easily publish their data and users to make useful queries over the live data sources. The SenseWeb platform transparently provides mechanisms to archive and index data, to process queries, to aggregate and present results on geo-centric web interfaces such as MSN Virtual Earth, etc. They expect that such a platform will encourage the community to publish more live data on web and users to build useful services on top of it.

In the proposed architecture, there are primarily two main components,

- GeoDB. GeoDB is the portal for registering sensor metadata. It is assumed that typical user queries will be based on sensor types, descriptive keywords, and geographic locations, such as list of all cameras along a route or average temperature reported by all the thermometers inside a geographic region, etc. To efficiently support this type of queries, GeoDB indexes data by using hierarchical triangular mesh (HTM) indexing scheme which is particularly suitable for geographic queries.

- Aggregator. The aggregator mashes up sensor data with maps. It accepts queries from the client and redirect the geographic components of the queries to the GeoDB. After obtaining the metadata of a set of sensors that satisfy a client query, it contacts the sensors (data publishing toolkits) for their real-time data. It then aggregates the data accordingly (e.g., depending on the zoom level of the underlying map shown to the client). By doing so, SenseWeb provides useful summarization of data to the client. The particular aggregation performed by the aggregator depends on sensor types. For example, an average of the temperatures in a neighborhood can be displayed for data collected from thermometers.

Nevertheless, both the IrisNet and the SenseWeb are based on the traditional approach: some servers and some clients. In this thesis, we intend to explore the possibility of the usage of peer-to-peer systems. Since the system structure change a lot, there are many issues to address such as scalability, fault-tolerance etc.

Chapter 3

GNet: A Geographic Address-based P2P System

3.1 Overview of GNet

In this chapter, we present design and evaluation of GNet, a peer-to-peer(P2P) protocol that is well suited to support wide area location-based service. The GNet protocol uses hierarchical geographic address as identifier of Internet-connected nodes. By combining domain-progressive routing mechanism like plaxton mesh with geographical domain hierarchy, this P2P protocol has the advantages of efficient routing, locality preserving and load balancing etc. In particular, it will benefit Internet scale location-aware applications by supporting position-based, proximity-based and especially geographically scoped operations.

3.2 Introduction

In ubiquitous computing environment, to provide context to support context aware service, more and more devices will be embedded into the surrounding physical environment. Such device might include sensor network units, web cameras as well as *invisible computers* in the future. By accessing these devices with physical location, some exciting applications can be expected.

Such location-based application examples include reading and aggregating information collected by sensor nodes in a given region (for security purposes or environmental monitoring), querying for specific resources available in a geographic area (for instance, looking

for a available parking space in a given neighborhood), and disseminating notifications to all nodes in a given region (to send bargain advertisement information from a department store, to multicast warnings about natural disaster such as floods, etc).

We believe that a practical way to connect and access these devices is to employ the Internet. However, in current Internet, data communication is based on network address rather than its physical location so that mechanism of correlation of geographical address and network address becomes necessary. Another more important issue we recognized is about the appropriate architecture for large scale location-based applications. To operate with huge number of device nodes, traditional way such as client/server model which usually works well at campus or enterprise level can not support such wide area large-scale system (to be weak of single node failure by malicious attack, or tends to be overloaded if many resource-hungry real time data processing tasks are simultaneously imposed upon a single node). In such a situation, we propose a new P2P architecture (called GNet) to support Internet scale location-based applications. GNet organizes nodes by their geographical location and especially address geographically scoped operations. GNet has advantages such as low network diameter (any node in GNet is always reachable in very limited hops), locality preserving (suitable for proximity application and region scoped operations) and self-organization etc. Our protocol in this chapter is mainly designed for location-based applications, but it can also be extended to other overlay network scenarios with little modification (e.g. distributed file system) in which node identifier is based on domain-based naming scheme and domain-based operations are required.

The rest of this chapter is organized as follows: in section 3.3 and section 3.4, we introduce our GNet protocol and some applications. We show some simulation results in section 3.5 and make conclusion in section 3.6.

3.3 GNet Protocol

In GNet, nodes self-organize into a flat overlay network using their *geographic identifier* that is different from physical address used in GeoPeer[8]. A *geographic identifier* is a geographic address where the node is physically located. It takes form of hierarchical domain as described in section 3.3.1. Each node has an elaborately built routing table where there keeps tracking nodes of some selected domains with their geographic addresses and their network addresses (IP address). A node therefore can make a routing path by

limited overlay hops to any arbitrary node of the whole network in a domain-progressive manner. In section 3.3.2 we will illustrate this in detail.

Our GNet implementation takes the form of library to be used in location-based application. The user application interacts with GNet in several ways. In current implementation, *lookup (id)*, *unicast (id, msg)* and *geocast (domain,msg)* are supported. Function *lookup (id)* resolves IP address of node with specified id. The *unicast (id,msg)* sends a message to node with specified id. The *geocast (domain, msg)* enables application to deliver message to all nodes in region of a specified domain.

Figure 3.3 shows architecture of system using GNet. In this architecture, the Internet becomes just a low-level transport tool. The GNet layer, which is the main contribution we have done in this chapter, acts as a substrate for routing geographically over the Internet. The *service* layer on a node provides service (for instance, region-based data aggregation) to the node itself or other node in the GNet. When *service* layer at a node receives a service request, it will decide, depending on covering region of service request, either to forward request to other node(s) or do service reply. We illustrate two example service in section 3.4. The *service* layer is open for developing new service based on our GNet. User application layer consume service provided by lower service layer by calling *service* layer API.

It must be noticed that our protocol is not designed for content distribution application such as mp3 file sharing, but is to be a substrate for location-based service. Since our GNet is not a general purpose P2P architecture, GNet does not provide facilities such as to save message/content based their hash value like in DHT-based protocols directly. Our architecture is rather suitable to region-based operation which other P2P protocols can hardly provide.

3.3.1 Geographic Domain Name Identifier

GNet assumes that all geographically distributed nodes each have a globally unique *Geographic domain name IDentifier* (GID). The GID takes the form of textual domain-based name in a hierarchical fashion. Mailing address scheme is a straightforward example that has hierarchical elements as state, city, street, and unit etc. For instance, "Tokyo.Meguro.Komaba.chome4.N001" stands for a node N001 located at "chome4, Komaba, Meguro, Tokyo" in Japan. It is obvious that relations between such GIDs can be repre-

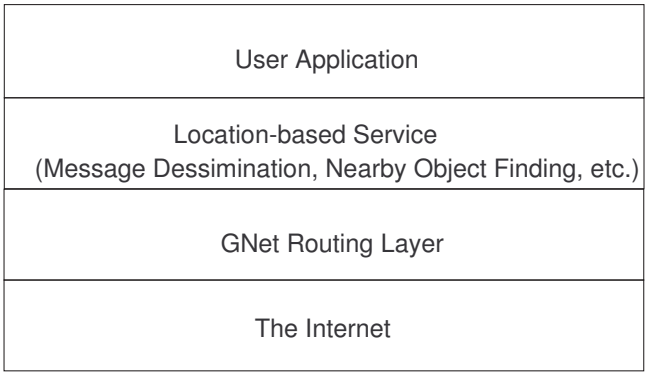


Figure 3.1: GNet system architecture

sented by a abstract *domain tree*. A city domain is a child domain of state, while a street domain is a child domain of city, and so on. It must be noticed that assigned GIDs are all leaf nodes (non-domain nodes) in the domain tree.

Comparing with DHT-based overlay network, we see the distinct difference between them. DHT-based protocol uses digital number, which is of less meaning by simply hashing node name (such as IP address, user name etc.), to identify individual node. Using hash value of node name string of geographical address as node identifier, will inevitably lose hierarchical tree property and make it difficult to preserve geographical relationship between nodes. It will then be difficult to do region-constrained operation to support location-based service. On the contrary, geographic address scheme keeps the inherent domain nature (i.e. domain is actually a physical region) and will preserve physical location relationship between nodes. In practice, the nodes whose GID has same domain prefix can be roughly considered physically adjacent or close to each other from the whole address space view, and nodes which are physically adjacent will share a same domain name with high probability. This characteristic is important to support region-based (actually domain-based) communication. Our work essentially depends on this domain-tree property of geographic address.

3.3.2 Node State and Routing

Each node maintains a *routing table.* Routing table is the core part of GNet. We begin with description of the routing table and then describe routing process. Process to build routing table will be discussed in *node joins* of section 3.3.4.

The routing table is composed of a number of pair entries of GID and IP address. In P2P network, peer does not have global knowledge of the whole network. Due to constantly changing network feature, maintaining information of huge number of all GID and their IP addresses on every individual node is impractical, so we designed a routing table that each node selectively holds information about a number of nodes of whole network. We describe routing table of any node N with GID $x_1.x_2...x_h$ as following:

- Node N maintains a routing table.
- A routing table has h ROWs. Row at level i is denoted by ROW_i.
- ROW_i ($i \in [1, h]$) contains a self entry N for self domain $x_1.x_2...x_{i-1}.x_i$ and d-1 entries of ***domain contact node*** (DCN) for neighbor domains of $x_1.x_2...x_{i-1}.x_i$. The d is the number of child domains of upper domain $x_1.x_2...x_{i-1}$ at domain level i. Each DCN represents neighbor domain of $x_1.x_2...x_{i-1}.x_i$ at same domain level. A DCN entry contains a pair of GID and its IP address of one of potentially many nodes that is located within the neighbor domain. DCN behaves like an entrance point to the domain it represents and is used when node N want to refer to the domain.
- ROW_i is tagged as $x_1.x_2...x_{i-1}$.This is ***parent domain*** that stands for upper domain of all node entries in ROW_i.

Note:

- Domain x_0 is defined as virtual root of domain tree.
- DCN of smallest domain (leaf node in domain tree) $x_1.x_2...x_h$ is node N itself.

Here is a simple example to explain routing table. Suppose a number of nodes are distributed in a geographical area as shown in Figure 3.2, geographic address of all nodes forms a 3-levels hierarchical domain tree. Node a3.b8.c15 then has a routing table shown in Table 3.1. First row of the table contains three domains a1, a2 and a3 and their respective DCNs. These DCN entries each represents domain it belongs to. In this example, node a1.b2.c5 is DCN of domain a1 and stands for domain a1, a2.b4.c8 is DCN of domain a2 and stands for domain a2 and a3.b8.c15 (self node) is DCN of domain a3 and stands for

domain a3. In the second row, domain a3.b8 and sibling domains have their DCNs being contained: a3.b6.c11 for a3.b6, a3.b7.c14 for a3.b7 and a3.b8.c15 (self node) for a3.b8. In the 3rd row, domain a3.b8.c15 is indeed node a3.b8.c15 and siblings are a3.b8.c16 and a3.b8.c17. Here it should be noticed that DCN of a domain was randomly selected during node-join process and could be safely replaced by other node of the same domain when this DCN leaves or fails subsequently.

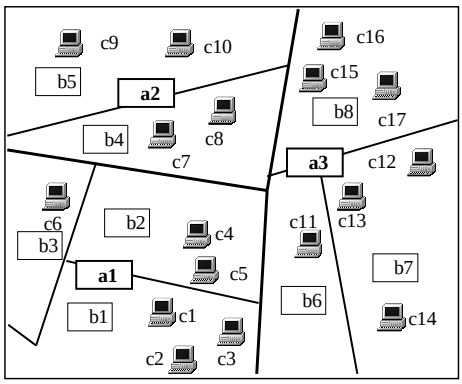


Figure 3.2: A example of three-levels hierarchical address

Parent Domain	**Sub Domain** and contact node		
(root)	**a1**.b2.c5	**a2**.b4.c8	**a3**.b8.c15
a3	**a3.b6.**c11	**a3.b7**.c14	**a3.b8.**c15
a3.b8	**a3.b8.c15**	**a3.b8.c16**	**a3.b8.c17**

Table 3.1: Routing table of node a3.b8.c15

Now we describe routing process. First let us see a routing example and then conclude the process using pseudo code. Suppose source node N_s with GID $a_1.a_2....a_h$ sends message to destination node N_d with GID $b_1.b_2...b_l$. Because N_d has top domain as b_1, source node N_s first look up its own routing table and found out DCN of domain b_1, say node N_2, then N_s forward message to N_2. N_2 is a DCN of domain b_1 so that N_2's GID must begin with b_1. And in addition, in N_2's routing table, ROW_2 contains DCNs of all child domain of b_1, including domain $b_1.b_2$. Suppose DCN of domain $b_1.b_2$ is N_3, at next step message is forwarded to N_3 that represents smaller (lower) domain $b_1.b_2$. After N_3 receive routing message, in the same way, it will find smaller domain towards the destination node N_d. We

conclude that the routing will finally converge at the destination node in this progressive manner. The overlay hops to reach the destination will be less than or equal to l, where l is the node depth in the view of domain tree. We describe this routing process using pseudo code shown in algorithm 1.

Algorithm 1: n.route (*dest_gid*, *msg*)

```
// Node n route message msg to node of dest_gid
Input: dest_gid destination gid
Input: msg message to route
1 begin
2     if n = dest_gid then
3         n get msg;
4     else
5         n' = n.smallest_domain_dcn(dest_gid);
6     end
7 end
```

Node degree, which means the number that a node has to keep record of other nodes, is also an important factor that needs to be considered. GNet has a reasonable node degree. This is referred to the number of DCNs in our routing table. From the definition of routing table, the degree of node N is:

$$D = \sum_{i=1}^{h} (d_i - 1) \tag{3.1}$$

Suppose having a balanced domain tree where h=8 and $d_i = 10$ ($i \in [1 \ldots \mathrm{h}]$) with total number of nodes 10^8, total number of entries in the routing table is up to only 72.

3.3.3 Geographically Scoped Routing

Geographically scoped routing is implemented in routine $geocast(region, msg)$. This routine disseminate a message to a given region. Because in GNet the *lookup(id)* and *unicast (id,msg)* is implemented similarly as *lookup(id)* , we do not explain these two operations in detail.

As mentioned in section 3.2, geographically scoped operation is one of important

Algorithm 2: n.smallest_domain_dcn(dest_gid)

```
// Node n search the local routing table for the
// DCN of smallest (lowest) domain to which dest_gid belongs
Input: dest_gid destination gid
Input: msg message to route
1 begin
2     parent_domain = commonDomain(n.GID, dest_gid);
3     row = getRowfromRoutingTable(parent_domain);
4     next_sub_domain = nextSmallerDomain(dest_gid, parent_domain);
5     n' = row.getDCN(next_sub_domain);
6     return n';
7 end
```

location-based service. We recognize it as a fundamental operation that needs to be supported at the GNet layer. In current scheme in this chapter, we only provide symbolic domain based routing. By mapping physical geographic address space to logical geographic address space, we are able to adapt GNet to physical address space as well.

Any GNet node has the ability to initiate a geographically scoped routing. This operation routes a message to all nodes in a specified domain. We define it using pseudo code in algorithm 3 and some notation is same as defined before algorithm 1:

It should be noticed that the destination domain can be an arbitrary region at any domain level. The region is either a single geographic domain or a compound domain. Given a compound domain being composed of some smaller domains, GNet is able to route message to the given *compound domain* simply by decomposing it to smaller domains and call above routine separately and recursively.

3.3.4 Dynamic Operations

Although nodes embedded in the environment tend to be stationary and may not join or leave frequently, as a autonomous peer-to-peer system, in practice, GNet needs to deal with node that join or leave the system voluntarily or fail occasionally. This section describes how GNet handles these dynamic situations.

Algorithm 3: n.geocast (*dest_domain*, *msg*)

```
// Node n route a message msg to dest_domain
Input: dest_domain destination domain
Input: msg message to route
1  begin
2      if n.GID is within dest_domain then
3          if dest_domain is equal to n then
4              n get msg;
5          for all direct child_domain of dest_domain do
6              n' = DCN of direct child_domain;
7              n'.geocast(child_domain, msg);
8          end
9      else
10         n' = n.smallest_domain_dcn(dest_domain);
11         n'.geocast(dest_domain, msg);
12     end
13 end
```

Node Joins. When a new node X is about to join, it needs to inform other nodes of its presence, and then initialize its routing table with DCNs of some domains. We assume the new node knows initially about a GNet node A that is already part of the system. Such a node can be located using *expanding ring* IP multicast, or be obtained by the system administrator through outside channels.

Let us assume the new node is X. Node X then ask node A to route a special *join* message with the GID and IP of X. GNet finally route this message to the existing node Z that is located in same or closest domain with X. The routing method is same as described in section 3.3.2.

In response to receiving *join* request, node Z then reply a message that contains Z's route table. Because X is located within same (or closest) domain as Z is, X's routing table will be composed of same (or similar entries) as Z's routing table, node X can copy most entries from Z and use it directly with little alternation if X and Z are located

Algorithm 4: n.smallest_domain_dcn(dest_gid)

```
// Search the local routing table for the DCN of
// smallest (closest) domain to dest_domain
Input: dest_gid destination gid
Input: msg message to route
1 begin
2     parent_domain = commonDomain(n.GID, dest_gid);
3     row = getRowfromRoutingTable(parent_domain);
4     next_sub_domain = nextSmallerDomain(dest_gid, parent_domain);
5     n' = row.getDCN(next_sub_domain);
6     return n';
7 end
```

within same domain.

However, subsequently copying may lead to "hot spot" problem, because all sequent joining node that belongs to same domain as Z does will all hold same DCN for same domains. To avoid this, after copying routing table, X then asks each DCN in the routing table to provide alternative DCN of domain it represents. Receiving the request, the DCN either select an appropriate node for the desired domain from its routing table, or if necessary use *geocast* routine to find out a node, then sends reply message to X with the newly selected node GID and IP address.

Node Leaves. When a node X leaves, it needs to inform nodes of its departure to those who have entry of X in their routing tables. This will give a chance to these holder nodes to replace X with alternative node as new DCN. Doing this is important to these holder nodes to keep correctness of routing table and to prevent from wrong routing paths to domain that X was representing.

For example, in Figure 3.2, a3.b6.c11 is DCN of domain a3.b6 in routing table of node a3.b8.c15. When a3.b6.c11 leaves, node a3.b8.c15 will lose domain contact to domain a3.b6, so a3.b8.c15 must has a3.b6.c11 entry replaced with new DCN to keep correctness of routing table.

Broadcasting to all top domains to inform all network nodes of X's departure generate

unnecessary excessive communication overhead and is apparently inappropriate. Since only a small number of nodes hold entry of X, GNet chooses to let X maintain a *holder list*. The *holder list* contains all nodes which hold entry of X. The *holder list* is updated when other node became holder of X.When leaves, X just send message to nodes in *holder list* to let holder nodes to replace X with other alternative DCN. We must notice that this *holder list* does not increase node degree because it needs not to be always correct, i.e. it is unnecessary to communicate with holder node to verify its aliveness.

Furthermore, GNet can optionally let X provide holder with one of its neighbors as alternative DCN when leaves. This can reduce traffic when a holder needs to find new alternative DCN.

Node Fails. The correctness of the GNet protocol relies on the fact that each node knows DCN of neighbor domains at each domain level. However, this invariant can be compromised if DCN nodes fail. Unlike node leave, sudden failure of node will make holder nodes unable to route into the domain where failed node was acting as DCN. For example refering to Figure 3.2 and Table 3.1, when a3.b6.c11 fails, one of its holder node a3.b8.c15 consequently loses domain contact of domain a3.b6. The routing table unfortunately lost its correctness temporarily until new substitute node found proactively. To handle this, each node has two modes (proactive mode and reactive mode) for checking routing table correctness. In proactive mode, node checks DCNs' aliveness periodically. In reactive mode, node does not perceive DCN's failure until trying to route message or execute service request to the DCN's domain. Whenever a DCN is no longer alive, a node has to try to find out an alternative to substitute it. This is done as follows: Node X sends a request to any valid node R in its routing table to search alternative DCN with domain information. On receiving request, node R initiates a search mission using geocast as described in previous section 3.3.3. Nodes at last receive such search message will reply back to the node X with its GID and IP address. X in turn, randomly choose one of them as new DCN.

To increase robustness, for each DCN entry, GNet node optionally maintains a backup DCN list of size r. If a DCN does not respond, the node can try the second entry in its backup list, and so on. Only all backup list nodes' simultaneous failure can result failure of routing into that domain.

3.3.5 Characteristics

From the definition of routing table, routing process and discussion on dynamic problem, our proposed GNet can be expected to have following features. We must be aware that GNet requires geographic *address tree* to be nearly balanced for better performance.

1. **Locality Preserving**: GNet has good location proximity property. The node N knows more about nodes within same domain and close neighbor domains than about nodes far away. This important feature contributes to shorter routing path in location-proximity application effectively. For instance, if N_s is a node under domain $a_1.a_2...a_c$ with GID $a_1.a_2...a_ca_{c+1}...a_h$ and N_d is a also node under domain $a_1.a_2...a_c$ with GID $a_1.a_2...a_cb_{c+1}...b_l$, then the length of routing path would be $l - c$, shorter than l.

2. **Load Balancing**: In GNet, load includes routing load and service execution load. When node joins, it will have equal probability to be a DCN as other nodes under same domain at any level. Hence, each node will act as a "router" with same chance and routing load is shared by these DCN for same domain.

   More than route load balancing, location-based service built on GNet is able to take advantage of load balancing as well. Because DCN that represents a certain domain can act as service provider on behalf of the domain as well, any service request for the domain will be executed on anyone of DCNs of the domain. Since each node within the same domain has the equal probability to be a DCN, service execution load is expected to distributed evenly through these DCNs. Node in GNet not only behaves as a consumer, but also as a router, a data aggregator or a database server etc. In this way, service execution load are well geographically balanced among nodes.

3. **Efficient Routing**: In particular, routing path in GNet is relatively short. Since our routing process is *domain-progressive* and every routing hop will make at least one level lower to destination node, the routing cost will be less than l_{max} hops, where l_{max} is the maximum number of domain levels of domain tree. DHT categorized P2P protocol, for example well-known Chord, has mean routing path length of $O(\log(n))$. GNet is better in routing cost respect than other P2P protocol, especially in proximity applications. This is very important and meaningful for large-scale net-

work.

4. **Scalability**: The size of routing table (node degree) is very small in comparison with the total network size n when level of tree l is a moderate number as shown in Equation 3.1.

   Given a regular and balanced domain tree, when the n increase as total level l being fixed, which implies increasing of each domain degree d, the average size of routing table will increase too. Suppose we have a regular domain tree with levels = l and size =n, the routing table size is $D = l * (\sqrt[l]{n} - 1)$. Because our GNet is specially designed to support location-aware service, and in real world case geographic address hierarchy is about to have roughly 6-10 levels, so D tends to be a reasonable small number. Assume l is 8 and network size is 10^8, the routing table size D will be 72 and this network degree is reasonably small. While Chord[44] has routing table size of O(log(n)) = O(27) under above assumption, our protocol is comparable in this respect.

