NOTE TO USERS

This reproduction is the best copy available.

uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Xun Ye

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S.

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Design and Implementation of Routing Algorithms for Supporting Multi-dimensional Range Query in HD Tree

TITRE DE LA THÈSE / TITLE OF THESIS

A. Boukerche

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

G. Wainer

A. El Saddik

Gary W. Slater

Le Doyen de la Faculté des études superieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Design and Implementation of Routing Algorithms for Supporting Multi-dimensional Range Query in HD Tree

by

Xun Ye

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.Sc. degree in
Computer Science

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-74159-7
Our file Notre référence
ISBN: 978-0-494-74159-7

Abstract

The HD Tree is a novel data structure, which was recently proposed by Yunfeng Gu to support multi-dimensional range queries in a distributed environment. My thesis work is a follow-up to the research on the HD Tree. It focuses on the basic routing strategies in order to support multi-dimensional range queries in the HD Tree. There are two basic routing strategies supported by the HD Tree data structure: hierarchical routing and distributed routing. Hierarchical routing is an adapted version of routing in the tree structure, while distributed routing is a novel routing strategy built over the distributed structure of the HD Tree in order to accommondate the distributed environment. Hierarchical routing can be applied to any source and destination pair in the HD Tree, but results in a massive work load on nodes near the root. It also has very limited options in case of a node failure. Although distributed routing is applied only to the source and destination pair at the same depth of the HD Tree, its communication load is distributed to two sets of nodes at neighboring depths. Distributed routing has more options to survive a node failure. In my thesis work, four distributed routing algorithms were explored. Each of these four is a stand-alone routing algorithm, and can be applied in different routing conditions. Both theoretical analysis and experimental results indicate that any routing algorithm alone is able to achieve similar or better performance compared to the equivalent tree structure. However, the significance of this thesis work pertains not only to the performance issue. These different routing algorithms provide multiple options to accommondate a changing environment. Based on my thesis work, there were two advanced routing algorithms developed, a X2X routing algorithm which is a combination of four distributed routing algorithms, and a DROCR algorithm which is a combination of the hierarchical and distributed routing algorithms. These algorithms not only achieve considerable performance gain over its equivalent tree structure, but also make the routing in the HD Tree highly error resilient and load balancing. They are the fundamental algorithms in supporting multi-dimensional range queries in the HD Tree.

Acknowledgments

I take great pleasure in thanking the many people who have helped me in my studies at the University of Ottawa. First, I would like to thank my supervisor, Professor Azzedine Boukerche, for his invaluable guidance and advice, and much needed financial support. I would also like to thank PhD. candidate Yunfeng Gu, for his kindly help with my research work. Finally, I would like to extend my thanks to my parents and my colleagues, whose love and encouragement have always sustained me.

Contents

List of Tables

List of Figures

List of Publications

The following publications are relevant to the topic of this thesis and have been authored by Xun Ye.

Yunfeng Gu, Azzedine Boukerche, Xun Ye, and Regina B. Araujo, *Supporting Multi-dimensional Range Query in HD Tree* . The 14th IEEE DS-RT, Fairfac, Virginia, USA, Oct. 17-20, 2010 (in press).

Chapter 1

Introduction

Peer-to-peer[12], often refered to as P2P, is a distributed network architecture often used in the network application level. P2P networks provide a fully distributed environment, which allows every participant in the network to directly share the resources of any other participants. This kind of fully distributed environment is very different from the client-server network, where all the resources are stored in the server, and the clients can only get the resource from the server side. Based on the structure difference between client-server and P2P networks, we can conclude that the P2P network can be more scalable and fault tolerable than the client-server network. P2P networks have become popular since they were first used in the file sharing system. P2P networks also need to support complex queries, such as range queries and key queries. A well designed P2P overlay network system should have low routing complexity and steady performance even in a large scale network.