5. **Decentralization**: GNet generally requires no server to support. It works at peer-to-peer mode. No node is more important than any other. This can help avoid single-point-failure to improve robustness and make GNet appropriate for loosely-organized applications.

3.4 Location-based Applications

GNet will especially help support location-based applications. Its location aware nature makes it particularly appropriate to support many applications which are based on geographical location and region over Internet.

Some example applications include location-based applications depend upon known host location, proximity-based applications depend upon discovery of host nodes. We now illustrate the benefits of our system by giving two examples of context-aware services that can be implemented on top of GNet. It should be noted that, with the exception of GeoPeer and variation of Squid, no other peer-to-peer system would directly support this service.

3.4.1 Data aggregation

Similar to Parking Space Finder in IrisNet[19], it is used to collect information from nodes located inside a given geographic region. This service can be used for real-time environmental or security monitoring of geographical areas by connection the relevant sensors to the GNet nodes. For a given domain, GNet first find a DCN of the query domain (or multiple DCNs of compound domain) and forward query to this node. This DCN node, in turn, can efficiently send query to all nodes in a given domain, collect all the replies, and send the aggregation result back to the client in a single message. If needed, the DCN can also perform data fusion services (such as computing averages, selecting the lowest or highest values, etc).

3.4.2 Information Dissemination

Application such as geographical advertising consists in disseminating a message to all nodes located inside a given geographic region. This service can be used, for instance, to disseminate alarm or bargain information. The service can be easily implemented by routing a notification to a DCN on behalf of the destination areas, which will, in turn, initiate a *geocast* of the notification, using the technique demonstrated in algorithm 3.

3.5 Evaluation

In this section, we present some experimental results we conducted experiments by using a prototype implementation of GNet. All our experiment software was implemented in Java and based on PlanetSim[27]. All experiments were performed on a Dell Optiplex 370 (3.0GHz Intel Pentium ET) with 2GBytes of main memory, running Windows XP Professional. In our experiments in this chapter, the GNet nodes were configured to run in a single Java VM.

For simplicity, we use a collection of hierarchical addresses that form a regular balanced tree shape, which means all branch in the domain tree have equal length and each domain has almost same number of child domains/nodes. Here we define some notation:

l: number of levels of domain tree

d: degree of a domain in the tree

r: mean routing path length in a network

n: network size

3.5.1 Routing Path Length

In our simulations, our program will begin with building GNet with configuration as the initial phase. Node joins into GNet every several steps and its routing table is to be built as described in *node joins* in section 3.3.2.

To verify effectiveness of GNet, we change l from 3 to 8, the range similar to real-world geographic address length. The reason why choose small l is because that level of geographic hierarchy in practice is small. The degree of each domain d is configured from 4 to 10. The network size vary from 1000-100000 which is appropriately maximum capacity for operation under our experiment hardware limitation.

Unlike DHT-based P2P protocol, in GNet, the routing path length is not directly related to network size, but depends mainly on l. We changed l in experiments and the results in Table. 3.2 shows: 1) r is always less than domain-tree depth. 2) r is roughly proportional to l. Because of locality preserving feature of GNet, the r will become smaller in location-proximity applications where source and destination node are closer.

Tree Depth	3	4	5	6	7	8
Avg. Hops	2.72	3.60	4.28	4.82	5.24	6.11

Table 3.2: Domain tree depth vs. Average number of routing hops

3.5.2 Load balancing

We expect that all nodes should ideally be selected as DCN in order to sharing routing load and service execution load. This can be verified through investigating their routing table. We do this by observing how many times of a node being a DCN entry for domains it is located in. We built a network with $n = 5000$ nodes and $l = 4$. Figure 3.3 shows that times of being DCN t are mostly ranged from 0-40. It implies that most of GNet nodes tends to share the routing load and service load among them from the whole network view. The variance from node to node is due to order they joined the network and randomness of being selected. Some “hot spot” can be elimitated by constraining node degree so that

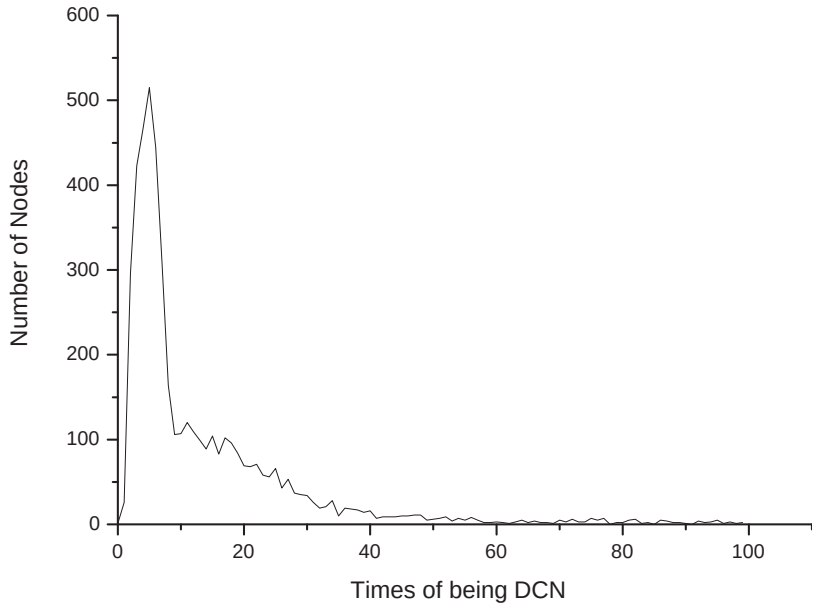


Figure 3.3: The nodes distribution with times of being DCN

network load becomes more balancing.

3.6 Summary

This chapter illustrates our early work on a new peer-to-peer protocol called GNet. GNet protocol is a domain-tree based P2P protocol and has the advantages of short locality preserving and short routing path. It does not directly support general-purpose P2P application such as file sharing, but it is well suited for wide area location-aware service over Internet. It is suitable for provision of services such as region-scoped multicast or region-scoped queries, which can be used by applications to disseminate alarm or advertisement message to a given area, to collect or aggregate sensor network unit data from an interested region. Although there are still some work to do such as performance on mass failures and balance addressing problems, we believe it can be useful in some particular location-based applications.

Chapter 4

DHR-Trees P2P System

4.1 Introduction

Enabling efficient access to distributed and dynamic data is a requirement for handling multidimensional data in P2P systems. Complex queries, such as range queries and k-nearest neighbors queries over multidimensional data, are becoming more important as large amount of data are shared between thousands of peers. For example, a P2P auction network [46], where peers store information on local real estate (geographical location, price, etc) needs to frequently deal with queries such as "find all real-estate advertisement for properties in a given region of a city". Another query example are P2P-based sensor data provision networks for environmental surveillance, which are required to answer requests such as "find the average temperature and dioxide concentration within 100 meters from point A".

Multidimensional indexing and complex queries problems have been extensively studied in the world of databases. The most popular structures in the database are R-Trees[21] and its variants. These tree structures split space with hierarchically nested, and possibly overlapping, minimum bounding rectangles (regions). The queries, started from the root of the tree, use the bounding rectangles to decide whether or not to search inside a child node. In this way, most of the nodes in the tree are never "touched" during a search. Therefore queries can be executed efficiently in most cases (the way of query execution in R-Trees is referred as R-Trees-like query in this chapter). However, it is nontrivial to apply R-Trees in P2P context, since it requires efficient network-based query processing, fault-tolerant network topology and low cost of network maintenance. Without any central

index server, all peers and shared data must be reachable and searchable while the network constantly changes. Much research effort has been done on P2P structures and many P2P structures have been proposed. However, few of them can deal with multidimensional queries efficiently. None of them can support R-Trees-like query in a distributed fashion.

In this chapter, we propose DHR-Trees (Distributed Hilbert R-Trees), a decentralized multidimensional indexing structure for P2P systems, which enables self-organization of peers, supports range queries and nearest neighbor queries efficiently and does network maintenance at low cost. In DHR-Trees, each peer is assumed to control a multidimensional data set enclosed by a rectangle in the space. Intuitively, the peer itself (with data rectangle) can be regarded as a leaf-node of R-Trees. Peer in DHR-Trees has its independent view of R-Trees (in which peer itself can be looked as left-most leaf node). In peer's routing table, peer dynamically acquires and maintains rectangle information on root-to-leaf path of the R-Trees. With these rectangle information, a peer analyzes multidimensional query and decides next peer to forward the query, as making decision of whether or not search inside a child node in the R-Trees. This makes it possible to accommodate existing R-Trees query algorithms with minor modification in P2P systems. We designed the DHR-Trees structure, in which the core part is the peer's routing table. We also designed two major multidimensional query execution algorithms: range query and nearest neighbor query. We have evaluated DHR-Trees system under various settings. Experiments are performed with different network sizes, different types of data distribution etc. The results proved the correctness of P2P systems and effectiveness of range query and nearest neighbor query algorithms. Comparison results also demonstrates that its query execution requires much less node visits than in Squid[41].

The rest of this chapter is organized as follows: In Section 4.3, we introduce our DHR-Tree structure, its dynamic features, and implementation issues. We then show multidimensional query algorithms of range query and nearest neighbor query in Section 5.2. In Section 5.3, simulation results are shown. Finally, conclusion in Section 5.4.

4.2 System Model

In DHR-Trees based p2p system, each peer node is responsible for storing a collection of spatial *data objects* within a spatial *region*. The peer is the controller of the *region* and is identified by assigned *location key*. A *data object* has either point location property or

spatial extent property, such as line, circle, polygon and so on. The *region* is a minimum bounding rectangle enclosing all *data objects*, which are stored at the peer. The *region* may shrink or expand at runtime when *data objects* are inserted in or deleted from the system. The *location key* is a Hilbert value of *location*.

The DHR-Trees protocol has essentially three operations:

1. **Point query.** Given a search point, find node(s) responsible for data located at the point. For example, "Find data object at the location P".

2. **Region query.** Given a search region, find node(s) responsible for data located in the region. For example, "Find data object within the region R".

3. **Nearest neighbor query.** Given a search point, find the nearest k nodes to the point, where k is number of nodes desired. For example, "Find k closest data objects near the location P".

The DHR-Trees system provides a distributed lookup service that allow applications to insert, lookup, and delete data objects using point and region as a handle. We expect the predominant use of DHR-Trees system to be as a lookup service for sharing spatial data among geographically distributed peers.

The API provided by DHR-Trees consists of serveral main functions, shown in Table 4.1. These API functions can be called by any applications built over DHR-Trees system. The first two functions are used when node join and leave a DHR-Trees system. When $lookup(key)$ is called, DHR-Trees efficiently find the corresponding node and return it to the caller. When $RangeQuery(region)$ is called, the system find node(s) responsible for data located in the region and return the collection of them to the caller. When $NNQuery(point, k)$ is called, the system find the nearest nodes to the point, where k is number of nearest neighbors desired.

The DHR-Trees system is implemented as an application-layer overlay network. The underlying layer is usually the TCP/IP network, where node is identified by its Internet Protocol address. Each DHR-Trees node maintains a set of the *data objects*, as well as routing table entries that point to a subset of carefully chosen DHR-Trees nodes. Client application may, but is not constrained to, runs on the same hosts as DHR-Trees nodes that provides storage and query service. This distinction is not important to the DHR-Trees protocol described in this chapter.

Function	Description
join(n)	Causes a node to add itself as a server to the Chord system that node n is part of. Returns success or failure.
leave()	Leave the DHR-Trees system.
lookup(key)	Returns the node associated with the key. The key is a mapped value on Hilbert Space Filling Curve, and the lookup is a exact match without using spatial information.
RangeQuery(region)	Returns nodes that owns region overlapping the parameter region.
NNQuery (point, k)	Returns k closest nodes by distance between node-owned region and search point

Table 4.1: API of DHR-Trees system

Based on the features described above and conditions on the Internet, we set the following design goals for DHR-Trees system.

1. **Scalability.** The system should scale well to potentially billions of keys, stored on hundreds or millions of nodes. This implies that any operations that are substantially larger-than-logarithmic in the number of keys are likely to be impractical. Furthermore, any operations that require contacting (or simply keeping track of) a large number of server nodes are also impractical.

2. **Availability.** Ideally, the lookup service should be able to function despite network partitions and node failures. While guaranteeing correct service across all patterns of network partitions and node failures is difficult, we provide a“ best-effort ”availability guarantee based on access to at least one of r reachable replica nodes.

3. **Load-balanced operation.** If resource usage is evenly distributed among the machines in the system, it becomes easier to provision the service and avoid the problem of high peak load swamping a subset of the servers. Chord takes a step in

this direction by distributing the keys and their values evenly among the machines in the system. More refined load balancing, for example to deal with a single highly popular key by replicating it, can be layered atop the basic system.

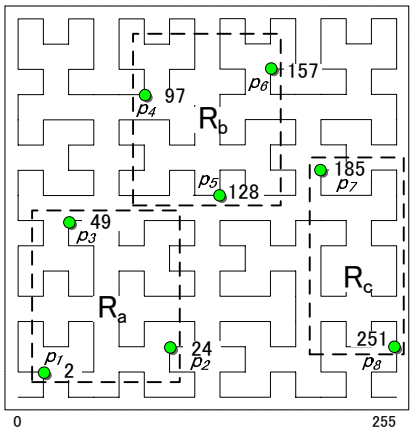
4. **Dynamism.** In a large distributed system, it is the common case that nodes join and leave, and the Chord system needs to handle these situations without any " downtime "in its service or massive reorganization of its key/value bindings to other nodes.

4.3 DHR-Trees P2P structure

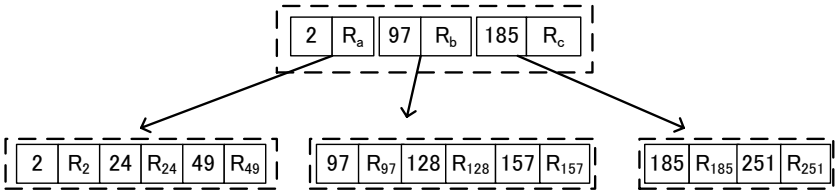
In multidimensional database, R-Tree[21] and its variants use tree structure for storing information. Each node of an R-tree has a variable number of entries (up to some predefined maximum). Each entry within a non-leaf node stores two pieces of data; a way of identifying a child node, and the bounding rectangle of all entries within this child node. Thus, a R-Tree can be regarded as an overlapping region (i.e. rectangle) tree. Search algorithms usually run from root of the tree; use bounding rectangles to decide whether or not to search inside a child node. Therefore, most nodes in the R-Tree are never "touched" during a search. The way of processing leads to high efficiency on query execution.

The Hilbert R-Trees, one of the best-performing multidimensional index structures[17] in R-Trees family, combines the overlapping regions technique of R-Tree with Hilbert space filling curves. It first stores the Hilbert values of the data rectangle centroid in a B+-tree, then enhances each interior B+-tree node by the minimum bounding rectangle of the sub-tree below. This facilitates the insertion and deletion of new objects considerably. Figure 4.1(a) demonstrates two dimensional space with some data objects and Figure 4.1(b) shows its corresponding Hilbert R-Tree.

P-Trees[14], a recently proposed Peer-to-Peer systems, enables one-dimensional range query to be executed collaboratively by peers. Instead of using Distributed Hash Tables method, it organize all nodes into a closed virtual ring by node identifiers (without hashing). As a result, one-dimensional range queries then becomes possible, since the proximity between peers' identifiers is preserved. Each peer independently maintains a routing table, where entries are exactly those on the left-most root-to-leaf path of B+-Tree. Each peer has its own view of B+-Tree, in which the peer's identifier is located at the left-most



(a) Data rectangles in a Hilbert R-Tree



(b) A Hilbert R-Tree

Figure 4.1: Hilbert R-Tree Example

leaf node. To keep freshness and correctness of routing table, peers in P-Trees system periodically call stabilization method (called stabilizeLevel).

DHR-Trees combines advantages of P-Trees and Hilbert R-Trees together. It takes the same topology as in P-Trees. It enhances peer's routing table with bounding rectangle information. It can be loosely regarded as a distributed version of Hilbert R-Trees, but each node has independent view of the Hilbert R-Trees. It can also be regarded as multidimensional extension of P-Trees, enabling indexing and flexibly searching multidimensional information in P2P systems.

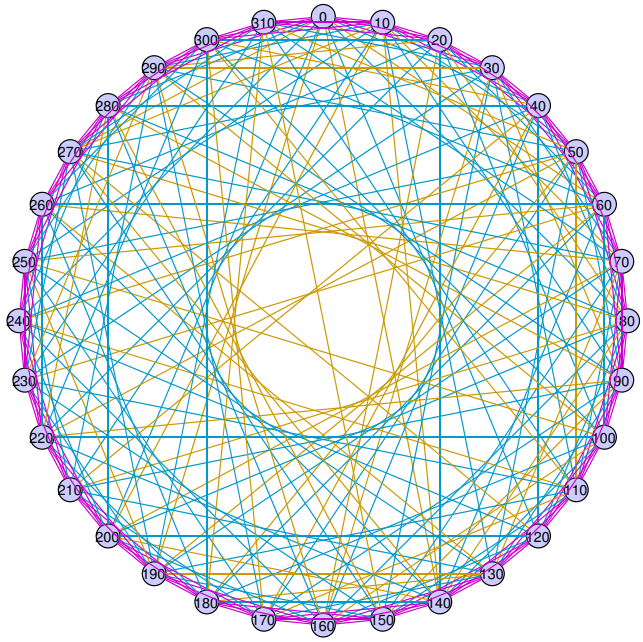


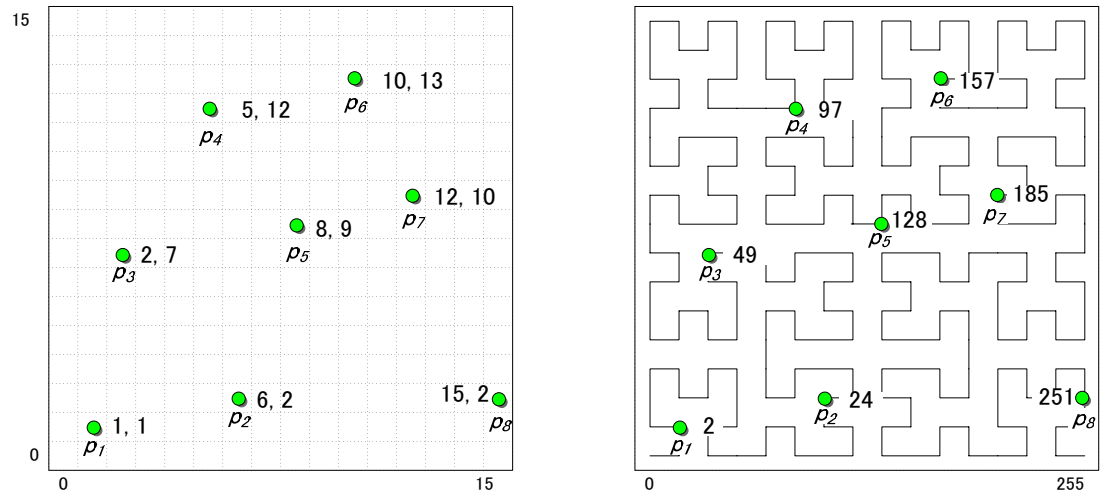
Figure 4.2: Global view of DHR-Trees structure (Network Size $N = 32$)

4.3.1 Overview of DHR-Trees

The idea is partially inspired by and based on the work of P-Trees [14], which is designed for supporting one-dimensional range queries in P2P network. While P-Trees can be viewed as a "distributed b+-tree" for one-dimensional data sharing, our DHR-Trees can be viewed as a "distributed Hilbert R-Trees" for spatial data sharing. The main enhancement is that each peer keeps relevant region tree information in its routing table. Hence it supports the same class of queries as in centralized R-Trees. This is significant since many complex multidimensional query algorithms can be adapted into the P2P network environment.

It gives up the notion of maintaining and sharing a globally consistent R-Tree by all peers, and instead maintains semi-independent DHR-Trees at each peer. This allows for fully distributed index maintenance without any need for inherently centralized and unscalable techniques such as primary copy replication.

In a DHR-Trees P2P system, each peer is assumed to control some data set comprised by a Minimum Bounding Rectangle (MBR) in multidimensional space. Peer p is associated with data rectangle $p.MBR$ and with $p.HCode$, where $p.HCode$ is the Hilbert value of



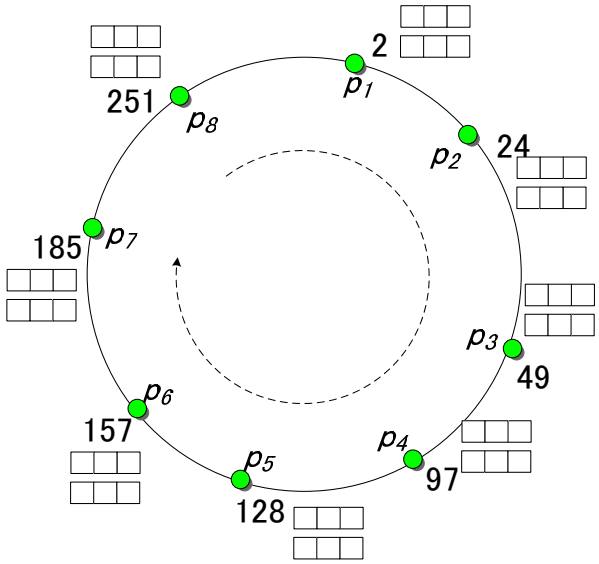
(a) Peers in 2-dimensional space

(b) Mapping by Hilbert Space Filling Curve

Figure 4.3: Overview of DHR-Trees

centroid of *p.MBR* from the space. The *p.HCode* is used as the index value (*key*) of p in the one-dimensional space. At the underlying network layer, each peer identifies itself by *p.peer*, which is usually the peer's IP network address. Peers organize themselves into a virtual ring as in P-Tree by their *p.HCode*. The Hilbert curve therefore becomes a closed ring and peers are indexed by their *p.HCode* in the ring.

Figure 4.3 demonstrates the structure of DHR-Trees for two-dimensional space case. There are eight peers in 16×16 space in Figure 4.3(a). For simplicity, each is assumed to store one point data at its location. As shown in Figure 4.3(b), by mapping from two-dimensional to one-dimensional space, each peer has a unique Hilbert value. These values are used as peer identifiers in DHR-Trees P2P system. Peers organize themselves into a virtual ring by their identifiers. Each peer has a pair of predecessor peer and successor peer as in Chord[44] system. Each peer also has its own composite routing table as in Figure 4.3(c). The composite routing table is the most important component of DHR-Trees. With Hilbert value and coverage information in routing table, DHR-Trees P2P system supports routing and equality search similarly as in P-Trees. By having spatial region information in the composite routing table, DHR-Trees P2P system supports multidimensional query directly and efficiently as in centralized R-Trees.

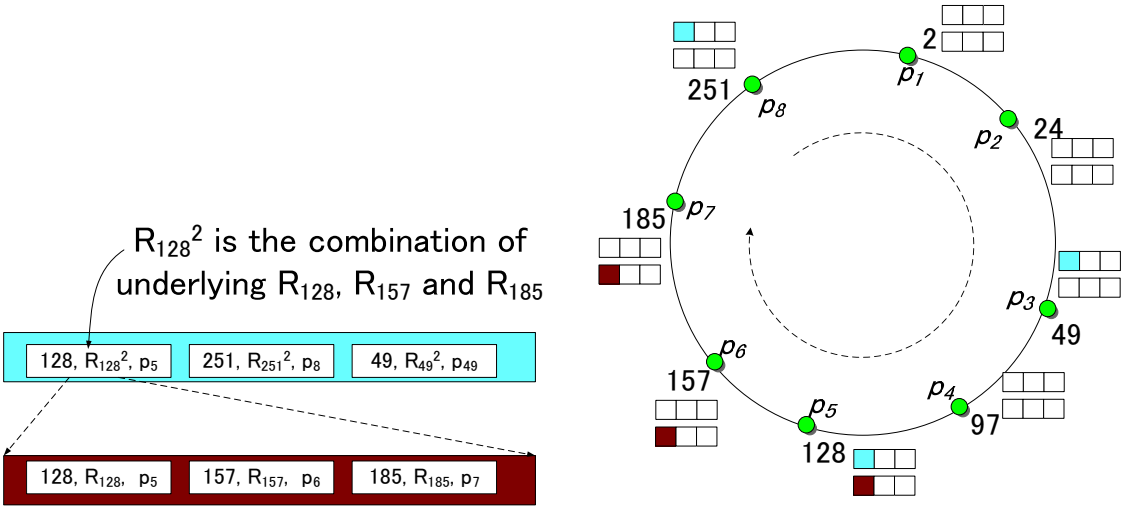


(c) Virtual ring of peers with routing table

Figure 4.3: Overview of DHR-Trees (con't)

4.3.2 Components on a Peer Node

As a member of DHR-Trees, each peer is designed to keep some links to other peer nodes. Basically, each peer has two types of links: Firstly, it should be aware of its preceding and succeeding nodes in the identifier space, so that all nodes can form into a ring structure and are properly ordered and linked together; Secondly, it has a routing table, which enable key lookup and query to be routed to other nodes efficiently. Table 4.2 illustrates primary components on a peer node and their description and their description. The *predecessor* is the previous node on the identifier circle. The *successor* is the next node on the identifier circle. The *nodes* is the Composite Routing Table, which has two significant purpose. Firstly, the *nodes* store some carefully selected neighbor nodes that will speed up the lookup process by identifier. Secondly, by having spatial region information in it, a spatial query can be efficiently executed by comparing with the region information.



(a) Content

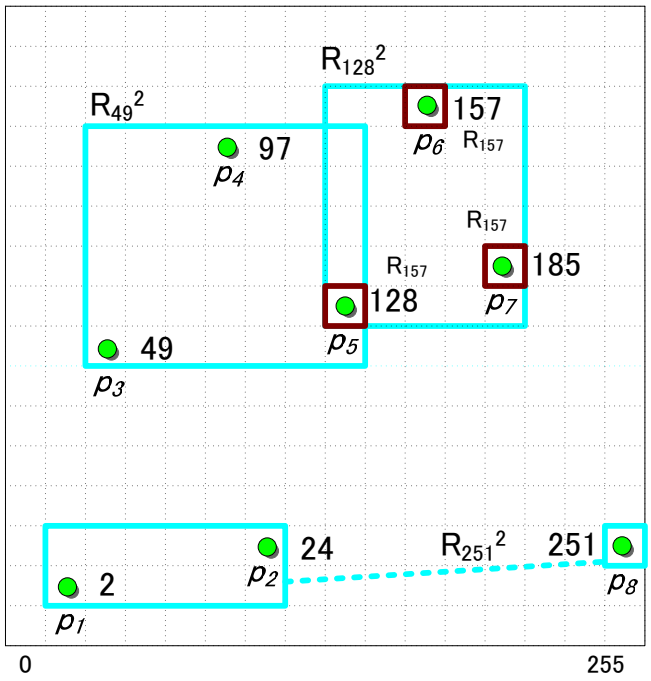
(b) Related peers

Figure 4.4: Routing table of Peer p_5

4.4 Predecessor and Successor

DHR-Trees topology takes the form of ring structure. In DHR-Trees peer-to-peer systems, nodes are identified by the corresponding code on Hilbert space filling curve that mapping Euclidean space to one-dimensional identifier space. Assuming all nodes have unique number as their identifiers, then it is possible to sort all nodes by the identifiers. The ring is formed by connecting the biggest identifier to the smallest identifier existing in the system.

Therefore, each peer, including the peer with smallest identifier and the peer with largest identifer, has only one closest preceding identifier and one closest succeeding identifier. Having correct predecessor and successor for each peer is a sufficient condition for the whole ring structure to be built correctly without disorder, sub-ring or partitioning. For instance in Figure 4.4(b), the peer p_3 has p_2 as predecessor and p_4 as successor, which both are closest to peer p_3 along the DHR-Trees ring. The peer p_8, with which is largest number in the system, has p_7 as its predecessor and p_1 as its successor.



(c) Region information

Figure 4.4: Routing table of Peer p_5 (con't)

4.5 Composite Routing Table

In DHR-Trees systems, each peer stores nodes information of the left-most root-to-leaf path of independent Hilbert R-Trees from its view in its composite routing table. The significant enhancement to P-Tree is that region information MBR (we inherit the word MBR from the database world for convenience) is stored as a part of multidimensional overlapping regions tree. The MBR is a minimum bounding rectangle of the sub-tree below (see example in Figure 4.4(a)). As an important part of DHR-Trees, the MBR entry facilitates multidimensional queries, e.g. range queries could be executed efficiently as in centralized R-Trees. Details of routing table are presented below.

As illustrated in Table 4.2, a peer p maintains a routing table $p.node$, a double indexed array, containing nodes information. Formally, the routing table $p.node$ consists of $numLevels$ rows. There are $p.node[i].numEntries$ of node entries at level i, where $0 < i \leq numLevels$. The $numLevels$ is at most $[log_d N]$, where N is the number of peers being indexed and d is the order of the R-Tree. The $p.node[i].numEntries$ is between d

Component	Definition
nodes	The Composite Routing Table; A double indexed array, containing nodes information
nodes[*i*]	A row at level *i* in the Composite Routing Table. It is also called as a **Routing Level**
nodes[*i*][*j*]	The j_{th} routing entry in the routing level *nodes*[*i*]. It is also called as a **Routing Entry**
successor	The next node on the identifier circle; always equal to nodes[1][1]
predecessor	The previous node on the identifier circle.

Table 4.2: Components on a Peer Node p

and $2d$ at the non-root level of the Tree. At root level, $p.node[i].numEntries$ is between 2 and $2d$. Each entry in the row contains a group of elements as following:

$$\langle HCode, MBR, peer, L, U \rangle$$

which points to the peer *peer* that owns MBR and is identified by the index value $HCode$. In the routing table $p.node$, every *peer*'s MBR at level i is a minimum bounding rectangle that comprises all $peer.node[i-1][j].MBR$, where $1 \leq j \leq peer.node[i-1].numEntries$. The L and U in Table 4.3 is used for maintenance purpose.

Formally, these five elements have meaning and usage as follows:

1. $HCode$. **An Hilbert Code used as identifier** for a peer node. It is a Hilbert value of a given Euclidean point when mapped to a Hilbert Space Filling Curve. The Euclidean point is usually the centroid of local spatial data.

2. MBR. **Minimum Bounding Rectangle**. If the i is 1, the MBR is the minimum spatial extension of local data. If the i is larger than 1, then it is a minimum bounding rectangle which contains all underlying MBRs.

3. *peer*. The **network address** of a peer node with HCode as identifier at the overlay network. It is a IP address when the DHR-Trees peer-to-peer is built on an IP network.

Element	Definition
$HCode$	An Hilbert Code used as identifier for a peer node.
MBR	Minimum Bounding Rectangle.
peer	The network address of a peer node with HCode as identifier at the overlay network.
L	The lower bound for the next routing entry (or minimum *distance* between this and next routing entry).
U	The upper bound for the next routing entry (or the maximum *distance* between this and next routing entry).

Table 4.3: Elements in a routing entry $p.nodes[i][j]$

4. L. The **lower bound for the next routing entry** (or minimum *distance* between this and next routing entry). It is d_{th} HCode at level [i-1] in routing table of *peer*. Formally equal to *peer*.nodes[i-1][d].HCode. If i=1, it is *peer*.HCode itself. Used as lower bound (exclusive) for the p.nodes[i][j+1].$HCode$.

5. U. The **upper bound for the next routing entry** (or the maximum *distance* between this and next routing entry). The successor of the entry that can be reached through right-most path of *peer*.nodes[i-1]. If i=1, it is *peer*.successor. Used as upper bound (inclusive) for the p.nodes[i][j+1].$HCode$

Figure 4.4 shows the composite routing table of peer p_5 and its meaning for intuitive purpose. Figure 4.4(a) presents the composite routing table of peer p_5, where $p_5.numLevels$ is 2. There are three entries at each level where the first one is about p_5 itself. At level 1, there are 3 entries corresponding to the triples (128,R_{128},p_5), (157,R_{157},p_6) and (185,R_{185},p_7). R_{128} is p_5's data MBR. R_{157} and R_{185} are acquired from p_6 and p_7 respectively during p_5 stabilization process on level 1. Notice that p_5,p_6 and p_7 in the triples represent the network address of these peers respectively. At level 2, there are three entries corresponding to the triples (128,R^2_{128},p_5), (251,R^2_{251},p_8) and (49,R^2_{49},p_3). MBR at

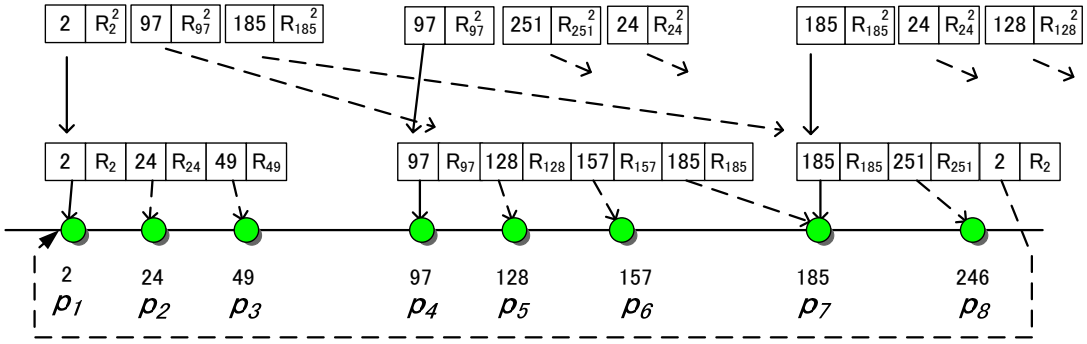


Figure 4.5: An example of DHR-Trees with routing tables at p_2 and related peers

level above 1 is tagged with level i as superscript, such as R^2_{128}, R^2_{251} and R^2_{49}. R^2_{128} is calculated and updated by calling Algorithm 14. The value of R^2_{251} and R^2_{49} are updated at corresponding peers, p_8 and p_3, and in turn these values are obtained by p_5 during p_5 later stabilization process on level 2. Figure 4.4(b) illustrates the distribution of p_5's routing table entries. Level 1 consists of closest succeeding peers. At level 2, entries are succeeding peers with *widened* intervals. The interval is equivalent to the coverage range in P-Tree. In Figure 4.4(c), the region information MBRs that are stored in p_5 routing table is shown. We can see that these rectangles in the Figure can be regarded as the left-most route-to-leaf part of the "sliding" Hilbert R-Trees, which has p_5 as its left-most leaf node. The R^2_{251} is special as it contains two rectangles, one is the rectangle before the end point 255, the other is the rectangle after wrapping point 0. The issue of the wrapped region will be introduced in subsection 4.5.3.

4.5.1 DHR-Trees Routing Table Properties

As shown in previous subsection, each peer p has a composite routing table, the p.nodes. To ensure routing efficiency and query efficiency, a routing table should comply with the following properties. Some properties are similar to P-Trees. Figure 4.5 shows routing table of p_2 and relevant peers.

1. **MBR Containment** This property describes that upper MBR region information enclose all spatial region of the sub-trees. It ensures all lower spatial region and spatial data are indexed by the MBR and no missed objects exist. Formally, for any

entry p.nodes[i][j] with *peer* as p',

$$\begin{aligned} &\forall j' \leq p'.nodes[i-1].numEntries, \\ &p'.nodes[i-1][j'].MBR \in p.nodes[i][j].MBR \end{aligned} \tag{4.1}$$

2. **Fanout** By fanout, we refer to the number of entries per routing entry. All non-root nodes have between d and 2d entries, while the root node has between 2 and 2d entries. Formally,

$$\begin{aligned} &\forall i < p.nodes.\max Level, \\ &p.nodes[i].numEntries \in [d, 2d] \end{aligned} \tag{4.2}$$

   While at the root level,

$$p.nodes[p.maxLevel].numEntries \in [2, 2d]$$

   Allowing the number of entries in a node to vary makes nodes more resilient to insertions and deletions as the invariant will not be violated for every insertion/deletion.

3. **Coverage** This property ensures that all search key values are indeed indexed by the DHR-Tree; i.e., it ensures that no values are "missed" by the index structure. In general, if there are many "missed" values between two adjacent entries, the search performance can degrade due to the long sequential scan along the ring (although the search will eventually succeed). Any"gaps" between adjacent sub-trees imply that search cost for certain queries can no longer be guaranteed to be logarithmic. The coverage property addresses this problem by ensuring that there are no gaps between adjacent sub-trees. A similar issue is ensuring that the subtree rooted at the last entry of each root node wraps all the way around the DHR-Trees ring. These two properties together ensure that all values are reachable using the index.

   Formally, assume $p.nodes[i][j]$ and $p.nodes[i][j+1]$ are two adjacent entries in the routing table, the coverage property of $p.nodes[i][j+1]$ is satisfied iff:

$$p.nodes[i][j+1].HCode \leq p.nodes[i][j].U \tag{4.3}$$

4. **Separation** The separation property ensures that the overlap between adjacent sub-trees is not excessive by ensuring that two adjacent entries at level i have at least d non-overlapping entries at level i-1. Though some overlap is possible and desirable

because the sub-trees can then be independently maintained, excessive overlap can compromise logarithmic search performance.

Formally, the separation property of $p.nodes[i][j+1]$ is satisfied between these two entries iff

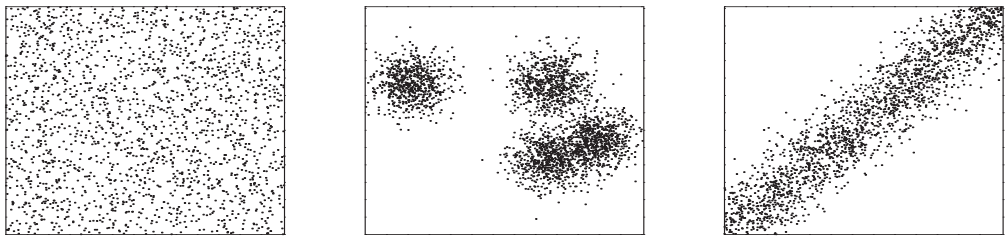
$$p.nodes[i][j].L < p.nodes[i][j+1].HCode \tag{4.4}$$

4.5.2 Mapping between Identifier and Network Address

Peer-to-peer systems is viewed as a kind of overlay network. From the layered structure view, it locates itself on top of existing network which is usually IP network. When run a query or lookup at a node in a structured peer-to-peer network, the node will make a decision of to which nodes to forward the request and perhaps some intermediate result. This is be done by referencing routing table, looking up appropriate node identifier to which may meet the query request. Having made decision on next hop nodes to forward, the node has to employ underlying network transporting mechanism that is usually a function of IP network, where each node is identified by an IP address. Therefore, the mapping between node identifier and network address is prerequisite for the overlay network to work.

In DHR-Trees peer-to-peer network, the mapping is done in the routing table. As shown in Table 4.3 in previous subsection, there is a triple in every routing entries, composed of $HCode$, MBR and *peer*. The $HCode$ is used as the node identifier and *peer* is network address of the peer node in the underlying overlay network. The mapping between $HCode$ and *peer* is one-to-one relationship. We have two assumption for keeping correctness of mapping: one is that the set of all $HCode$ has property of uniqueness, which means there is no duplicate $HCode$ exists; the other assumption is that *peer* has uniqueness in the network address space. The network address is guaranteed by existing network protocols, while the uniqueness of HCode should be guaranteed by implementation of DHR-Trees peer-to-peer system itself.

Since the pair of $HCode$ and *peer* is always updated at the same time, the mapping relation always holds under any circumstances. However, when contacting with the node by the address failed, the routing entry is marked as **stale** entry . This occurs in either of following situations:



(a) Uniform peer distribution

(b) Clustered peer distribution

(c) Skewed peer distribution

Figure 4.6: Three types of distribution(each for (2000 nodes))

- The node intentionally leaved the network.
- The node changed its address and unfortunately failed to notify other relevant nodes.

4.5.3 Wrapping-around problem

Since all peer nodes are indexed in a Hilbert curve ring, a problem about MBR arises: when a peer is near the end of curve, the MBR entries (i.e. *p.node[i][0].MBR*s) at upper level of the routing table will need to cover a group of *peer.MBR* located near the starting of the Hilbert curve, which are distant from *p.MBR* in the m-dimension space. Intuitively, redundant space will be possibly comprised and it will result in increasing search cost. To solve this problem, we introduce an auxiliary MBR, denoted as *AuxMBR*, for containing the wrapped-around rectangles, while MBR still contains non-wrapped ones. Therefore a more general structure of route entry should be

$$\langle HCode, MBR, AuxMBR, peer, L, U \rangle$$

A null value of *AuxMBR* is allowed when no wrapping-around MBR exists. Accordingly, the range query search (Algorithm 5) should be also altered. Experiment in Figure 4.9 shows routing cost decreased when this solution is applied.

4.6 Evaluation

To evaluate the basic features of DHR-Trees, we conducted experiments using a two-dimensional space. This space was mapped by a Hilbert curve of order 16 down into a

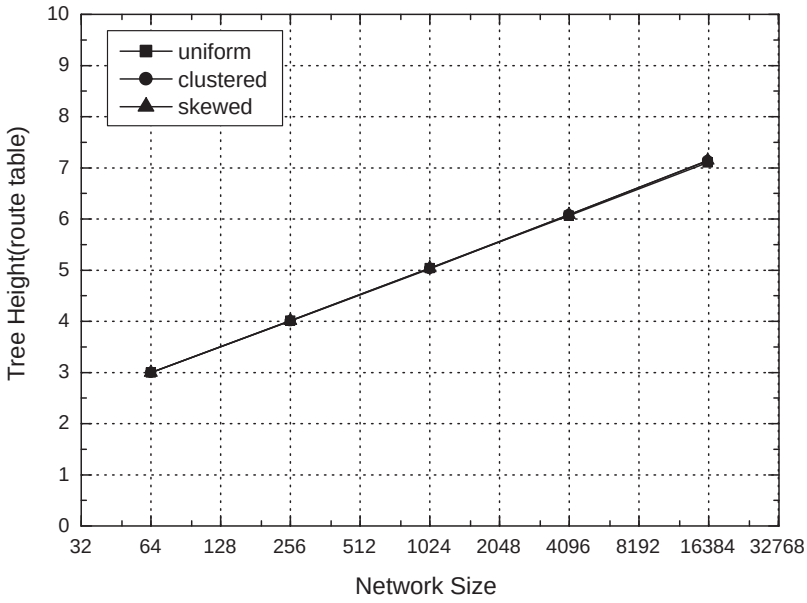


Figure 4.7: Routing Table Height (*nodes.maxLevel*)

one-dimensional space such that it is partitioned into a $2^{16} \times 2^{16}$ grid. For simplicity of evaluation, we assumed that each peer controlled a rectangle of size 1×1. For testing purpose, the number of routing entry at one level was fixed as 4 (the D) and the network size varied from 128 to 16384 nodes. The simulation begins with DHR-Trees building progress as the initial phase. Nodes randomly joins into DHR-Trees every several steps in a random order and the stabilization processes (the stabilization processes will be introduced in Chapter **??**) works periodically, together with the $UpdateMBR$ process. Routing and query experiments were executed after the network becomes stabilized. Three types of distribution of data sets were used: uniform, clustered and skewed distributions as shown in Figure 4.6(a), Figure 4.6(b), Figure 4.6(c).

Routing Table Height. Figure 4.7 illustrates that average routing path height (the *nodes.maxLevel*) are almost the same on the 3 types of distributions. It implies that results show that DHR-Trees were built correctly on 3 different distributions and DHR-Trees was well-balanced in all cases. The routing table height is roughly equal to ($log_D N$). This illustrates the scalability of routing table to the network size.