1.1 Motivation

As P2P network applications become more popular, the requirements for P2P overlay systems increase. The support for the information query of the network is more and more complicated. Basically, there are two classes of information queries: key queries

and range queries.

Key queries, which are always applied in a single dimensional data space, is concerned about the service of fetching back the corresponding data by the identifier key we give. Key queries have already been well studied, and efficiently supported by many existing P2P overlay systems. P2P overlay systems can be classified into two classes: structured P2P overlay networks and unstructured P2P overlay networks. The most common P2P structured overlay networks for key queries are based on a distributed hash table (DHT)[15][26]. DHT uses the hash function to coordinate each key with the value of the data object, so that we can easily get any value by using a given key. There are many existing studies about DHT-based P2P applications, such as CAN[3, 17], Chord[10] and Pastry[11]. The performance for these P2P overlay networks are all in logarithmic time, but the maintenance of the topology costs more, because they have to keep on updating the information globally. The unstructured P2P overlay networks do not have this kind of topology maintenance problem, since in the unstructured P2P systems, the peers are located on a random graph. Every peer in the network is an independent object, and they are fully distributed. There is no fixed organization or topology for the location of the peers. All unstructured P2P overlay networks function through the ad hoc network. Unstructured P2P overlay networks use flooding[23], random walking[24], or Time-to-Live (TTL)[25] for searching. Since a benefit is its low cost of maintenance, many unstructured P2P overlay networks have been widely used in real life. There are some networks we might be familiar with, such as Gnutella[8], Freenet[6, 7], and BitTorrent[9]. However, because of the ad hoc nature and flooding-based routing of the unstructured P2P overlay network, the correctness and performance of the routing, consumption of network bandwidth, and system scalability are all uncertain.

Range queries are more complicated, especially multi-dimensional range queries. The request of a range query uses the location range to search the data instead of using the corresponding keys, which requires that the system not change the location of the data during construction. The multidimensional range query has brought two main require-

ments to the P2P overlay system. One requirement is application constraint, which is how to partition and map the data space into the identifier space of the distributed system. Application constraint allows us to easily find the identifier according to the corresponding data object, and find the data object according to the corresponding identifier. The other requirement is system constraint, which is how to organize and manage the identifiers of the distributed system. This requires the system to preserve the data object locations, and maintain the topology of the system. All the above systems we have mentioned are designed for exact key queries, they can not be directly adapted to multidimensional range queries. However, there are many studies on distributed overlay networks, which also support multidimensional range queries, such as SONAR[1], SCARP[5], MURK[5], ZNet[26], Skipindex[27] and Squid[28]. For application constraint, in order to split the data space into balanced parts without affecting the locality of the data, the existing P2P overlay systems for multidimensional range queries use some well known data structures, like K-d tree[19, 20], Quadtree[18, 19], Z-order curve[20, 21] and Hilbert curve[22]. All of these have been proven to preserve the locality effectively. These data structures have a common feature: they all use the recursive decomposition to do the partition. For system constraint, MURK, Squid and SONAR use the existing CHORD and CAN, which were originally designed for exact key queries. SCARP, ZNet and Skipindex use the new structure Skip Graph[4], which was designed for multidimensional range queries. We will see later that, as the third party test results show, these structures can not support multidimensional range queries efficiently.

In previous work[14], Yunfeng Gu proposed a new data structure, the Hierarchically Distributed Tree (HD Tree). The HD Tree is a structured P2P overlay network system which is built over a complete tree structure. More than the hierarchical tree structure, HD Tree also comprises the distributed structure. Routing for the HD Tree is more multiplex. Besides the basic hierarchical routing for the tree, the routing algorithms in the HD Tree also include distributed routing, which makes the HD Tree structure more tolerant of node failure. After observing the existing data structures used for application

constraint, the structures all use recursive decomposition, which can be easily represented by the tree structure. Therefore we have used the tree structure as the basic structure of the HD Tree, and adapted the hierarchical structure to the distributed environment.

My thesis work is an extension of Yunfeng Gu's study of the HD Tree. Four basic hierarchical routing operations and four basic distributed routing operations for the HD Tree topology are proposed in this thesis. This thesis work can also prove that the HD Tree has the following features:

- Fault tolerance: There have been found multiple routes between any two nodes in the HD Tree based on hierarchical or distributed routing, so that the routing algorithms in the HD Tree can support fault tolerance. If the next node on the route is not available because of failure or high load, the routing algorithm will immediately redirect the routing request to any of its available neighbors, and start the routing process over again.

- Load balancing: Since routing in the HD Tree contains two parts, hierarchical routing and distributed routing, we can choose hierarchical links, distributed links, or both hierarchical and distributed links to complete the routing.

- Support multi-dimensional range query: Range query finds the minimum complete set of nodes, which holds the data covered by the requested range. The HD Tree maintains the locality of the multi-dimensional data. In the implementation of the routing algorithm in the HD Tree, hierarchical routing is used to search the nodes from top to bottom, and distributed routing to find the nodes at the same depth to get the requested range of the data.

1.2 Contributions

The main contributions of this thesis are:

- Four hierarchical routing operations, and four distributed routing algorithms are specialized for the newly proposed P2P overlay network system, the Hierarchically Distributed Tree (HD Tree), to guarantee the routings in this new data structure. Therefore, each node can choose a routing algorithm to get to any other node in the HD Tree structure.

- Since we have proposed more than one routing algorithm, there should be multiple routes between any two nodes in the HD Tree structure. Even if a following node in the route fails, the routing can simply change its direction by choosing a different routing algorithm. It proves that the HD Tree is fault tolerable.

- The distributed routing algorithms are used to apply the routing between any two nodes at the same depth in the HD Tree by using only the level of nodes the source and destination belongs to, and the level of their parents. Therefore, these routings can be achieved without interrupting the higher level nodes. These distributed routing algorithms can significantly reduce the work load of the nodes closer to the root.

- As we will discusse in Chapter 5, and show by the simulation results in Chapter 6, two advanced routing algorithms can be developed based on my thesis work: X2X[43] and DROCR[43]. They have not only significantly improved the routing performance over the equivalent tree structure, but also made the HD Tree structure efficient in its support of multi-dimensional range queries.

1.3 Thesis Objective

The main objective of this thesis is to design and implement distributed routing algorithms for the new data structure HD Tree, which could be used in future studies. This thesis will also show that distributed routing can present higher capabilities in dealing with node failure and load balancing problems. Towards this objective, this thesis

addresses the following components.

- A comprehensive study of the P2P overlay network is presented. The study is carried out based on how the existing P2P overlay network systems can support multi-dimensional range queries, and the performance of the P2P overlay network systems' routing algorithms.
- A brief introduction to the HD Tree. This thesis will mainly talk about why to propose this data structure, what attributes and improvements it has given to the study of the P2P overlay network, and the definitions and features of the HD Tree related to our work.
- Four basic hierarchical routing operations will be presented specifically for the HD Tree, which we call 2DP, 2HP, 2DC, 2HC. By using these basic operations, we can then get four basic distributed routing algorithms, H2D, D2H, H2H, D2D. In order to improve the performance of these distributed routing algorithms, this thesis also defined four matching functions of the HD Tree, one for each distributed routing algorithm, which can help to skip some route steps. The distributed routing algorithms show multiple routes between any two nodes at the same depth of the HD Tree, which can provide more reliable and fault tolerable routing.

1.4 Thesis Organization

The thesis is organized into the following chapters:

- Chapter 2 presents a background study of the existing P2P overlay network systems for both key queries and multi-dimensional range queries, the challenges for system design to support multi-dimensional range queries, and some existing routing performance of well-known P2P overlay network systems.
- Chapter 3 gives a brief view of a new data structure, the hierarchically distributed tree (HD Tree). This chapter includes some basic definitions of the HD Tree, the

special properties of the HD Tree, and the technical improvement the HD Tree achieved to support multi-dimensional range queries.