Routing Path Length. Figure 4.8 illustrates that average routing path length are

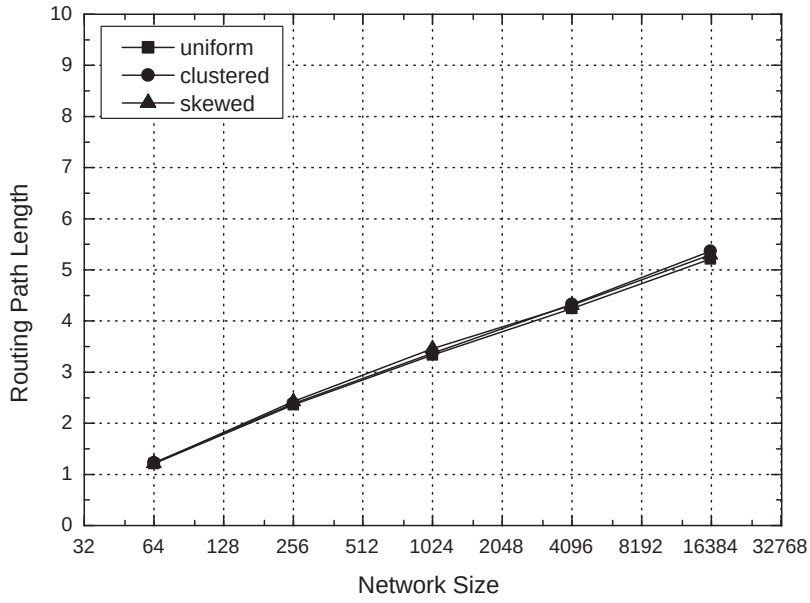


Figure 4.8: Routing Path Length (hops)

almost the same on the 3 types of distributions. The routing path length is less than tree height ($log_d N$) since routing initiated at a peer near the destination peer needs less hops due to proximity.

Improvement by Auxiliary MBR. In Section 4.5.3, the auxiliary MBR is introduced to solve the extensive coverage problem of wrapping-around MBR. Figure 4.9 (uniform distribution) shows that this approach reduced routing cost (number of visited peers) at about $12\% \sim 13\%$ in the case of uniform distribution.

4.7 Summary

In this chapter, we presented the new multidimensional indexing structure called Distributed Hilbert R-Trees. To the best of our knowledge, DHR-Trees structure is the first R-Trees-based P2P structure that can handle dynamism of P2P systems. It utilizes semi-independent R-Tree structure and allows fully distributed index maintenance. Preliminary experimental results under various configuration show that routing cost increase logarithmically as network size increases, which demonstrates routing table scalability and routing path scalability to the network size.

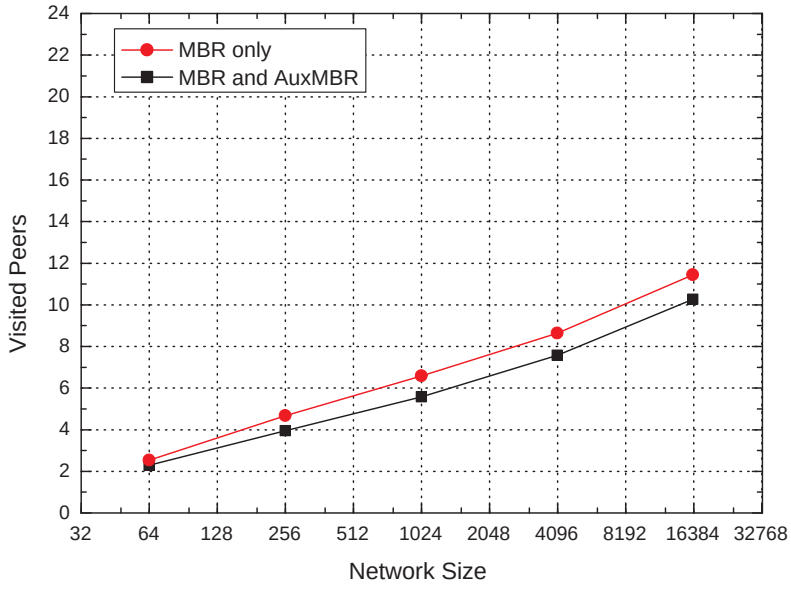


Figure 4.9: Improvement with Auxiliary MBR

Chapter 5

Multidimensional Queries Support in DHR-Trees

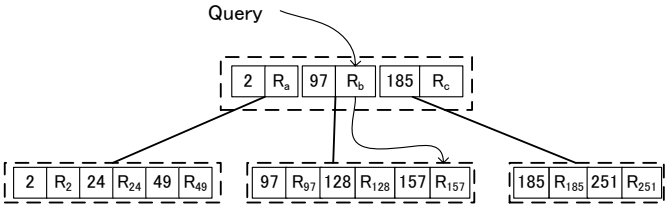
5.1 Introduction

In multidimensional database, the centralized index is exploited to facilitate multidimensional queries. In contrast, in the DHR-Trees peer-to-peer system, the big difference is the absence of centralized index structure. Instead, the index are distributed among peers in the system. Each peer holds only partial region tree in its routing table. A number of other peers and their level-dependent subtree information are also maintained in the routing table as well. These features make the query execution quite different from traditional database.

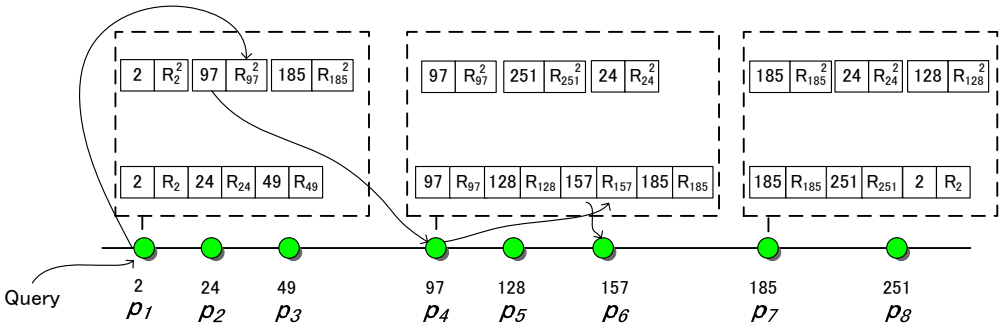
The main difference from centralized database are recognized as follows:

- **Collaborative Query Execution.** In peer-to-peer systems, a peer node usually neither know the whole indexing structure nor holds all data in the system. The execution is often fulfilled by collaborative effort among peers. For example, in an R-Tree database, searching data within a region is a process of descending the R-Tree structure, along with comparing the query window with rectangles stored at the non-leaf nodes. All these operations are done within same memory in the computing device. On the contrary, the query initiated at a peer, will have to be resent by the peer to other peer nodes for fulfilling the query. The query execution can be viewed as a collaborative job among relevant peers.

- **Query Cost Evaluation**. The region information (i.e. the rectangles) stored



(a) R-tree (Descending the R-tree)



(b) DHR-Trees (Collaborative query execution)

Figure 5.1: The difference on query execution

at R-Trees non-leaf nodes, are distributed among peers' routing tables. During a query, any movement among non-leaf nodes becomes an action of visiting peer. In the database community, the storage access overhead is an quantitative criterion for evaluating query cost. In peer-to-peer systems, the network traffic becomes more important than storage access. Consequently, the number of visited peers (or messages sent among peers) becomes an important evaluation standard.

The query execution difference is illustrated in Figure 5.1.

5.2 Multidimensional Queries

5.2.1 Range Queries

In previous chapter, it is shown that the rectangle-based overlapping region trees are kept and maintained in a distributed fashion among peers. This characteristic makes DHR-Trees capable of doing R-Tree-like range queries in P2P systems. Due to the similarity of

point queries and range queries, we focus on how range queries are executed.

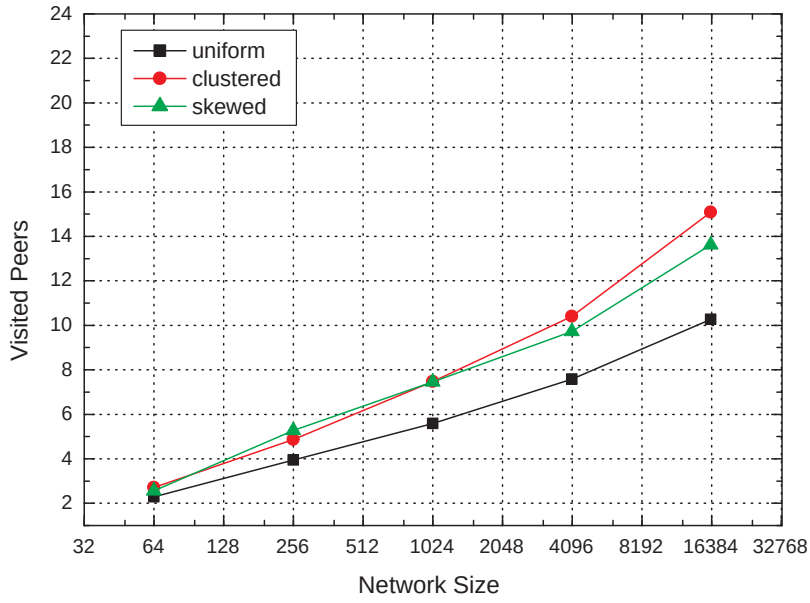
Lemma. *Given that a DHR-Trees is in consistent state, a range query initiated at any peer can finally be answered with all qualified data object that exist in the network with guarantee.*

In DHR-Trees, each peer is ordered by an one-dimensional value - Hilbert code, and all peers are orderly indexed in the underlying ring. When the system reached consistent state, any two neighboring entries at same level in routing table are guaranteed with coverage property, that is, no peer (and data) is "missed" by the index structure. Furthermore, the coverage property also ensures that all peer entries at the lower level are contained by entries at the higher level. Since MBR at the higher level is a combination of all lower *peer.MBR* and is guaranteed with correctness by $UpdateMBR$, that works from the lowest level to highest level, we can prove all MBRs at higher level comprise all lower MBRs by induction bottom-up. As a result, we can assert that any peer's MBR is contained by root level entries on a peer in the P2P system. By virtually descending region tree in the routing table, DHR-Trees can guarantee all qualified data object for a query being returned.

The pseudo-code of range query process is listed in Algorithm 5. The start level l is initialized as highest level of DHR-Trees. Notice that the result, usually being composed of several pieces, is returned in an asynchronous fashion due to different path length and network latency. The peer initiated query has to be responsible for collecting and synthesizing the results.

5.2.2 k-Nearest Neighbors queries

The k-nearest neighbors (kNN) query is described as follows. Given a query point pt, find k nearest neighbors from the multidimensional data. The kNN issue is also researched extensively in database world. The design of kNN query algorithm in DHR-Trees uses similar metric as in work of Roussopoulos et al. [39]. The pseudo-code of kNN query is listed in Algorithm 6. It adapts an ordered depth first traversal into DHR-Trees. At the start, $knnList$, the container of kNN results, is initialized to empty and l is initialized to highest level in the routing table. The $branchList$, the container of sorted searching branch, is built from MBR at the level l and sorted by distance to the point pt incre-



(a) Point queries cost

Figure 5.2: 2-dimensional DHR-Trees

mentally. The *distance* is minimum Euclidian distance applied to a point and a rectangle (same as MINDIST in [39]). This distance is zero if the point is inside the rectangle. The p.kNNQuery is executed iteratively as depth-firth traversal, and terminate when k neighbors are discovered and no more closer branches exist.

5.3 Evaluation

In this section, we present the results of our experiments using DHR-Trees which was implemented in Java and based on PlanetSim[27]. All experiments were performed on a 3.0GHz Intel Pentium ET machine with 2GBytes of main memory, running Windows XP Professional. In our current simulation environment , the DHR-Trees nodes were configured to run in a single Java Virtual Machine.

5.3.1 Two-Dimensional DHR-Trees

The first experiments were conducted using a two-dimensional space. This space was mapped by a Hilbert curve of order 16 down into a one-dimensional space such that it is partitioned into a $2^{16} \times 2^{16}$ grid. For simplicity of evaluation, we assumed that each peer controlled a rectangle of size 1×1. The order d of R-Tree was fixed as 4 and the network size varied from 128 to 16384 nodes. The simulation begins with DHR-Trees building progress as the initial phase. Nodes randomly joins into DHR-Trees every several steps in a random order and the stabilization process of P-Trees works periodically, together with the $UpdateMBR$ process. Insertion/deletion, routing and query experiments were executed after the network becomes stabilized. Three types of distribution of data sets were used: uniform, clustered and skewed distributions as shown in Figure 4.6(a), Figure 4.6(b), Figure 4.6(c).

Point Queries. Point queries cost is measured as the number of visited peers during a query. Using different distributions caused slightly different results as shown in Figure 5.2(a). In the case of skewed distribution and clustered distribution, more peers were visited. This is because dense data sets produce more extensive overlapping MBRs in the routing table, so that query messages are more frequently forwarded to unnecessary peers.

Range Queries. In range query experiments, we used data set in uniform distribution, where network size N varies from $128 \sim 16384$ nodes. Rectangles were used as query windows with side length being adjusted in accordance to the network size, such that the same number of matched peers (MBRs) could be expected to fall into a query window. In our experiments, the number of *matches* was configured as 5 and 10 hits respectively. The location of a query window was randomly selected and peers that initiated a query were also randomly chosen from the network. Query costs were measured by number of visited peers. For each combination of N and *matches*, the queries were executed 1000 times and results were averaged. The results shown in Figure 5.2(b) indicates that range queries can be efficiently executed in DHR-Trees.

Nearest Neighbor Queries. In nearest neighbor queries experiments, a query point was randomly generated. Then a query was initiated at a randomly chosen peer. The depth-first traversal algorithm was then executed among peers. Finally, the desired number of nearest neighbor peers was found and returned to the query peer. Figure 5.2(c)

shows the result of the experiments with configuration for 5,10 and 20 hits, respectively. It demonstrates that the search cost increased slowly as the network size increased.

5.3.2 High Dimensional DHR-Trees

Some experiments with high dimensional data has also been conducted. Space dimensionality m is changed from 2 to 6. Network size is set as 5^m, changing from 25 ($m = 2$) to 15625 ($m = 6$). In each dimensionality configuration, side length of space fixed at 1024 (therefore the space has 1024^m hypercubes) and Hilbert curve of order 10 is used. Multidimensional data are distributed uniformly in the space. Auxiliary MBR is enabled in all experiments.

Point Queries. As shown in Figure 5.3(a), Point queries cost increases as network size increases. The value is close to the result in the 2-dimensional case. This indicates that point queries cost is mainly determined by network size, while dimensionality has less influence on point queries performance. This is because the point queries cost is mainly determined by tree height, which is further determined by total network size when with same order of d.

Range Queries. Three group of range query experiments has been executed. Query window size is set to be 0.05, 0.1 and 0.2 respectively. As for each combination of query window size and dimensionality m, the center of query window is randomly chosen in space and query is executed 1000 times to average results. Figure 5.3(b) shows: With small windows size, query cost increases slowly. With big window size, however, the query cost increases rapidly when dimensionality and network size both increases. This indicates that more peers will have to be involved in query since bigger query window overlaps much more MBRs in multidimensional cases.

Nearest Neighbor Queries. m-dimensional query point is randomly generated. Queries are initiated at randomly picked peers. Nearest neighbors k is configured as 5,10 and 20 respectively. As for each combination of k and dimensionality m, the query is executed 1000 times and number of visited peers is averaged. Figure 5.3(c) indicates the query cost increase slowly with dimensionality and network size increases.

5.3.3 Performance Comparison with Squid

As introduced in Section 2.2, Squid P2P system is the most related one to DHR-Trees as they have the same method of mapping higher dimension space to one-dimensional space and use the similar virtual ring as the overlay network structure. The most different part is that it uses a different routing table (called finger table) and a different query execution mechanism. Squid does cluster refinement to decompose query ranges into smaller query clusters recursively until it reaches *finest* level of the Hilbert curve, since top-down queries are not supported as each peer has no region information in its routing tables; In contrast, DHR-Trees supports R-Trees-like queries directly by having MBR region information maintained in peer's routing table. Hence, DHR-Trees visits less peers and generates less traffic messages when performing range queries, proving it is superior to Squid in efficiency of query execution. Figure 5.4 shows the comparison results of range queries. Moreover, nearest neighbor query is even not supported by Squid system because spatial proximity in multidimensional space is not preserved after mapping into one-dimensional space.

5.4 Summary

In this chapter, we presented two major multidimensional query algorithms with DHR-Trees Peer-to-Peer system. The initial experiment shows that it performs well on multidimensional range query and nearest neighbor queries, which is not supported well in other P2P systems. By comparison with squid, which is closest one to DHR-Trees by using ring structure and reducing dimensionality with Hilbert Space Filling Curve, our experiment results demonstrates that DHR-Trees outperforms squid in query efficiencies.

Algorithm 5: p.WindowQuery (Rectangle w, int l)

Input: w query window

Input: l level of search to start

```
1  begin
2      j = 1;
3      localIntersect = false;
4      if l > 1 then
5          while j <p.node[l].numEntries do
6              if p.node[i][j].MBRintersectsw AND l > 1 then
7                  p' = p.node[i][j].peer;
8                  p'.WindowQuery(w, l − 1);
9                  if j = 1 then
10                     localIntersect = true;
11             j + +;
12         end
13     if l = 1 then
14         while j < p.node[l].numEntries do
15             if p.node[i][j].MBR intersects w then
16                 p' = p.node[i][j].peer;
                   // search in local data set and reply any results to
                      requestor
17                 reply p'.searchLocalData(w);
18             j + +;
19         end
20     if localIntersect then
21         this.WindowQuery(w, l − 1);
22 end
```

Algorithm 6: p.kNNQuery (Point pt, int l, int k, List $knnList$)

Input: pt search point

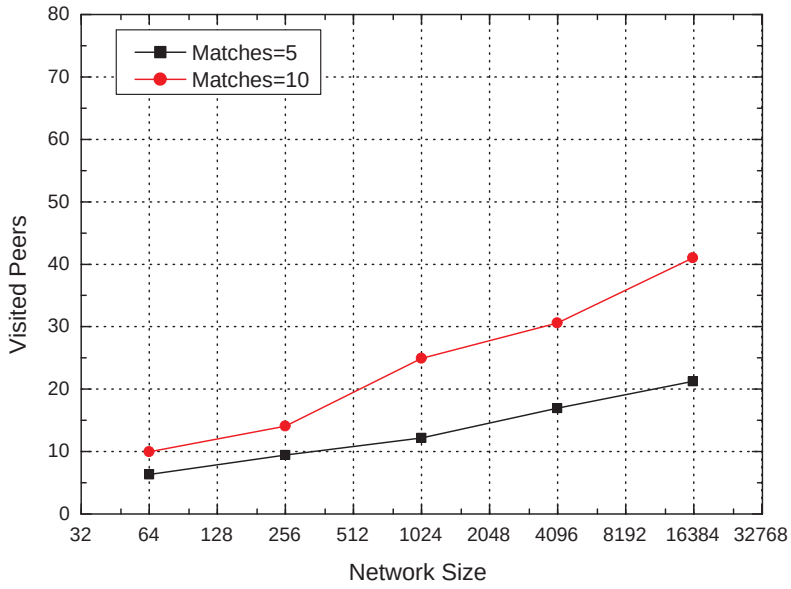
Input: l level of search to start

Input: k desired number of nearest neighbors

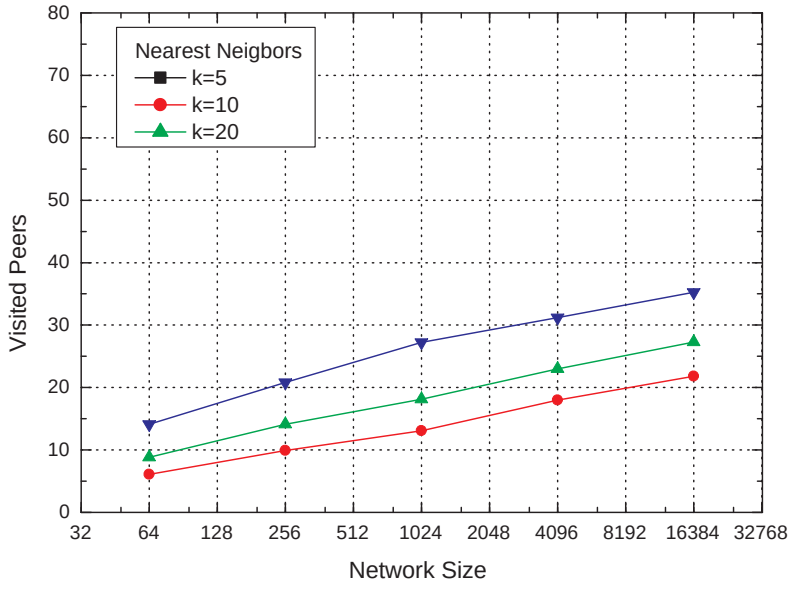
Input: $knnList$ container of k nearest neighbors; empty at start

Output: $knnList$ container of k nearest neighbors result

```
1  begin
2      if l >1 then
           // sort all MBRs at level l by distance to pt and save into
              temporary branchList
3          branchList = p.sortMBRs(l);
4          while j <branchList.size do
               // sort by distance to point pt and prune to size k
5              sort(knnList, pt);
6              prune(knnList, k);
               // distance of furthest MBR in knnList
7              distanceofFurthest = distance(knnList[k], pt);
8              if distance(branchList[j].MBR, pt) < distanceofFurthest then
9                  p' = branchList[j].peer;
10                 p'.kNNQuery(pt, l − 1, k, knnList);
11             j + +;
12         end
13     if l = 1 then
           // add lowest level data to knnList
14         p.addLocalData(knnList);
           // sort by distance to point pt and prune to size k
15         sort(knnList, pt);
16         prune(knnList, k);
17 end
```

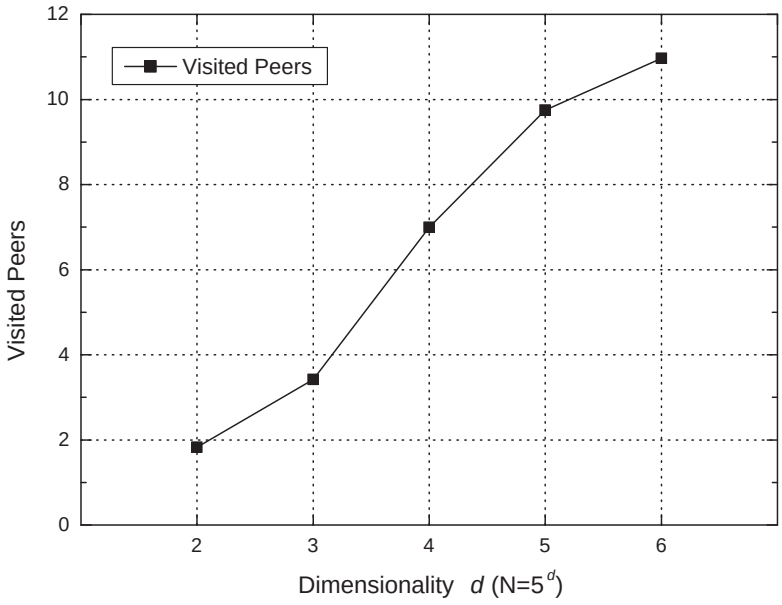


(b) Range queries

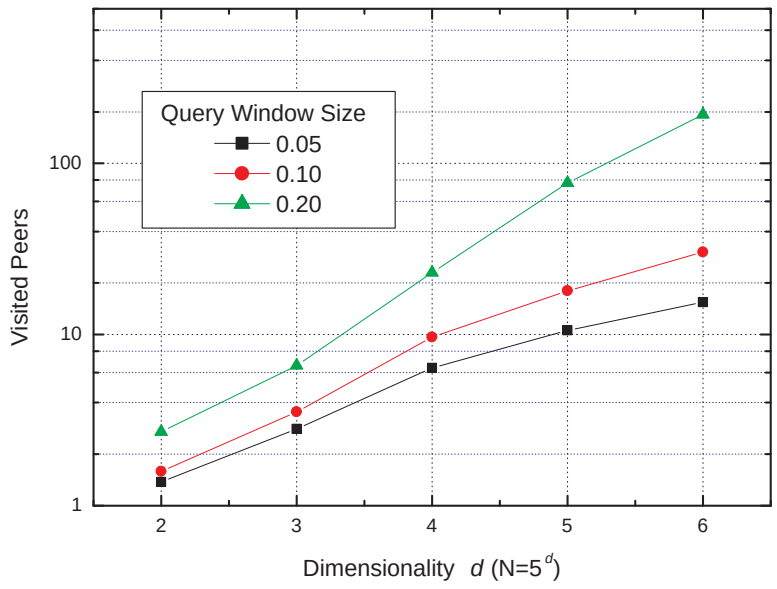


(c) Nearest neighbor queries

Figure 5.2: 2-dimensional DHR-Trees (con't)

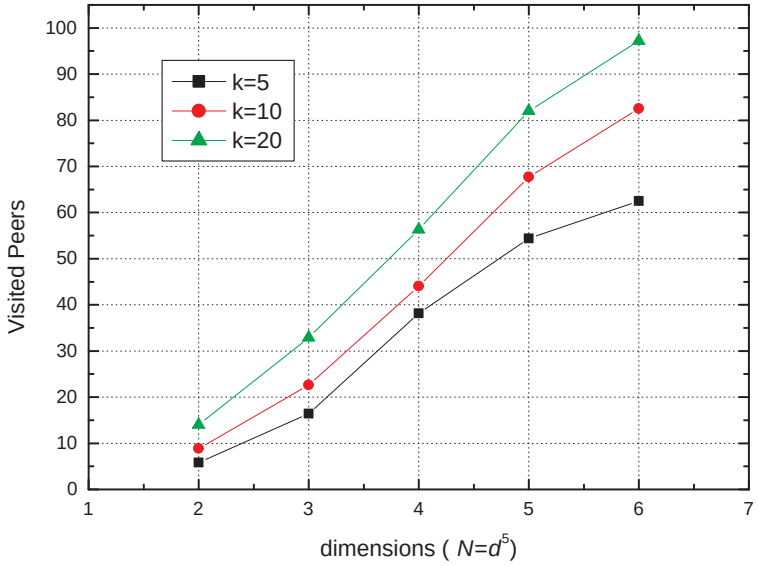


(a) Point queries cost



(b) Range queries

Figure 5.3: Multidimensional DHR-Trees



(c) Nearest neighbor queries

Figure 5.3: Multidimensional DHR-Trees (con't)

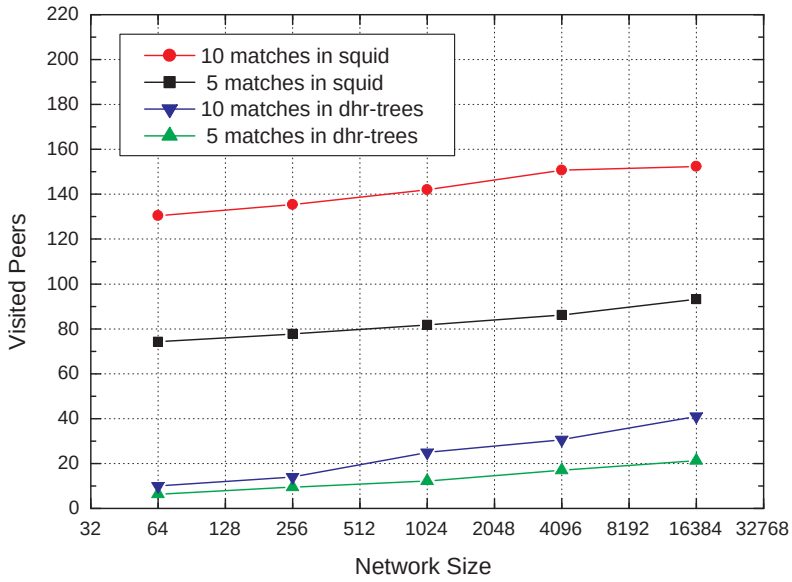


Figure 5.4: Query performance comparison with Squid

Chapter 6

Maintenance in DHR-Trees

A Peer-to-Peer system is an evolving system, since nodes can join/leave/fail at any time. In structured P2P systems, every node must keep correctness of its links to other nodes by making changes according to the node members' changes. In this chapter, the discussion will be made about dynamic operations of joining, leaving and failure. The problem definition is that, assuming the network is in steady state, and suppose a node joins into, leaves from or fails in the system, then what is the changes to make and how many changes will occur in the whole system. We will give algorithms that allow nodes to actively maintain its components. The global view of maintenance processes are illustrated in Figure 6.1. The ring stabilization repairs incorrect predecessor and successor relationship for the ring structure; the ping process check if the entries are alive and if they are consistent in the routing table; the routing table stabilization process repair inconsistent entries; the change notifier are triggered to inform relevant nodes in the system when certain changes occurred in current table. Before discussion of this problem, some preliminary knowledge will be first given.

6.1 Preliminary Knowledge

6.1.1 Consistent State

We define consistent state upon components on a peer node as follows.

Definition. *A node n is in consistent state iff following invariants are preserved:*

1. *In the network, there is no node n' , which satisfies $n.id < n'.id < n.successor.id$.*

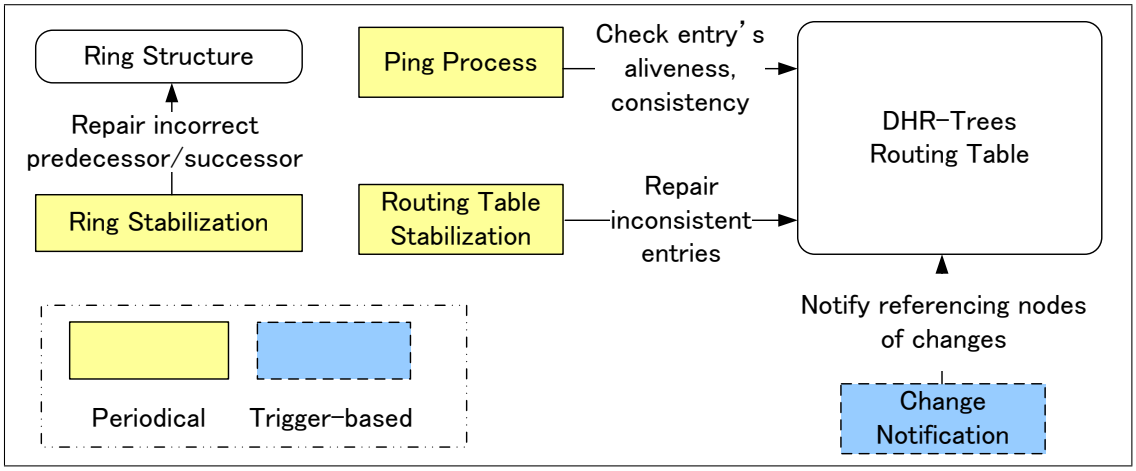


Figure 6.1: Processes for DHR-Trees maintenance

This invariant ensure that the node has correct preceding/succeeding relations on the circle.

2. *Routing table of the node is correctly filled. The criterion of "correctness" is that the properties of routing table holds.*

If all nodes in the network are in consistent state, then the network is said to be in consistent state.

6.1.2 Lookup Procedure

Given a identifier *id*, the lookup is used to find the corresponding node. At the node where the lookup originated, the node selects the farthest away pointer that does not overshoot *id* and forwards the query to that peer. Once the algorithm reaches the lowest level of the DHR-Trees, it traverses the successor list until the value of a peer is equal to *id*. We show the necessary algorithms used in lookup procedure. Algorithm 8 illustrates lookup procedure: the node first set *level* to $MaxLevel$, call *closest_preceding_node* (Algorithm 7) to get closest preceding node to the specified *id*, then forward lookup query to that node. The process is recursively run (at other remote nodes) until the *level* reaches to lowest level of routing table.

Algorithm 7: p.closest_preceding_node (*id*, *level*)

Input: *id* identifier of the peer node to find

Input: *level* the level under which the routing table is traversed

```
1  begin
2      closest = p;
3      while  i < level do
4          forall entryinp.nodes[i] do
5              if closest.id < entry.id AND entry.id > id then
6                  closest = entry;
7              end
8          end
9          i++;
10     end
11     return closest;
12 end
```

6.1.3 Tracker List Structure

To reduce messages of polling routing nodes for their L and U, a Tracker List Structure is introduced. For a peer q, the TLS is a storage for the nodes that have q as a routing entry in their routing tables. When a peer p find q as appropriate routing entry that satisfies the L and U and decided to put q into its routing table at level i, it send q a message informing the addition of q. The q then put the p into the TLS with the level i. We call peer p as a **tracker**. The procedure of setting a Tracker to the TLS is shown in Figure 6.2.

The usage of Tracker List is: when detected change of U at level i, the peer will send the updated U to nodes at level i in the track list. Essentially, the messages of updating U is only triggered when U change occurs. This reduces the message overhead of periodically polling request for U to maintain the consistent state of routing table.

Property of Tracker List Structure The Tracker List Structure can be viewed as a **reverse** for routing table. Because a Tracker p appears in the TLS of a tracked peer q iff the Tracker has q in its routing table, the sum of number of entries in TLS should be

Algorithm 8: p.lookup (id)

Input: id identifier of the peer node to find

```
1  begin
2      level = p.nodes.MaxLevel;
3      if p.id = id then
4          return p;
5      end
6      p' = p;
7      while p'.id ≠ id AND level > 0 do
8          p' = p'.closest_preceding_node(id, level);
9          level − −;
10     end
11     if p'.id = id then
12         return p';
13     else// lookup failed
14
15         return null;
16     end
17 end
```

equal to the total entries in routing table if both are correct. While routing table is core part of DHR-Trees peer-to-peer systems, the TLS is an auxiliary tool to support updates of U item in routing table. The routing table is critical to the system to work properly and must be maintained correctly and refreshed frequently, the TLS is not necessarily to be precisely correct, i.e. the TLS is allowed to contain redundant obsolete Trackers information. When the Tracker p failed, the TLS of q has no means to sense this and is unaware of the failure of p until the q need to send U update message to p and the transmission failed.

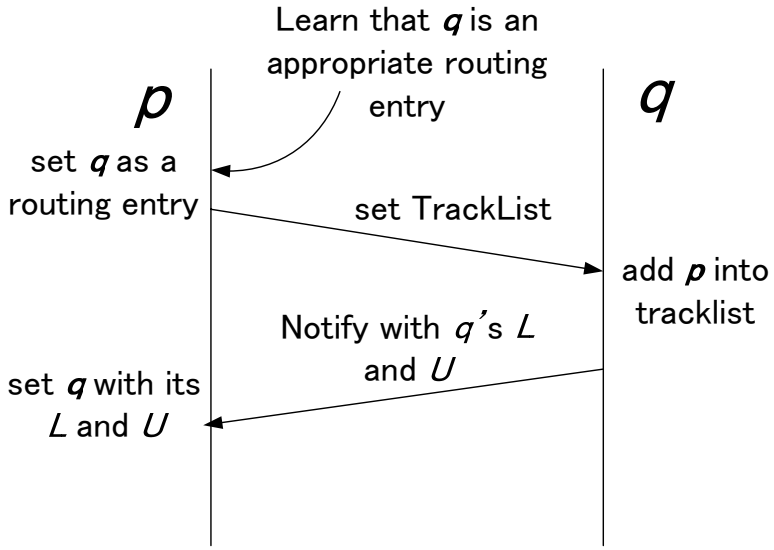


Figure 6.2: Procedure of setting track list

6.2 Node join

To keep the consistent state discussed above, when a node n joins the network, we have to perform three operations: First, initialize the predecessor and successor. Second, initialize routing table of node n. Third, update the routing tables and predecessors/-successors of existing nodes to reflect the change in the network topology caused by the addition of n. We assume that the bootstrapping for a new node is handled off-line, perhaps by someone configuring the newly joining node n with the identifier of at least one other node n' already in the Chord network. Once this is done, node n uses n' to initialize its state. It performs the above three tasks as follows.

Initializing predecessor/successor: Node n ask bootstrapping node n' to find successor for it. The n' will run find_successor to fulfill it, as listed in algorithm 9. It will cost only $log_d N$ messages. After getting acknowledged, n will set it as successor. Then n notifies the successor, allowing successor to set n as its predecessor. The stabilization for ring structure use same algorithm as in [44]. The stabilization process periodically runs at each peer, checking whether the n.successor.predecessor is the n. If not, it will get the n.successor.predecessor as its new successor. This approach can handle concurrent node joins.

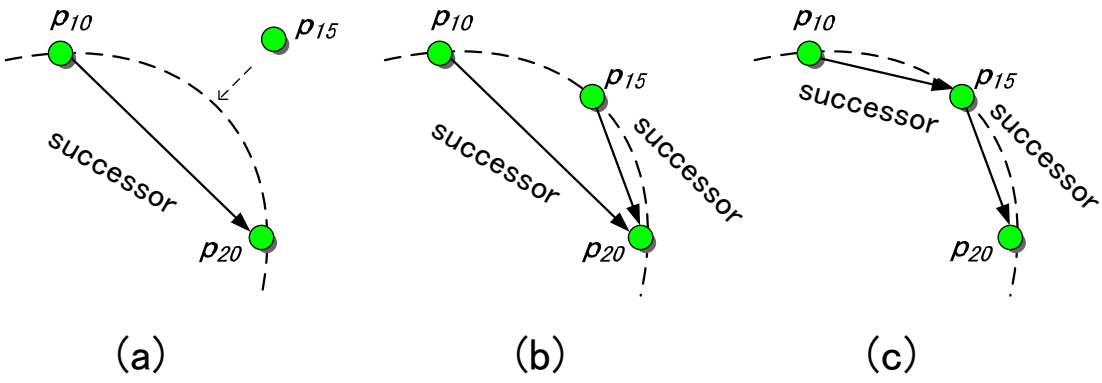


Figure 6.3: Ring structure changes when node joins

Initializing routing table: Having established predecessor/sucessor relationship with previous/next nodes in the identifier space, node n can eventually build its routing table by stabilizationLevel process. However, this will use many messages to find appropriate nodes and fill them into routing table, since finding each node require O($log_d N$) messages. Furthermore, this will take many stabilization cycle until all entries are filled, which leads to long time period before reaching consistent state again. To alleviate this, node n gets a copy of routing table of its successor. Node n send a message to request a copy to its successor. The successor, after getting the message, encloses its routing table into a message and replies to n with it. Node n then only need to replace the first entry at each level with itself. The entries in new routing table may not holds the coverage and separation properties, but succedent stabilization process will do the checking task and, if necessary, the replacing task. This can speed up stabilizing process and reduce messages.

Updating existing nodes' routing tables: After node n joined into the network and established predecessor/successor relationship with certain nodes, its predecessor and successor will be aware its existence. The predecessor p will set n as new successor and insert n into routing table as the first entry in p.nodes[0]. This insertion may cause U of p.nodes[0] to change. If the change occurs, p will send U update messages to trackers contained in the first level of its Tracker List Structure. In turn, the peers received U message may further notifies their trackers of changes in their routing table. In this way, eventually, A small part of the existing nodes will update some entries in their routing

table if needed. The coverage and separation may, however, may subsequently be violated due to updates to the routing tables. In next section, we illustrate the Ping process that periodically checks aliveness of nodes in the routing table and check the consistent state of each entry for the coverage and separation. If the node is not contactable or the coverage/separation condition is broken, then at some time, the routing table will be repaired by stabilization process shown in next section.

Algorithm 9: p.find_successor (id)

Input: id identifier of the peer node to find

```
1  begin
2      p' = p.find_predecessor(id);
3      return p'.successor;
4  end
```

Algorithm 10: p.find_predecessor (id)

Input: id identifier of the peer node to find

```
1  begin
2      level = p.nodes.MaxLevel;
3      if p.id = id then
4          return p;
5      end
6      p' = p;
7      while id ∉ (p'.id, p'.successor) AND level > 0 do
8          p' = p'.closest_preceding_node(id, level);
9          level − −;
10     end
11     return p';
12 end
```

6.3 Maintenance of Ring structure

6.3.1 Ring stabilization

In a dynamic network environment, nodes join and leave frequently with the network evolving. The first definition in the consistent state 6.1.1 implies that all peer nodes are ordered properly by their identifier along the ring structure. This may not holds if there is harsh changes, such as concurrent joins/leaves. To deal with such case, periodical checking and repairing for predecessor/successor relationship is required. DHR-Trees's ring structure uses the similar ring stabilization approach as in Chord[44] to realize this. With strengthening by successor list to be introduced in next subsection, the ring structure is expected to be robust and resilient enough under dynamic network circumstances. The algorithm of ring stabilization is shown in Algorithm 11.

Algorithm 11: p.stabilizeRing

```
1 begin
2     x = p.successor;
3     x = x.predecessor;
4     if x ∈ (p, p.successor) then
5         p.successor = x;
          // Notify x of adding x as p's successor.
          // Let x set p as x's predecessor.
6         notify(x);
7 end
```

6.3.2 Successor list

To improve the robustness of ring structure in DHR-Trees, we use successor list in DHR-Trees peer-to-peer systems. It has similar property of robustness as in other ring structure P2P systems. When a node entry in the successor list failed, it will be removed away and following entries will just shift left. If the length becomes less than predefined system parameter Z, then one of ring stabilization process will send message to the last one n' in the successor list to obtain successor list of n', and fill them into successor list

of node n.

6.3.3 Analysis of ring robustness

We use T_{stabr} as period of ring stabilization, Z as number of entries in successor list and $P_{failure}$ as fail rate, which means the probability of failing per unit time. A node will lose its connection to next nodes on the ring iff all nodes in the successor list failed during T_{stabr}. Then we have: The probability of losing connection to next nodes in the ring is,

$$P_{loss} = \prod_{i=1}^{Z} (P_{failure_i} * T_{stabr}) = (P_{failure} * T_{stabr})^Z$$

For instance, assume the ring stabilization period T_{stabr} is 10s, Z is 16 and $P_{failure}$ is 0.005, the possibility of disconnection is

$$P_{loss} = 1.526 \times 10^{-21}$$

6.4 Maintenance of Routing Table

The main task of maintenance of routing table is to keep consistency of routing table under dynamic network changes. The DHR-Trees has the similar structure as P-Trees. The main difference is that DHR-Trees exclusively uses notification-on-change to actively inform changes to referencing nodes and that a node in DHR-Trees system needs to additionally maintain region information of MBR.

We first introduce the maintenance method for consistency. In particular, we emphasize the method for updating U by exploiting the Tracker List Structure. We then give a analytical result on maintenance cost for routing table, which is not presented in the P-Trees[14]. The experimental results is shown later to prove the correctness of the analysis. Finally, we illustrate the region information maintenance in routing table.

6.4.1 Ping process and Stabilization Process

The Ping process is similar to the P-Trees[14]. For the stabilization process, although it is also used in P-Trees, but the definition of L and approach of retrieving L and U (the *reach* in P-Trees) from other nodes is not addressed enough and no quantitative analysis on traffic cost is given. The P-Trees use periodical processes to detect any changes in referenced peers' routing table, which could bring a lot of redundant traffic even if there is

few changes to occur. The distinct feature of DHR-Trees maintenance is that it introduces a trigger-based process for notifying changes. It uses Tracker List Structure to notify the relevant nodes of the U changes in its routing table. This improvement is important for reducing overhead of maintenance, because the stabilization process is done without inquiring other nodes in the system. At the same time, keeping tracker list does not requires additional messages. As described in subsection 6.1.3, the tracker list for one node is added when referencing nodes setting the node in their routing tables. The stale entry in the tracker list is deleted when sending notification to a node failed.

Algorithm 12: p.Ping (int *level*)

Input: *level* The level of routing table to ping

```
1  begin
2      j = 1;
3      while j < p.nodes[level].numEntries do
4          if p.nodes[level][j].peerhasfailed then
5              Remove(p.nodes[level][j]);
6          else
7              p.nodes[level][j].state =
8              CheckCovSep(p.nodes[l][j-1], p.nodes[l][j]);
9          j++;
10     end
11 end
```

6.4.2 Notification mechanism

As introduces in subsection 6.1.3, Tracker List Structure is employed to notify referencing nodes of changes occurred in local routing table. Different from ping process and stabilization process, the notification process is trigger-based, i.e. only when some *event* occurs, it is called and passed with some parameter.

REACH message. To reduce maintenance messages, a conservative strategy is adopted for notification purpose in DHR-Trees peer-to-peer systems. The notification is only triggered for one kind of event: the change of U of the first entry in the routing

table row, i.e. when $nodes[i][1].U$ changed its value. The message for the U is denoted as REACH.

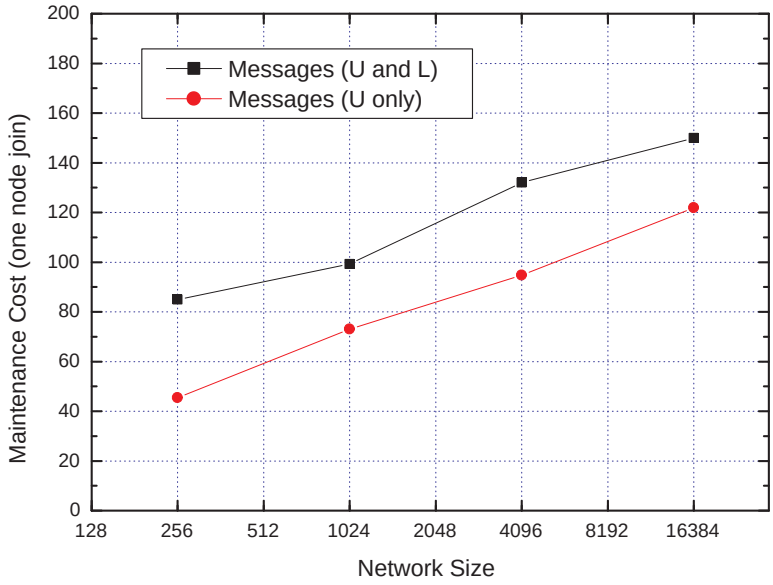
The reason for notifying U only is that, when the coverage property is satisfied, the cost of lookup is guaranteed logarithmically. If the coverage is not satisfied at some peers, the lookup at the peers has to be executed sequentially at level 1 to ensure that no data is missed. In contrast, even if the separation properties is temporarily destroyed, it does not directly affect query performance. Moreover, in the conservative strategy, although the value of L does not has explicit message for update purpose, it will be updated by REACH message which carries not only new U, but also L value (updated or unchanged) when REACH message occurs. By doing these, the U is maintained with high priority and the L is updated some time by REACH message.