- Chapter 4 presents a detailed introduction of our routing algorithms, H2D, D2H, H2H, and D2D, for the HD Tree in terms of the motivation, and the important characteristics of the routing algorithms. We also present two improved routing algorithms: X2X and DROCR, which are developed based on the routing algorithms in my thesis work.

- Chapter 5 gives a brief discussion about how the routing algorithms we have designed can help to prove that the HD Tree can efficiently support multi-dimensional range queries.

- Chapter 6 presents the performance evaluation of H2D, D2H, H2H, D2D, X2X and DROCR, along with the experimental results.

- Chapter 7 concludes this thesis, and includes some suggestions for additional research in the future study of the HD Tree.

Chapter 2

Background

In this chapter, some existing work is introduced related to this thesis. We first define the information query, specify its characteristics and discuss the requirements it brings to P2P overlay networks. We then focus on the multi-dimensional range query, compare it with the key query, and provide an overview of how the existing P2P overlay network structures support them.

2.1 Key Query

In a distributed system, the data space is partitioned and mapped to the identifier space. The single dimensional key query is concerned with the service of fetching back the corresponding data by the identifier key we give. The data distributed management (DDM) here mainly works on distributing each data piece onto each peer in the identifier space using an appropriate strategy.

The process of partition and mapping for a key query includes extracting the data space onto the key space, mapping the key space to the identifier space, and assigning nodeIDs to the peers in the identifier space. The most common scheme used for a key query is a distributed hash table (DHT). It provides a service like a hash table, which stores (key, value) pairs, so that every node can retrieve its value efficiently by a given

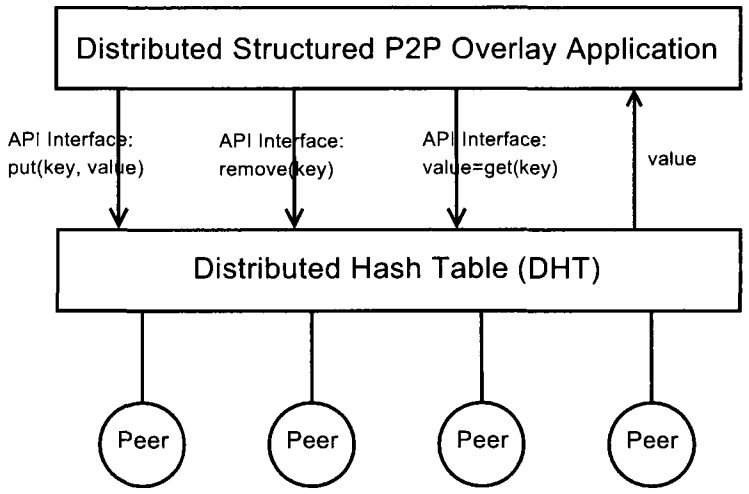


Figure 2.1: Overview of DHT-based P2P overlay system

key. This kind of maintaining and mapping the keys to the distributed nodes makes the DHT scalable to a large amount of nodes, and highly fault tolerant. As the Figure 2.1 shows (adapted from [17]), the P2P overlay networks support the scalable storage and retrieval of {key,value} pairs on the overlay network. DHT-based systems use *put(key, value)* to distribute unique key to each identifiers, and simply use *value=get(key)* to retrieve the value of the node.

Based on the topology of these P2P overlay systems, we can classify them into two classes: unstructured P2P overlay systems and structured P2P overlay systems. Structured P2P overlay systems may also be called DHT-based systems.