The Figure 6.4.2 proved our description above. The Figure 6.4(a) demonstrate that maintenance messages reduced about 18% - 40% with network size varying from 256 - 16384. In Figure 6.4(b), the network size is fixed at 1024. We adding randomly generated nodes into the network, then run spatial query(query window is set for 5 matches in a query) to see the difference for U-only scheme and both U and L notification scheme is enabled. The result proves that there is nearly no difference of query performance between two notification schemes.

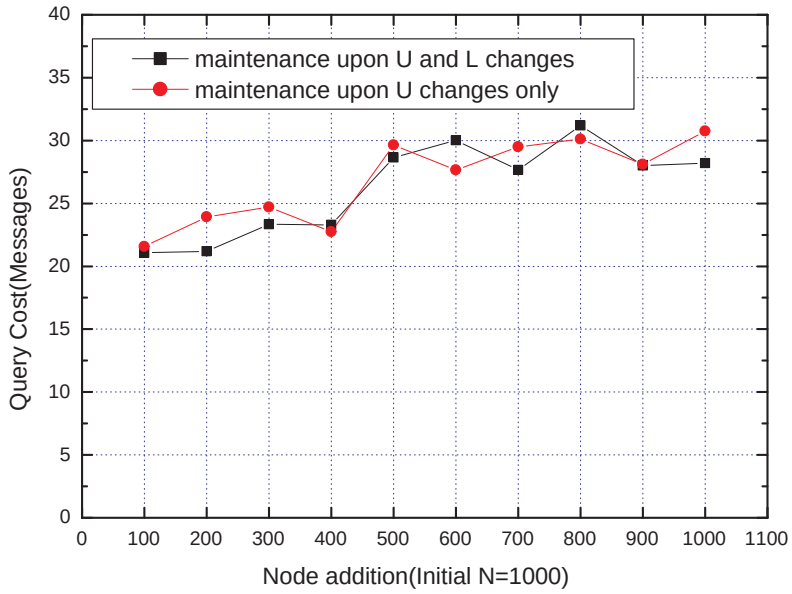
Notification Process. Since a peer p is referenced with its first routing entries at each level (nodes[i][1]), changes to these entries is to be informed to the referencing nodes set S (stored in Tracker List Structure). For all peer s ($s \in S$), after updating entries of p at routing level i, will recalculate L and U of own first entry at upper level. If U is changed, peer s will in turn send REACH message to peers that reference s. If U is unchanged, no message need to send and the notification stops at s. While such notification process is parallel and disperse, other nodes may send REACH nodes to relevant nodes. The notification is bounded at the top level of routing table, where there is no further upper level requiring update.

6.4.3 Theoretical analysis of joining cost

As a node joins into the network, becoming aware of its existence, a part of nodes already existing in the network *eventually* update their routing table to reflects the existence of this node. Some peers need to insert, replace or delete one or more entries in



(a) Maintenance cost(join only) decreases when notifying U changes only



(b) Query performance(Range query, 5 matches)

Figure 6.4: Reducing maintenance cost without degradation in query performance

their routing table to make the routing table consistent again after the node joins. In this section, we theoretically analyze: 1. The number of entry changes in routing tables to occur after a node join; 2. The number of messages notifying U changes to send.

As illustrated in Chapter4, DHR-Trees structure works dependently on the routing tables. In essence, each routing table stores links (or pointers) to certain selected peers in the network. The property *coverage* ensures all peers and data are indexed and logarithmical search efficiency. The property *separation* ensures indexing efficiency by eliminating excessive overlap. In practice, the *coverage* is more important because the sequential scan will occurs when it is violated, which leads to query latency. Temporary *separation* violation, which may increases height of DHR-Trees, does not impose direct influence on query. Therefore, we focus on the quantitative analysis on effort to keep *coverage* consistency.

Overview of Influence First, we generally outline the influence by a newly joined node. When joins into the network, a node finds its successor and copies routing table from it. Then its predecessor will learn its existence by ring stabilization process, and will put it into its routing table at level 1. After changing its routing table, the predecessor p, which in turn referenced by some other peers in their routing table, has to notify them of the change occurred on $p.nodes[1][1].U$ in its routing table by sending REACH message. These peers then update entry of p. These updates, may again leads to changes on $nodes[2][1].U$ in the routing tables of the peers. The peers which has changes on U will send again new REACH messages to notify relevant nodes. This procedure runs recursively until reaching the highest level of the routing tables.

In the following analysis, we will first give some intuitive analysis during the initial phase of recursive notification, then we conclude the formal result. Finally, experiment result is shown to verify correctness of the analysis.

Lemma 1. *With high probability, the number of times for a node to appear at a certain level i (i <p.nodes.MaxLevel) of all other nodes' routing tables is* $D = \frac{3d}{2} - 1$. *The number of times that a node appears in other nodes' routing tables is* $(\frac{3d}{2} - 1) \times \log_{3d/2} N$.

Proof: From the definition of routing table in chapter 4, we have: given a node, it has a routing table of $\log_{3d/2} N$ levels; at each level, there is $[d,2d]$ entries. Let us think of level l, where $l \in [1, \log_{3d/2} N]$. The average numbers of entries at level l is $(\frac{3d}{2} - 1)$, because

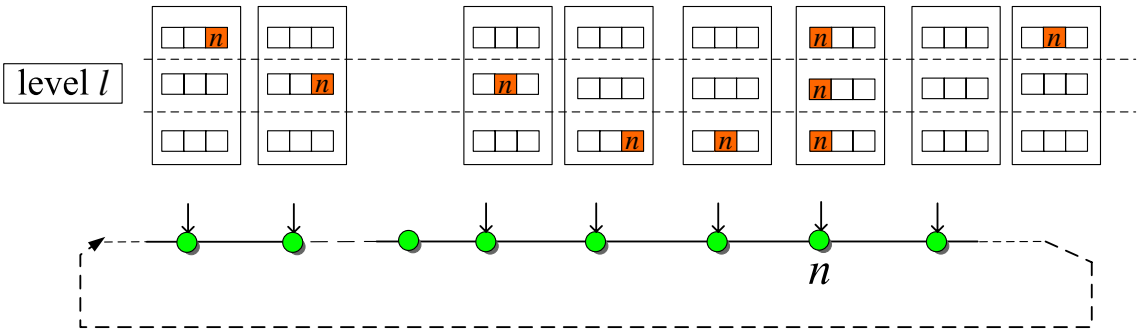


Figure 6.5: Times of being referenced at same level

the length of a table level is expected to be randomly distributed in [d,2d]. Then all nodes in the routing table will have totally $N * (\frac{3d}{2} - 1)$ numbers of entries at level l, which means that there are totally $N * (\frac{3d}{2} - 1)$ level-l references to the nodes in the system. Because the structure is **symmetrical**, each node is referenced $N * (\frac{3d}{2} - 1)/N$ times, i.e. a node tends to appears $(\frac{3d}{2} - 1)$ times in other routing tables at level l. Note that this holds at any level and the routing table has $\log_{3d/2} N$ levels, so a node tends to appear $(\frac{3d}{2} - 1) \times \log_{3d/2} N$ times in other routing tables.

Since the Tracker List Structure is an *inverse image* of routing table, the following corollary holds.

Corollary 1. *On average, the size of Tracker List at level i (i <p.nodes.MaxLevel) is* $(\frac{3d}{2} - 1)$. *The total number of entries in Tracker List structure is* $(\frac{3d}{2} - 1) \times \log_{3d/2} N$.

The Corollary 1 implies that when U of a level occurs, the peer generally needs to send $(\frac{3d}{2} - 1)$ messages notifying of change.

Lemma 2. *A. When a node join the network, the number of peers setting n as its successor is 1. B. The probability for the peer to change its* $nodes[i][1]|_{i=1}.U$ *is 100%. C. The number of REACH message to send is about* $(\frac{3d}{2} - 1)$.

Proof: The part A of Lemma 2 is obvious because the uniqueness in identifier space and ordering along the ring. When a node n join into the network, through the bootstrapping node, it will find out its successor n', which has predecessor p. After ring stabilization process, having detected its successor changed from n' to n, the predecessor p will reset successor to n. Since there is no other nodes more closer to n than p in the anticlockwise direction, the Lemma 2 holds. The part B of Lemma 2 is true because the $nodes[i][1]|_{i=1}.U$

is actually the successor. As the successor changed, the $nodes[i][1]|_{i=1}.U$ will change subsequently. By combining Corollary 1 with Lemma 2 A and B, we can prove correctness of part C in Lemma 2.

Intuitively, Lemma 2 describes direct influence to existing peers in the network when a new node joins. The influence includes two parts: First, the number of table changes to occur is one (A peer set n as successor and altered its routing table once); Second, $(\frac{3d}{2}-1)$ REACH messages will be generated to notify other nodes of change of U. These REACH messages will again affect other peers in their routing tables.

Formally, we denote table changes at level i (change of $nodes[i][1].U$) times as $Tabs_i$, the number of generated REACH for next

Lemma 3. *In routing level p.nodes[i], when one entry p.nodes[i][x]*
($1 < x \leq p.nodes[i].numEntries$*) is changed with its U, the p.nodes*$[i+1][1].U$ *will be changed at probability*

$$P_U = \frac{1}{d+1}\sum_{m=d}^{2d}\frac{1}{m-1} \tag{6.1}$$

Proof: In general, the length of p.nodes[i] ($i < p.nodes.maxLevel$) is expected to be evenly distributed in [d,2d]. We denote m as the length of p.nodes[i]. Then the probability mass function of m is:

$$f(m) = \begin{cases} \frac{1}{d+1}, & d \leq m \leq 2d \\ 0, & otherwise \end{cases}$$

From the definition of routing table, the $p.nodes[i+1][1].U$ is equal to U of last entry at level $p.nodes[i]$. It implies the $p.nodes[i+1][1].U$ will change iff $p.nodes[i][m].U$ is changed. Assume the entry with updated U is located at j of level[i], only if the j happens to be the m, the $p.nodes[i+1][1].U$ will change. When $j < m$, the $p.nodes[i+1][1].U$ will not change directly. So the probability for $p.nodes[i+1][1].U$ to change given p.nodes[i].length is m is

$$P(U_{i+1}|nodes[i].length = m) = \frac{1}{m-1}$$

Therefore, we conclude that $P_{U_{i+1}}$ is independent of level i:

$$P_U = \sum_{m=d}^{2d} P(U_{i+1}|nodes[i].length = m) * f(m) = \frac{1}{d+1}\sum_{m=d}^{2d}\frac{1}{m-1}$$

Lemma 3 illustrates that the probability that a peer update its $nodes[i+1][1].U$ when the peer received a REACH message at level i. If the peer updated the upper reach, it will has to send new REACH messages to peers at level i in its Tracker List.

The P_U is independent on network size N and the routing level i. We calculated its value with d changing from 2-10 and the result is listed in Table 6.1.

Lemma 4. *Consider REACH message when a new node joined, the new U carried by REACH message will probably point to a smaller number.*

Proof: This is because the U can be regarded as a upper bound of index range for the routing entry. When some new nodes with identifer that falls into the range, due to the capacity of the sub-trees, some biggest identifier (the right-most identifier of the sub-trees) will be *squeezed* out from the U. As a result, the U will adjust to new smaller right-most identifier of the sub-trees.

Lemma 5. *In case of one node join, at routing level p.nodes[i], when p received one REACH message for updating entry p.nodes[i][j].U, the probability for the coverage property between p.nodes[i][j] and p.nodes[i][j+1] to be violated, i.e. the probability P_{mr_i} (The **r**eplacement caused by a **m**essage) for the p.nodes[i][j + 1] to be replaced is expected to be,*

$$P_{mr_i} = \frac{1}{d^{i-1}} \sum_{m=1}^{d^{i-1}} \frac{1}{m}$$

Proof: Based on Lemma4, the *coverage* between $p.nodes[i][j]$ and $p.nodes[i][j+1]$ is to be destroyed iff lower bound (the $HCode$) of $p.nodes[i][j+1]$ happens to be equal to original $p.nodes[i][j].U$. As $p.nodes[i][j+1].HCode$ is evenly distributed in $[p.nodes[i][j].L + 1, p.nodes[i][j].U]$ and

$$(U - L - 1) \in [1, d^{i-1}]$$

, the probability for *coverage* to be destroyed is expected to be,

$$P_{mr_i} = \frac{1}{d^{i-1}} \sum_{m=p.nodes[i][j].L+1}^{p.nodes[i][j].L+d^{i-1}+1} \frac{1}{m - (p.nodes[i][j].L)}$$

$$P_{mr_i} = \frac{1}{d^{i-1}} \sum_{m=1}^{d^{i-1}} \frac{1}{m}$$

Based on above Lemmas, we conclude Lemma 6 as follows. In Lemma 6, it implies that the number of routing table changes and the number number of notifying messages is independent of network size.

Lemma 6. *Given the system is in stable state and routing tables are randomly distributed, consider a node join into the system, the total number of messages to notify routing table changes is*

$$S_{msg} = \sum_{i=1}^{H} Msg_i$$

where

$$H = \left\lceil \log_{\frac{3d}{2}} N \right\rceil,\ M_i = D \cdot P_{U_{i-1}} \cdot M_{i-1} \cdot (1 + P_{mr_{i-1}} \cdot P_{ru_{i-1}})$$

The Msg_i is defined as total notification messages received at routing level i through the whole network, and H is $ceiling(\log_{3d/2} N)$. When receiving totally M_{i-1} at level $i-1$ $(i > 0)$ through all peers' routing table, the total number of routing entries at level $i-1$ $(i > 0)$ to change is approximately

$$C_{i-1} = M_{i-1} \cdot P_{mr_{i-1}}$$

The total number of newly generated messages at level $i-1$ to notify current level $i (i > 0)$ is

$$\begin{aligned} M_i &= (C_{i-1} \cdot P_{ru_{i-1}} + M_{i-1}) \cdot P_{U_{i-1}} \cdot D \\ &= D \cdot P_{U_{i-1}} \cdot M_{i-1} \cdot (1 + P_{mr_{i-1}} \cdot P_{ru_{i-1}}) \end{aligned}$$

For bootstrapping, the value is $(i = 0)$: $M_0 = 0, P_{mr_0} = 1, P_{U_0} = 1, C_0 = 1$. Other parameter when $i > 0$ is defined above.

Note that the expression of C_i is an approximation. It omitted possible changes on the right side of the entry change induced by a message. The reason for this is that entries entry changed by message is guaranteed under the condition of node join only.

6.4.4 General Form of Maintenance Cost

In previous subsection, the cost including message cost and routing table changes are analyzed for case of node join. As to node failure, the analysis is similar except that all referencing nodes to the failure node need to find substitute for it. As described in Lemma 1, a peer appears in other peers' routing table about D times at each level. This implies two important thing: The first is that when a peer joins into the network, it will sooner or later be incorporated into other routing tables and will appears D times at certain routing level. The second implication is that when a peer leaves or fails, referencing peers will replace the peer in their routing table D times at certain routing level. When such a

replacement occurs, it may affect the right-side routing entry, and may further affect the U of current level, i.e. the upper level will change the U. Formally, the replacement will cause the routing level to change its U at probability P_{ru_i}. So the Msg_i should additionally include D entry substitutes as follows:

$$M_i = D \cdot (C_{i-1} \cdot P_{U_{i-1}} + D \cdot P_{ru_{i-1}})$$
$$= D \cdot P_{U_{i-1}} \cdot M_{i-1} \cdot (1 + P_{mr_{i-1}} \cdot P_{ru_{i-1}}) + D^2 \cdot P_{ru_{i-1}}$$

Therefore, the general form of cost for node addition into the network or deletion from the network evaluation purpose is as follows.

$$S_{msg} = \sum_{i=1}^{H} M_i$$

where, the M_i refers to the number of messages created at certain routing level i, and H is $\log_{3d/2} N$. The M_i is defined as: Because the $P_{mr_{i-1}}$ decreases quickly with i, the S is simplified as:

$$S_{msg} = \sum_{i=1}^{H} M_i = \sum_{i=1}^{H} (D \cdot P_{U_{i-1}} \cdot M_{i-1} + D^2 \cdot P_{ru_{i-1}}) \tag{6.2}$$

In this expression, the $P_{U_{i-1}}$ is a constant when the parameter d is known. However, due the uncertainty for right-side entries to replace, and the new value for further new entries, the $P_{ru_{i-1}}$ is difficult to calculate. Assuming the $P_{ru_{i-1}}$ is unified to P_{ru} for a certain system, the we have (The proof is given in Appendix A.2):

Theorem 1. *Consider a node join or failure, the upper bound for total number of generated notification messages is*

$$S_{msg} = \frac{a \cdot (a^H - 1)}{(a-1)} \cdot M_0 + \frac{b \cdot [a^{H+1} - H \cdot (a-1) - a]}{(a-1)^2}$$

where

$$D = \frac{3d}{2} - 1,\ H = \left\lceil \log_{\frac{3d}{2}} N \right\rceil,\ a = D \cdot P_U,\ b = D^2 \cdot P_{ru}$$

For any random replacement in routing table, suppose the probability for consistent state of next right-side entry to be violated is a constant P_v, similar to the calculation of P_U, we can prove the:

$$P_{ru} = \frac{(1 - P_v^D)}{(1 - P_v) \cdot D}$$

then we have simplified maintenance cost as:

$$S = \frac{a \cdot (a^H - 1)}{(a-1)} \cdot M_0 + \frac{D \cdot (1 - P_v^D)[a^{H+1} - a - H \cdot (a-1)]}{(1 - P_v)(a-1)^2} \tag{6.3}$$

6.4.5 Verification of Analysis

To verify the correctness of Lemma 6, we run a number of simulation test. We set the network size N to 4096, the order of DHR-Trees to 4. We first let the peer-to-peer system reached stable state, i.e. no network membership changes and the routing tables of all peers reach to consistent state. Then randomly generated 100 nodes are added into the network, every time before adding a node, we run stabilize to let all peers maintain their routing tables to reach consistent state. The result is collected statistics on routing table changes and messages generated for notifying purpose.

Table 6.2 shows the analysis result for DHR-Trees p2p systems with order 4. The value of Msg_i and C_i both are independent of network size N. The unlisted P_{U_i} is 0.22 ($i \geq 1$).

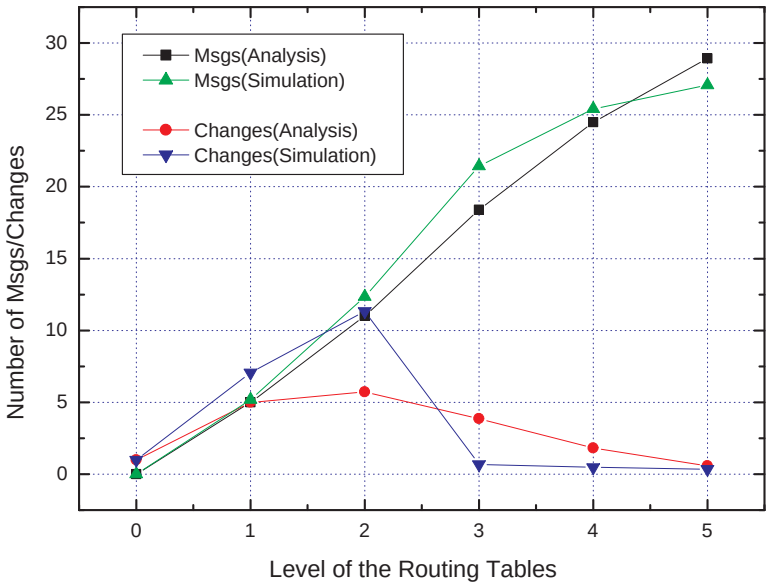


Figure 6.6: Analysis vs. Simulation on cost when a node joins

To verify Theorem 1, we run simulation with network size N varying from 256 to 16384, with order d changed from 4 to 8. For the expression 6.3, we set P_v as 0.5, which is a experimental value in practice. We then compare the result in Figure 6.7.

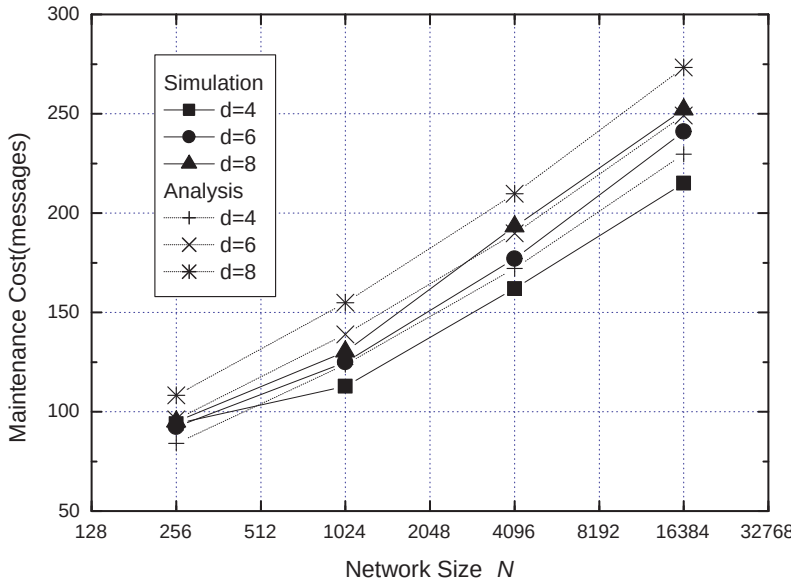


Figure 6.7: The average cost for node join and failure

6.4.6 Maintaining Region Information in Routing Table

One of the challenges for structured P2P network is maintaining peer's routing table while the network constantly changes. DHR-Trees' semi-independent structure enable each peer to maintain its composite routing table properly. The maintenance process is nearly the same as in P-Tree[14], except that $peer.MBR$ needs additional update. The $peer.HCode$ is updated as its equivalent $peer.value$ in P-Tree. With regard to MBR, p only needs to calculate MBR for the first entry, in which the $peer$ is the p itself, by calling $UpdateMBR$ method. $UpdateMBR$ method is shown in Algorithm 14. Other entries ($p.nodes[i][j].MBR$ (j>1)) will be acquired and updated from corresponding peers, $p.nodes[i][j].peer$, during $modified$ p.stabilizeLevel(i) process (In P-Tree, stabilizeLevel is periodically called to keep routing table fresh and correct. For details, see [14]). The only exception is that when at level 1, the MBR is the original rectangle of data set that $peer$ owns and there is no need to call $UpdateMBR$.