2.1.1 Unstructured P2P Overlay System

In unstructured P2P systems, the peers are located on a random graph. They are fully distributed. There is no fixed organization or topology for the location of the peers. When a peer needs to join or leave the network, it only needs to notify the neighbors, which makes the unstructured network easy to maintain. The most popular algorithms for locating resources in unstructured systems are flooding and random walk. Flooding

is adequate when applied in a small area, but as the area of the network grows, it will become much more inefficient. Random walk, on the other hand, is different from flooding. It randomly chooses some neighbors on each step, but the performance of random walk mainly depends on the number of the neighbors it chooses and the life time of each message query. In order to get a high performance, the parameters of random walk need to be set to appropriate values.

Gnutella[8, 39, 40] is the first decentralized file sharing system, which was developed by Nullsoft. It is a network protocol, which allows all the peers in the network to share files with or download files from each other. Every node in the network is connected to an ultrapeer, which stores the information of shared files between the users. In the classic Gnutella model, when the request is sent node by node, and arrives at the destination node, it will send the response message through the query from which the request comes. After using the ultrapeers, the destination node can create a direct connection to the original node, and the search result will be delivered over the User Diagram Protocol (UDP).

Freenet[6, 7] implemented in a P2P network, queries the request node to node until it reaches the destination. It is fully distributed; no node in the system is privileged over other nodes. Every node stores its own local data and the addresses and keys of the other nodes it chose. The basic routing mode in Freenet is passing a request from node to node in a chain of proxy query. Every node decides the next node to go to based on the IP address, and every node only knows its immediate up and down nodes in the query chain. In order to prevent infinite searching, every request has a limited life time, the "hops-to-live", which decreases every time the request gets to a node. When the "hops-to-live" has been decreased to zero, and the node is not the destination node, the result will be sent back with a failure message. Every request is also assigned a unique identifier, so that if a node has seen the identifier twice, it means the request query has a loop. When this happens, the processing node will immediately send the request to another node, until it gets to the destination node. A success or failure message will be

sent back to the original node by retrieving the query chain after the search stops.

BitTorrent[9, 41, 42] is the most commonly used P2P file sharing protocol for distributing large amounts of data. In a Bittorrent network, when a node wants to transfer a file, it will first divide the whole file into small chunks, with each chunk hashed using SHA-1 hash function[16]. This initiate file sharing node is the seed, and other nodes want to download the file are peers. The seed creates connections to all peers, but they receive different chunks of the file from the seed. After the multiple peers received multiple chunks of file data, the Bittorrent network allows them to connect with each other, so that each peer can also be the seed. This kind of switch between peer and seed can take over the uploading and downloading load of the initiate seed, reduce the network usage, and efficiently improve the speed of file sharing. Every peer has to download a torrent file from the seed. The torrent file has the extension *.torrent*, which contains an announce section. The information about the file name, file length, number of chunks, and SHA-1 hash code of each chunk are all stored in a section of the *.torrent* file. This information will be used for integrating the chunks after the peer receives the whole file. A tracker will be used to organize all the peers that have the file, while the peers will also be connected to each other. Every time a new peer makes a request to download the file, the tracker will respond with a random list of of the peers who are downloading the same file, so that the downloaders can connect to others. At that time, the seed starts and sends out at least one copy of the file.

Unstructured P2P overlay networks do not have a constant topology. They are commonly organized in a flat, random, or simple hierarchical manner. As shown in Table 2.1, data is duplicated among the system depending on its popularity, and the queries use flooding, random walks or Time-to-Live (TTL). In practice, the unstructured P2P overlay network appears to be more efficient in fetching very popular contents. However, because of its ad hoc nature and flooding-based routing, the correctness and performance of the routing, system scalability and consumption of network bandwidth are all uncertain.

	Gnutella	Freenet	BitTorrent
Decentralization	ultrapeers used to control the routing path	fully distributed, all peers are equal	centralized model with a tracker
Lookup protocol	flooding	key search from peer to peer	tracker
Routing performance	good performance for popular content	key search until it exceeded the Time-to-Live limit	good performance for popular content
Routing state	constant	constant	constant

Table 2.1: Summary of unstructured P2P overlay systems

2.1.2 Structured P2P Overlay System

Structured P2P systems are well organized networks, and they always have a fixed topology. They can also be divided into two classes by their functionality: systems for multi-dimensional range queries and systems for one-dimensional key queries.