Through our experiments, as shown in Figure 6.8 (uniform distribution), almost all maintenance messages are generated by underlying overlay network, while update of MBR requires very few messages to inform other nodes. The basic intuition of such result is:

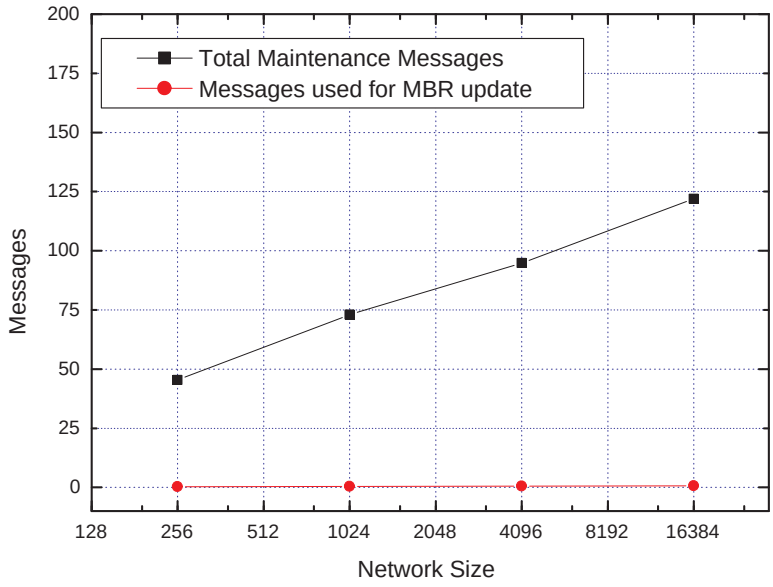


Figure 6.8: Total Maintenance Messages and MBR update messages (joins only)

when any change occurs, e.g. peers join/leave/fail, the update message of underlying binary index of HCode (the P-Trees) are propagated to all related nodes where the MBR should also be updated. Therefore, there are very few exception in which the MBR need to be updated using explicit messages. As a conclusion, the multidimensional structure of DHR-Trees require the same class maintenance cost as in P-Trees.

In practice, the MBR also employs the Tracker List Structure and is updated by the form of trigger-based notification. When MBR is renewed, a message that carries new MBR is sent to referencing nodes. For spatial query to be correctly executed, the MBR information must keep up with the change of the network. This may leads to overhead when the peer-to-peer network changes quickly. To relax such updating requirements, the system can choose usage of Adaptive Bounding Rectangle instead of MBR, which use automatically-calculated region information for spatial query purpose and does not require real region information. The adaptive bounding rectangle will be introduced in the chapter 8.

6.5 Scalability of DHR-Trees

As well as one of the distinct feature of peer-to-peer systems, the dynamism of network is also a challenge requiring to address enough. To maintain the consistent of the routing tables, high maintenance cost can cause network traffic overhead seriously when network grows very large. In this section, we discuss the scalability of DHR-Trees by analyzing upper bound of maintenance cost.

In the equation 6.2, the $P_{U_{i-1}}$ is independent to the i and can be given by 6.1. If the $P_{ru_{i-1}}$ can be assumed to be level-independent P_{ru}, then the equation 6.2 will be:

$$S_{msg} = \sum_{i=1}^{H} M_i = \sum_{i=1}^{H} (D \cdot P_U \cdot M_{i-1} + D^2 \cdot P_{ru}) \tag{6.4}$$

Given M_0 is known, then 6.4 is:

$$S_{msg} = \frac{a \cdot (a^H - 1)}{(a-1)} \cdot M_0 + \frac{b \cdot [a^{H+1} - H \cdot (a-1) - a]}{(a-1)^2} \tag{6.5}$$

where $a = D \cdot P_U$ and $b = D^2 \cdot P_{ru}$. Because P_{ru} is the one and only variable, so it is obvious that S_{msg} is maximum when P_{ru} takes value 1. So formally, we have following conclusion.

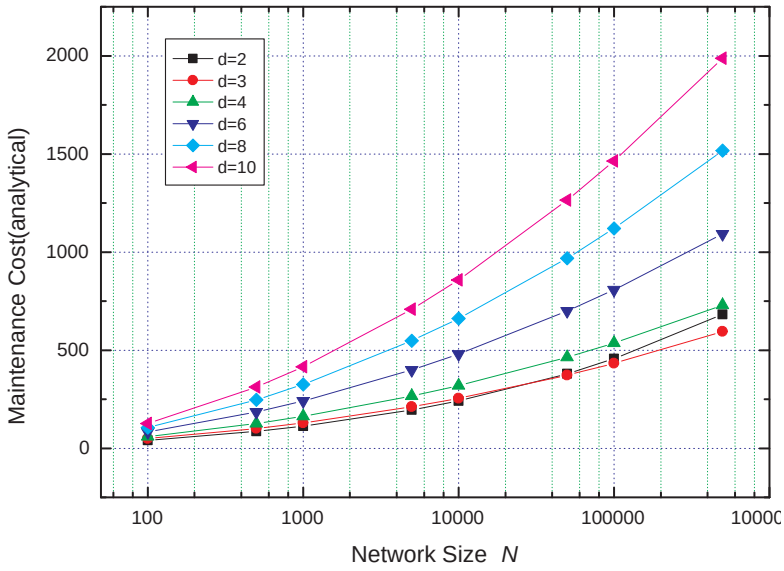


Figure 6.9: The scalability on maintenance cost

Theorem 2. *Consider a node join or failure, the upper bound for total number of generated notification messages is*

$$S_{msg} = \frac{a \cdot (a^H - 1)}{(a-1)} \cdot M_0 + \frac{D^2 \cdot [a^{H+1} - H \cdot (a-1) - a]}{(a-1)^2}$$

where

$$D = \frac{3d}{2} - 1,\ a = D \cdot P_U,\ H = \left\lceil \log_{\frac{3d}{2}} N \right\rceil$$

In the above formula, total message S includes two parts: the first part with M_0 is created by the direct notification messages bottom-up, as described in previous subsection; The second part can be viewed as the subsequent cost for adding or removing the peer in other routing tables.

From the Figure 6.9, even the network size increases exponentially, the maintenance cost for a node addition or deletion increases nearly linearly. Moreover, the figure illustrates that the maintenance cost increases when big value is selected as the degree d of DHR-Trees.

6.6 Summary

One of the main concern of scalability in structured peer-to-peer systems is the maintenance cost. Structured design provides efficient lookup and query performance, but the maintenance is indispensable and must be efficient and lightweight in the dynamic network environment. The number of messages being used for maintenance purpose is commonly considered as the maintenance cost.

In this chapter, after the consistent state, lookup procedure and tracker list structure being first introduced, we described the process for a node to join the network. We then illustrate the processes for maintenance, ping process to maintain the ring structure and stabilization process incorporating with notification mechanism to maintain the routing tables. We then give analysis of cost for routing table maintenance when a node joins. Finally, a more general form of cost is given. Our simulation concludes that the cost increases almost logarithmically to the network size. As a result, it proved the **scalability** of the DHR-Trees.

Algorithm 13: p.Stabilize(int *level*)

Input: *level* The level of routing table to stabilize

```
1  begin
2      j = 1;
3      while j < p.nodes[level].numEntries do
4          if p.nodes[l][j].state! = consistent then
5              prevPeer = p.nodes[l][j - 1].peer;
6              newPeer = succ(prevPeer.nodes[l - 1][d - 1].peer);
7              if p.nodes[l][j].state == coverage then
8                  INSERT(p.nodes[l],j,newPeer);
9                  p.nodes[l].numEntries++(max 2d);
10             else
11                 REPLACE(p.nodes[l],j,newPeer);
12             p.nodes[l][j + 1].state
13             = CheckCovSep(p.nodes[l][j], p.nodes[l][j + 1]);
14         if  COVERS(p.nodes[l][j], p.HCode) then
15             p.nodes[l].numEntries = j + 1;
16         j++;
17     end
18     while ¬COVERS(p.nodes[l][j − 1], p.HCode) ⋀ j < d do
19         prevPeer = p.nodes[l][j-1].peer;
20         newPeer = succ(prevPeer.nodes[l-1][d-1].peer);
21         INSERT(p.nodes[l],j,newPeer);
22         j++;
23     end
24     if COVERS(p.nodes[l][j − 1], p.HCode) then
25         return true;
26         else
27             return false;
28 end
```

order d	2	3	4	5	6	7	8	9	10
P_U	0.6111	0.3208	0.2186	0.1659	0.1388	0.1121	0.0847	0.0965	0.0754

Table 6.1: The Probability P_U

level (i)	Msg_i(input)	C_i	P_{mr}	M_{i+1} (level i output)
0	0	1	1	5
1	5	5	1	11
2	11	5.72	0.52	18.39
3	18.39	3.86	0.21	24.47
4	24.47	1.81	0.074	28.92
5	28.92	0.58	0.02	32.45

Table 6.2: Notifying Messages(Msg) and Routing Table Changes (Changes)

Algorithm 14: p.UpdateMBR (int i)

Input: i level of routing table, i >1

```
1 begin
2     reset p.nodes[i][1].MBR to empty;
3     j = 1;
4     while j < p.nodes[i − 1].numEntries do
5         p.nodes[i][1].MBR = CombineRect(p.nodes[i][1].MBR,
          p.nodes[i − 1][j].MBR);
6         j + +;
7     end
8 end
```

Chapter 7

Improving Fault-Tolerance in DHR-Trees

In this chapter, we discuss the fault-tolerance of query execution in DHR-Trees peer-to-peer systems. We also give the method to make DHR-Trees more fault-tolerant in query processing. The methods include the usage of Adaptive Bounding Rectangle, and the introducing of successor list as alternatives for entries in routing tables.

7.1 Problem with Region information Update

In chapter 4, the DHR-Trees structure and its properties are demonstrated. Essentially, a DHR-Tree (stored the routing table) on a peer node is composed of information along the left-most root-to-leaf path of a Hilbert R-Tree, in which the peer node itself is located at the left-bottom of the Tree.

The capability of supporting multidimensional indexing and querying is dependent on the region information, which is stored in the routing table. So the correctness of indexing and querying is guaranteed as long as the relevant region information is correctly stored in routing tables of relevant nodes in the P2P system. This is not a problem if the system is changing slowly and the stabilization process is run relatively frequently, i.e. no frequent nodes' joining/leaving/failing occur, and the minimum bounding region information of peer nodes generate relatively few updates during the runtime.

However, even if the system changes slowly, the DHR-Trees can not guarantee all qualified data to be retrieved at 100%. Considering query that are send to the network before the routing table being updated to reflect new comer nodes, the query result will not include the data stored on the new comer, even though the new comer is existing in the network. After all, the DHR-Trees is a best-effort system in case of dynamically

changing network. This is because that some query may unfortunately goes to the nodes that are still under inconsistent state, though such a rate could be expected to be very low.

7.2 Using Adaptive Bounding Rectangle

In this chapter, we propose new approach with regard to the region information. Rather than using real region information (the MBR), we instead use the "adaptive bounding rectangle" (denoted as ABR) information. Though the ABR may contain much bigger region than the real MBR, but it can be calculated with HCode already in the routing table by a node itself, without contacting other nodes for retrieving fresh region information. This new approach has the following advantages:

1. **Less maintenance traffic**.

   It eliminates the requirement of retrieving and updating MBR information from other nodes. So that the messages for stabilization purpose need not anymore carry MBR information during stabilization process. This decreases the size of maintenance message, which originally requires to carry MBR and AuxMBR in a routing entry of a peer node.

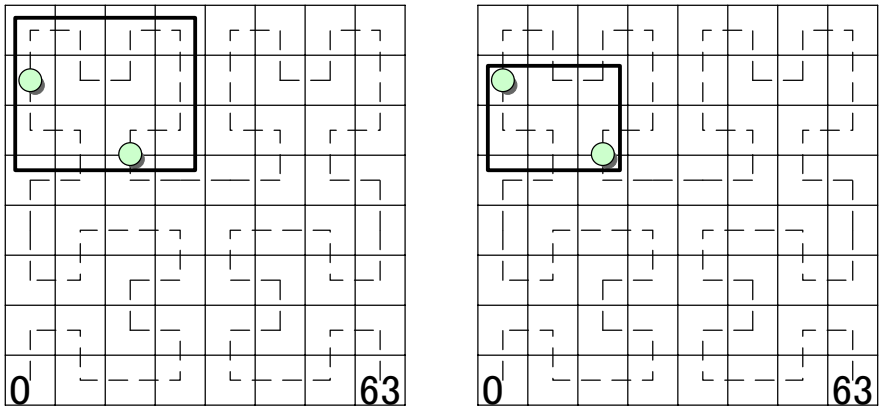
2. **Higher quality of query result under churn**

   The region information is automatically calculated by the peer node itself. So the condition of consistent state of nodes become unnecessary for processing query correctly. Therefore even under the higher churn rate, the systems becomes possible to answer queries without missing data by combining with entry success list in next subsection.

The ABR for each routing entry in the routing table is defined as:

Adaptive Bounding Rectangle. *Given a routing entry in the routing table, the adaptive bounding rectangle (ABR) is the minimum bounding rectangle of Hilbert curve segment between its Hilbert code (HCode) and the closest next Hilbert code in the routing table.*

Figure 7.1(a) illustrates a ABR between two points. For comparison purpose, the MBR is illustrated in Figure 7.1(b). In a DHR-Tree, a MBR is calculated with the region information of all sub-trees. So the correctness of the MBR is **dependent** on the region



(a) Adaptive Bounding Rectangle (b) Minimum Bounding Rectangle

Figure 7.1: difference between ABR and MBR

information of all subjacent region information in the routing table. But in a dynamic network environment, the frequent membership change induces MBR change, which may seriously affect the correctness of MBR. *Synchronizing* and Checking correctness of such information has to be run frequently to keep up with the changes, which causes network traffic by many messages.

7.3 An Example

Let us see an example of ABR and see the calculation of ABR. In figure reffig:abr-example, suppose we need to calculate the ABR of first routing entry at level 3. To calculate it, we need only know the next closest HCode, which is 110 in this example. We use 20 and 110 as input and run algorithm A.1. Then we can get the region information. Note this procedure does not require any information of the subjacent information. The calculating is straightforward and not influenced by any failure or incorrectness of subjacent routing entries.

However, the calculation of MBR is dependent on the subjacent information and sensitive to changes below. It requires that region information of all entries at level 2 is correctly stored. For instance, if a node with HCode 70 joined the network, the region information in nodes[2][2] is not updated until the stabilization process executed.

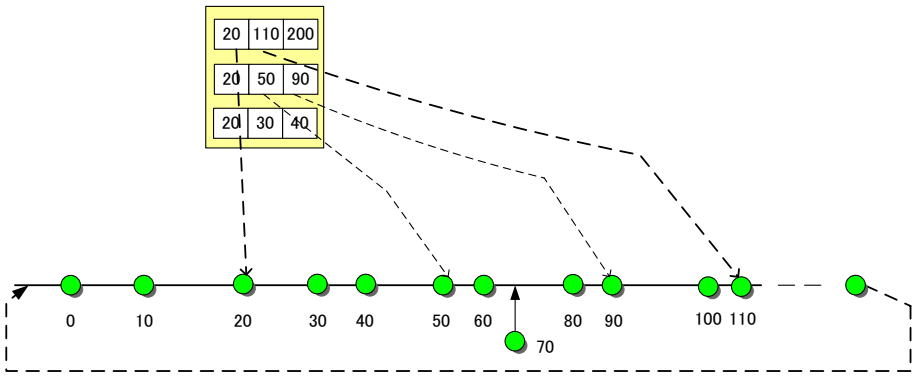
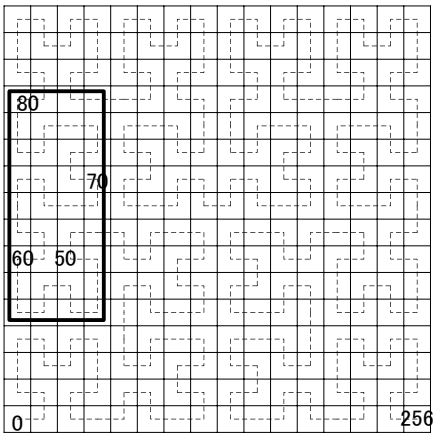
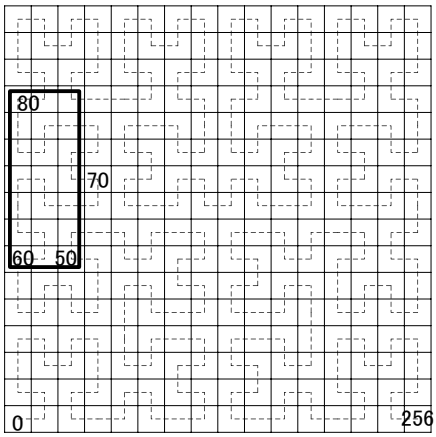


Figure 7.2: An example of Adaptive Bounding Rectangle



(a) ABR enclose all possible subjacent region



(b) MBR needs update to reflect point 70

Figure 7.3: Advantage of ABR

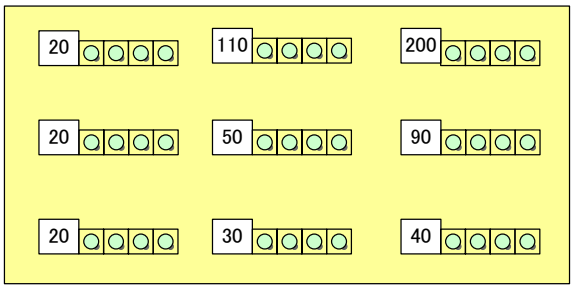


Figure 7.4: The Entry Successor List

7.4 Strengthening query path by successor lists

In structured peer-to-peer systems, queries are executed distributed and cooperatively. When an query is sent to the network, the peer node who receives the query will play the role of query executor and result collector. To resolve the query, as described in section5.2, the node lookups its routing table, finds nodes that may possess required data, and then forwards the query or sub-queries towards them. In consequence, the correctness of routing table determines whether the query can be executed correctly.

For DHR-Trees p2p systems, it experiences the same problem in dynamic network environment. The participating nodes may fail suddenly, which consequently results in stale entries in routing table which contains pointers to them. So the queries to be forwarded to these nodes will have to terminate exceptionally. When such problems occur, it becomes impossible to fulfill the query, or only partial results can be retrieved. This may extremely deteriorate p2p network reliability and service quality.

To mitigate such problems, several approaches are possibly to be effective. Among them, one method is to find alternate nodes to forward the query. As to ring structure like Chord, one choice is to route query along the ring using successor instead of routing table, though the query performance will avoidably worsen by bigger latency. Chord[44] has reported that latency increases as more successor are used for routing query.

We take a novel approach to strengthen the query path. It makes query path more reliable and robust. The idea is inspired by successor list for a peer node. The details of the approach is as following: In a routing table, each entry contains additional successor list to the peer node in the entry; the successor list is just the same as the one being hold at the peer node. When needs to forward a query to the peer node, the current node try to forward and if the time-out occurs in underlying network to reach the peer node, then

the destination node must have already failed or left the network. The current node, then choose a successor of the destination node from the routing table entry as a substitute and send the query to the substitute. The figure 7.4 shows the routing table of peer node with identifier 20. And the successor list here is 4.

Probability of Routing Success. First let us consider a single forwarding step. Since the successor list works as the one for ring structure, the forward will success at the following probability:

Assume the query involve s forwarding steps to fulfill, then the query will success iff all forwarding goes successfully. So the global success is at probability as:

$$P_{success} = \prod_{i=1}^{s} \mathrm{P}_{\mathrm{forward}} = \prod_{i=1}^{s} (1 - (\mathrm{P}_{\mathrm{failure}} * T_{stabr})^z) = (1 - (\mathrm{P}_{\mathrm{failure}} * T_{stabr})^z)^s$$

7.5 Evaluation

Improvement in fault-tolerance of query.

Using adaptive bounding rectangle, DHR-Trees P2P systems is expected to be more failure-tolerant and the MBR updating load could also be alleviated.

To verify the effectiveness of successor lists added into routing table, we run simulation and have nodes suddenly failed. Without stabilization processes stopped, the range query is executed and the number of range success is measured. We tested two scenarios: one is without successor lists equipped and another is with successor lists equipped. We run 150 range queries each and see how many range query finally succussed without any interruption by broken routing path. The results illustrated in Figure 7.5 shows that the scenario using entry successor list greatly improved that path effectiveness and success rate of range query decreased little even when massive nodes failed.

Degradation in query cost.

The adoption of ABR can solve data missing problem and make the system more reliable because it eliminated the need of updating region information when nodes joins or leaves. However, it can lead to slight degradation in query performance. This is because ABR contains some unnecessary region information, which leads to, for example, that a range query goes to some irrelevant nodes due to in the network. We evaluated the performance

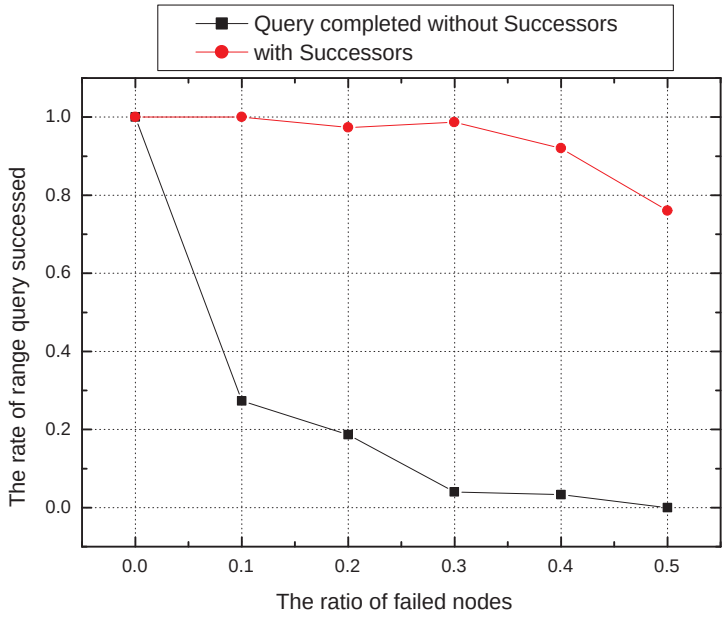


Figure 7.5: The improvement by entry successor list

of the range query and verified this experimentally. In the experiment in Figure 7.5, the network size is set to 1000, the order of DHR-Trees d is 4 and the *matches* is set to 5 data items.

7.6 Summary

Since the MBR is the minimum region information that enclose spatial information of sub-trees below, as illustrated in Figure 7.5, it execute query efficiently that ABR. The ABR, which does not directly require information exchange between peers, can decreases communication traffic in maintenance. This is helpful in band-limited network environment. Moreover, by combining with entry successor list, which though does not holds any MBR information at all, spatial query can be executed with high guarantee even under highly churning environment. As the trade-off of fault-tolerance improvement, ABR approach require more query message to exchange (for instance, range query execution).

As a result, to select MBR or ABR, the tradeoff between spatial query cost and fault-tolerance has to be considered. If the network is with less churn, we select MBR to decrease spatial query result. In contrast, if the network is too dynamic with more churn,

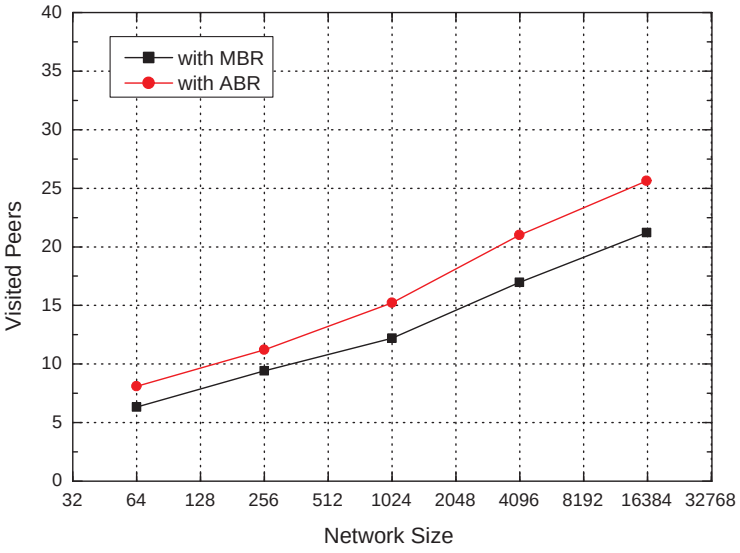


Figure 7.6: Query Cost Degeneration with ABR

the ABR can be a good choice so that the query can be guaranteed with answers with higher probability, though it requires more messages to fulfill the query.

Issue	DHR-Trees with MBR (Minimum Bounding Rectangle)	DHR-Trees with ABR (Adaptive Bounding Rectangle)
Region Precision	Precise	Contains extra regions
Spatial query cost	Less	More
To obtain region information	Inquire owner nodes	Calculate on the fly
Updating overhead	Yes	No
Query fault-tolerance	No	Yes

Table 7.1: Comparison of MBR and ABR

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, the peer-to-peer systems to support spatial data sharing are studied. Although there are many existing approach for supporting data sharing, most of them are designed for one-dimensional data and only a number of them is capable of complex query such as range queries. Moreover, few of them can support spatial data sharing among participating peers. The difficulties we identified is the peer-to-peer structure that should be able to support handling network dynamism as well as spatial data indexing, since the continual network membership changes is a regular issue in peer-to-peer systems.

In this thesis, two new novel peer-to-peer protocols are proposed. Both GNet and DHR-Trees are designed to be architectures for sharing geographically distributed spatial data. In particular, DHR-Tree's structure, properties, query support, fault-tolerance and maintenance cost method and theoretical analysis are provided in this thesis. To the best of our knowledge, DHR-Trees structure is the first peer-to-peer structure that has semi-independent R-Trees structure and is capable of supporting region-based multidimensional search predicates efficiently, while handling network dynamism efficiently as well.

Comparing with other protocols (see Table 8.1), The DHR-Trees has good resilience under network churn, rich search predicates to support, logarithmical maintenance cost for scalability and query result guarantee under dynamic network environment. We believe our approaches can help realization of certain distributed spatial data sharing applications. We hope our works will stimulate more research interest in both peer-to-peer structures and spatial data sharing applications.

Protocol	GeoPeer	Squid	ZNet	P2PR-Tree	GNet	DHR-Trees
Dimension Mapping	No	By HSFC	By Z-Curve	No	Geographical address	By HSFC
Topology	Mesh	Ring	Skip List	Tree	Tree	Ring
Routing table size	$O(m)$	$log(n)$	$log(n)$	$d * log_d N$	$d * log_d N$	$d * log_d N$
Routing path length (hops)	$O(\sqrt{N})$ in worst case	$logn$	$d * log_d N$	$\leq log_d N$	$\leq log_d N$	$\leq log_d N$
Churn Resilience of structure	△ (long range contact)	○	○	×	×	○
Logarithmical Maintenance Cost	○	○	○	×	×	○
Range Query Cost	○	△ (cluster decomposition requirement)	△	○	○	○
Nearest Neighbor Query Support	×	×	×	×	○	○
Query Result Guarantee	○	○	△ (probabilistic)	×	×	○ (using ABR)

*m is the dimension of space; n is the size of identifier space; N is the number of nodes; d is the order of Tree;

Table 8.1: P2P protocols comparison

8.2 Future Work for DHR-Trees

Access Control. Peer-to-peer systems present a particular challenge for the access control problem due to their open and anonymous nature. In DHR-Trees p2p systems, a special concern arises: If the user application is allowed to run some kinds of queries without any control, then malicious behaviors could be a potential threat to the system.

Imagine some user client application run range queries with query windows setting to the whole spatial area, all peer nodes in the system will be probably involved into the query execution and the query result content could probably be huge. This may saturate the network with query messages and result messages quickly.

A possible solution is employing a central certification or identification authority. As an alternative, constraining query window or trimming query window smaller is a possible approach. Before processing query and forwarding refined sub-queries, decision could be made to reject the query request at routing layer of DHR-Trees.

Handling Heterogeneity. By now, most peer-to-peer systems is based on the assumption of equivalency of all peer devices. In the future, the variety of computing devices to be used in the peer-to-peer systems is an important issue to address. Different computing power, storage, available network bandwidth characterize heterogeneity of devices.

Current DHR-Trees p2p systems does not address heterogeneity issue. Some *transient* peer nodes could degrade stability, because the system has to frequently adjust to handle network member changes. Furthermore, if the system (the routing table of peers) is in the inconsistent state or there are many stale routing entries in the routing table, the query performance may degrade sharply in worst case.

One possible solution is to make a multi-tier DHR-Trees p2p system. Peers are categorized by the computing capability, storage and transiency. Powerful and stable (with long lifetime) nodes compose the top-tier of the network, which can be regarded as *backbone* of the systems. Each top-tier peer node then additionally acts as a super-peer to manage an smaller DHR-Trees system below, in which second-class peers are clustered. Peer nodes in the second-tier DHR-Trees, in turn can be a super-node to manage third-class peers, etc. Considering spatial data sharing, spatially local clustering is expected. In multi-tier DHR-Trees, maintenance cost decreases due to that under tier nodes need not join top-tier ring structure, instead join into local ring structure. The query performance is kept due

to hierarchy of DHR-Trees.

Chapter A

Appendix

A.1 Calculation of Adaptive Bounding Rectangle

The problem of calculating Adaptive Bounding Rectangle is described as8: given two Hilbert value a and b, calculate the minimum rectangle that bounds the Hilbert curve segment between a and b. The naive approach to calculate the ABR is to do a traversal of all possible Hilbert value between start point and end point, decoding them into locations in Euclidean space, and then merge them together into a rectangle. However, this can be too costly since the segment can be very long when the order of the Hilbert curve is high. Here, after introduction of the Hilbert Space Filling Curve, we illustrate the method that we used in practice.

A.1.1 Hilbert Space Filling Curve

Mapping multi-dimensional data to one dimension, enabling simple and well-understood one-dimensional access methods to be exploited, has been suggested as a solution in the literature in the multidimensional database area. One way of realizing such a mapping is to utilize space-filling curves which pass through every point in a space once so giving a one-one correspondence between the coordinates of the points and the one-dimensional sequence numbers of the points on the curve.

Many mapping method have been proposed and evaluated. The most prominent ones include the z-order, Hilbert curve. Hilbert curve and others were a topic of interest for leading pure mathematicians in the late 19th century and the first graphical representation of one was given by David Hilbert in 1891.

The basic Hilbert curve of a 2*2 grid, denoted H1, is shown in Fig. A.1. The procedure to derive higher orders of the Hilbert curve is to rotate and reflect the curve at vertex 0 and at vertex 3. The curve can keep growing recursively by following the same rotation and reflection pattern at each vertex of the basic curve. Fig. 2.3 also shows the Hilbert curves of order 2 and 3.

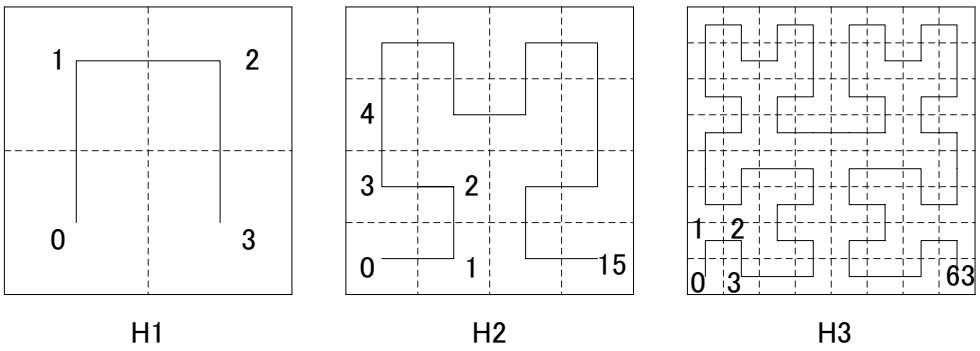


Figure A.1: Hilbert curve of order 1, 2, and 3

A.1.2 The Fast Recursive Algorithm

We designed this algorithm to improve the speed of calculating Adaptive Bounding Rectangle. The basic idea is to recursively decompose the space into some subspaces, checking the Hilbert value range $[s, t]$ of each subspace and process them by comparing the range with the Hilbert curve segment between $[a, b]$. There are three cases of relationships of the range of subspace and range of Hilbert curve segment. The cases and handling are as follows, where R denoted as the range of subspace $[s, t]$ and S denoted as segment range $[a, b]$.

1. **Separation.** If $R \cap S = \emptyset$, then the subspace is discarded. No further decomposition on it is needed.

2. **Containment.** If $R \cap S = R$, then the subspace is merged into the ABR. No further decomposition on it is needed.

3. **Intersection.** If $R \cap S \neq \emptyset$, then the subspace needs further decomposition.

The recursive algorithm stops when either the process has reached the finest level of the Hilbert curve or no more intersection requires further decomposition. For reference,

the source code is listed as follows.

Listing A.1: Calculation of ABR

```
/**
* Given Hilbert value range between a and b,
* calculate the Adaptive Bounding Rectangle.
* @param a lower bound(inclusive)
* @param b upper bound(inclusive)
* @return the ABR of a and b.
*/
public static Region getABR(long a, long b) {
    if(a<=b)
        return calcRect(a, b, 0, 1, null);
    else
        return null;
}
/**
* @param a lower bound(inclusive)
* @param b upper bound(inclusive)
* @param start The start point that is set as 0 at beginning.
* @param decompose_order Decomposition order, initial value 1.
* @constant ORDER The order of Hilbert Space Filling Curve.
*/
private static Region calcRect(long a, long b, long start,
int decompose_order, Region mbr)
{
    int order_diff = ORDER-decompose_order;
    Region r;

    int coords[],x,y;
    int boundary[] = new int[4];

    long min, max;
    long span = 1<<(order_diff<<1);
    for(int i=0;i<4;i++){
    //check four blocks at one decomposition.
        min = (start + i)*span;
        max = (start + i + 1)*span-1;
        //contain this block
        if(a<=min && b>=max){
            r = new Region();
```

```
            //get the coodination of corner point of the block.
            coords = Hilbert.decode2d(start+i,decompose_order);
            x = coords[0]; y = coords[1];
            boundary[0] = x << order_diff;
            boundary[1] = y << order_diff;
            boundary[2] = ((x+1) << order_diff) - 1;
            boundary[3] = ((y+1) << order_diff)-1;
            r.set(boundary);
            if(mbr==null) mbr = r;
            else mbr = Region.CombineRegion(mbr,r);
        }
        //not overlap this block, skip this block.
        else if(a>max || b<min){}
        //partially contain the block.
        //further refinement required
        else{
            if(decompose_order<ORDER){
                r = calcRect(a, b, (start+i)<<2,
                    decompose_order+1, mbr);
                if(mbr==null) mbr = r;
                else mbr = Region.CombineRegion(mbr,r);
            }
        }
    }
    return mbr;
}
```

A.2 The Proof of Theorem 1

Theorem 1 is described as:

Consider a node join or failure, the total number of generated messages to notify changes is

$$S_{msg} = \frac{a \cdot (a^H - 1)}{(a-1)} \cdot M_0 + \frac{b \cdot [a^{H+1} - H \cdot (a-1) - a]}{(a-1)^2}$$

The Proof:

Because

$$M_i = D \cdot P_{U_{i-1}} \cdot M_{i-1} + D^2 \cdot P_{ru_{i-1}}$$

we denote $D \cdot P_U$ as a, and $D^2 \cdot P_{ru_{i-1}}$ as b, we have:

$$\begin{aligned}
&M_1 = a \cdot M_0 + b \\
&M_2 = a \cdot M_1 + b = a^2 M_0 + ab + b \\
&\vdots \\
&M_H = a \cdot M_{H-1} + b = a^H M_0 + a^{H-1} b + a^{H-2} b + \cdots + ab + b
\end{aligned}$$

Then

$$S = \sum_{i=1}^{H} M_i = (a + a^2 + \cdots + a^H) \cdot M_0 + \underbrace{a^{H-1} b + 2a^{H-2} b + 3a^{H-3} b + \cdots + H \cdot b}_{part\ U}$$

For part U,

$$\begin{aligned}
U &= a^{H-1} b + 2a^{H-2} b + 3a^{H-3} b + \cdots + H \cdot b \\
U \cdot a &= a^H b + 2a^{H-1} b + 3a^{H-2} b + \cdots + H \cdot ab
\end{aligned}$$

Solve U as:

$$U = \frac{b \cdot [a^{H+1} - H \cdot (a-1) - a]}{(a-1)^2}$$

Finally, we conclude:

$$S_{msg} = \frac{a \cdot (a^H - 1)}{(a-1)} \cdot M_0 + \frac{b \cdot [a^{H+1} - H \cdot (a-1) - a]}{(a-1)^2}$$

where

$$D = \frac{3d}{2} - 1,\ H = \left\lceil \log_{\frac{3d}{2}} N \right\rceil,\ a = D \cdot P_U,\ b = D^2 \cdot P_{ru}$$

Bibliography

[1] The gnutella web site. http://gnutella.wego.com.

[2] The kazaa web site. http://www.kazaa.com.

[3] Hilbert r-tree: An improved r-tree using fractals. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 500–509, Santiago, Chile, 1994.

[4] Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.

[5] Karl Aberer. *P-Grid: A Self-Organizing Access Structure for P2P Information Systems*, volume 2172. January 2001.

[6] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.

[7] A. Andrzejak and Zhichen Xu. Scalable, efficient range queries for grid information services. In *Peer-to-Peer Computing, 2002. (P2P 2002). Proceedings. Second International Conference on*, pages 33–40, 2002.

[8] F. Araujo and L. Rodrigues. Geopeer: a location-aware peer-to-peer system. *Network Computing and Applications, 2004.(NCA 2004). Proceedings. Third IEEE International Symposium on*, pages 39–46.

[9] James Aspnes and Gauri Shah. Skip graphs. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 384–393, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[10] Farnoush Banaei-Kashani and Cyrus Shahabi. Swam: a family of access methods for similarity-search in peer-to-peer data networks. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 304–313, New York, NY, USA, 2004. ACM Press.

[11] P.A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. *Proceedings of the WebDB Workshop*, 2002.

[12] B. Carton and V. Mesaros. Improving the scalability of logarithmic-degree dht-based peer-to-peer networks. *Proc. of EUROPAR*, 2004.

[13] G. Chen and D. Kotz. A survey of context-aware mobile computing research. *Dartmouth Computer Science Technical Report TR2000-381*, 2000.

[14] Adina Crainiceanu, Prakash Linga, Johannes Gehrke, and Jayavel Shanmugasundaram. Querying peer-to-peer networks using p-trees. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 25–30, New York, NY, USA, 2004. ACM Press.

[15] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. *Proc. of IPTPS*, 58, 2003.

[16] C. Faloutsos and I. Kamel. *Beyond uniformity and independence: analysis of R-trees using the concept of fractal dimension.* ACM Press New York, NY, USA, 1994.

[17] Volker Gaede and Oliver Gunther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[18] Prasanna Ganesan, Beverly Yang, and Hector Garcia-Molina. One torus to rule them all: multi-dimensional queries in p2p systems. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 19–24, New York, NY, USA, 2004. ACM Press.

[19] PB Gibbons, B. Karp, Y. Ke, and S. Nath. Irisnet: an architecture for a worldwide sensor web. *Pervasive Computing, IEEE*, 2(4):22–33, 2003.

[20] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 381–394, New York, NY, USA, 2003. ACM Press.

[21] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, New York, NY, USA, 1984. ACM Press.

[22] A.Y. Halevy, Z.G. Ives, P. Mork, and I. Tatarinov. Piazza: data management infrastructure for semantic web applications. *Proceedings of the twelfth international conference on World Wide Web*, pages 556–567, 2003.

[23] U. Hengartner and P. Steenkiste. Exploiting information relationships for access control. *Proceedings of 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005)*, pages 269–278, 2005.

[24] R. Huebsch, J.M. Hellerstein, N. Lanham, B.T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. *Proceedings of the 29th VLDB*, 2003.

[25] T. Imielinski and J.C. Navas. Gps-based geographic addressing, routing, and resource discovery. *Commun. ACM*, 42(4):86–92, 1999.

[26] H. V. Jagadish, Beng Chin Ooi, and Quang Hieu Vu. Baton: a balanced tree structure for peer-to-peer networks. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 661–672. VLDB Endowment, 2005.

[27] Helio Tejedor Navarro Jordi Pujol Ahullo, Ruben Mondejar Andreu. Planetsim. http://planet.urv.es/planetsim/.

[28] Hye-Young Kang, Bog-Ja Lim, and Ki-Joune Li. *P2P Spatial Query Processing by Delaunay Triangulation*, volume 3428. January 2005.

[29] B. Karp and HT Kung. Gpsr: greedy perimeter stateless routing for wireless networks. *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, 2000.

[30] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 233–242, 2002.

[31] R. Lopez-Gulliver, H. Tochigi, T. Sato, M. Suzuki, and N. Hagita. Senseweb: collaborative image classification in a multi-user interaction environment. *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 456–459, 2004.

[32] Q. Lv, P. Cao, E. Cohen, K. Li, and Shenker. Search and replication in unstructured peer-to-peer networks. *Proceedings of the 16th international conference on Supercomputing*, pages 84–95, 2002.

[33] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192, New York, NY, USA, 2002. ACM Press.

[34] K. Minami and D. Kotz. Secure context-sensitive authorization. *Pervasive and Mobile Computing*, 1(1):123–156, 2005.

[35] Anirban Mondal, Yi Lifu, and Masaru Kitsuregawa. *P2PR-Tree: An R-Tree-Based Spatial Index for Peer-to-Peer Environments*, volume 3268. January 2004.

[36] C. Greg Plaxton, Rajmohan Rajaraman, and Andrew W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320, New York, NY, USA, 1997. ACM Press.

[37] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[38] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. *Proceedings of the USENIX Annual Technical Conference*, 2004.

[39] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 71–79, 1995.

[40] Antony Rowstron and Peter Druschel. *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*, volume 2218. January 2001.

[41] C. Schmidt and M. Parashar. Enabling flexible queries with guarantees in p2p systems. *Internet Computing, IEEE*, 8(3):19–26, 2004.

[42] Yanfeng Shu, Beng Chin Ooi, Kian-Lee Tan, and Aoying Zhou. Supporting multi-dimensional range queries in peer-to-peer systems. In *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, pages 173–180, 2005.

[43] Joo-Han Song, Vincent W.S. Wong, and Victor C.M. Leung. A framework of secure location service for position-based ad hoc routing. In *PE-WASUN '04: Proceedings of the 1st ACM international workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, pages 99–106, New York, NY, USA, 2004. ACM Press.

[44] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[45] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 189–202, New York, NY, USA, 2006. ACM Press.

[46] E. Tanin, A. Harwood, and H. Samet. A distributed quadtree index for peer-to-peer settings. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 254–255, 2005.

[47] Yannis Theodoridis and Timos Sellis. A model for the prediction of r-tree performance. In *PODS '96: Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 161–171, New York, NY, USA, 1996. ACM Press.

[48] Haojun Wang, Roger Zimmermann, and Wei-Shinn Ku. Aspen: an adaptive spatial peer-to-peer network. In *GIS '05: Proceedings of the 13th annual ACM international workshop on Geographic information systems*, pages 230–239, New York, NY, USA, 2005. ACM Press.

[49] Shenyquan Wang, Dong Xuan, and Wei Zhao. On resilience of structured peer-to-peer systems. In *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, volume 7, pages 3851–3856 vol.7, 2003.

[50] Xinfa Wei and Kaora Sezaki. Gnet: a peer-to-peer protocol for internet scale location-based applications. In *Embedded Software and Systems, 2005. Second International Conference on*, pages 8 pp.–, 2005.

[51] Xinfa Wei and Kaoru Sezaki. Dhr-trees: A distributed multidimensional indexing structure for p2p systems. *ispdc*, 0:281–290, 2006.

[52] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 5–14, 2002.

[53] B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Computer*, 2001.

[54] R. Zimmermann, We-Shinn Ku, and Haojun Wang. Spatial data query support in peer-to-peer systems. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, volume 2, pages 82–85 vol.2, 2004.