Almost all of the one-dimensional key queries are based on the distributed hash table (DHT). DHT hashes the IP addresses of the nodes or the URLs of the data, and maps them with the keys, which is the hashed result, to support the lookup protocol. There are many well known DHT-based systems such like CAN[3, 17], Chord[10], and Pastry[11].

A Content Addressable Network (CAN) represents each node as rectangles on a two-dimensional data space. When a node wants to join, it needs to find an existing node, split the node's data space, and map itself with the extra data space. Every node in CAN gets information from its neighbors for routing. Routing is achieved by recursively going to the node's neighbor until it reaches the destination. If a node leaves the network, a takeover algorithm will be run immediately. A chosen neighbor of the leaving node will take over its space and start a takeover time. This new node will send its new

information to all the neighbors to update their routing tables. Because of the special structure of CAN, CAN can efficiently insert and retrieve contents, so CAN is usually used in large-scale storage management systems such as OceanStore[29], Farsite[30], and Publius[31].

Chord gives each node a key primitively. It uses consistent hashing[13], which may help balance the load of each node, and may reduce the interruptions in the network when nodes join or leave. To distribute the nodes, consistent hashing allocates the nodes in a circle based on their identifier, and uses the hash function to assign keys to the nodes. Each node will have at least one clockwise successor by key, which used for routing. For the routing process in Chord, each lookup query contains the key and nodeID of the destination, and the query is sent node to node by the successors of each until it gets to the destination. To ensure the correctness of routing, Chord uses a stabilization protocol that runs periodically to update the information of the successor of each node. Chord has also been used in many applications, such as Cooperative mirroring or cooperative file system (CFS)[32] and Chord-based domain name system (DNS)[33].

Pastry is a scalable distributed network system which is similar to Chord. The difference is that the routing table in each node in the Pastry network is dynamically designed based on the routing. There may be an external program to determine the routing metric based on the destination node. This helps the Pastry network to reduce the routing cost by avoiding flooding.

In structured P2P overlay systems, all the peers in the identifier space are well organized in a certain topology. As shown in Table 2.2, each peer is assigned to a key by using a hash function. This kind of structure ensures that the systems support efficient searches by using a given key. On the other hand, it also shows they do not support complex queries.

	CAN	Chord	Pastry
Decentralization	DHT-based	DHT-based	DHT-based
Lookup protocol	use uniform hash function	match key and nodeID	match key and prefix in nodeID
Routing performance	$O(d.N^{1/d})$	$O(logN)$	$O(log_B N)$
Routing state	$2d$	$logN$	$2Blog_B N$

N - number of nodes, d - dimensionality of data space, b - number of bits (B = 2b) used for the base of the chosen ID

Table 2.2: Summary of structured P2P overlay systems

2.1.3 Summary

A key query is a special case in the information query, where the request query just asks for a single value to return in a single dimensional data space. But in real life, things are more complex. Usually the information we want to get is more than a single data, but a set of data, or a range of data. The data space can also be multi-dimension instead of one-dimension. As we can see, the above structures cannot support complex queries efficiently. The next section discusses some existing systems designed for complex queries, such as multi-dimensional range queries, the features of their work, and the problems they face.

2.2 Multi-dimensional Range Query

Multi-dimensional range queries have been widely used in P2P networks, such as grid computing, Publish/Subscribe systems, multi-player games, group communication, P2P data sharing, and global storage.

Although the DHT-based system works well on the key queries, it may not be suitable for the range queries, since DHT-based systems got the keys by hashing. The nodes who have the serial IP addresses may not have their keys in range, which makes the search for a

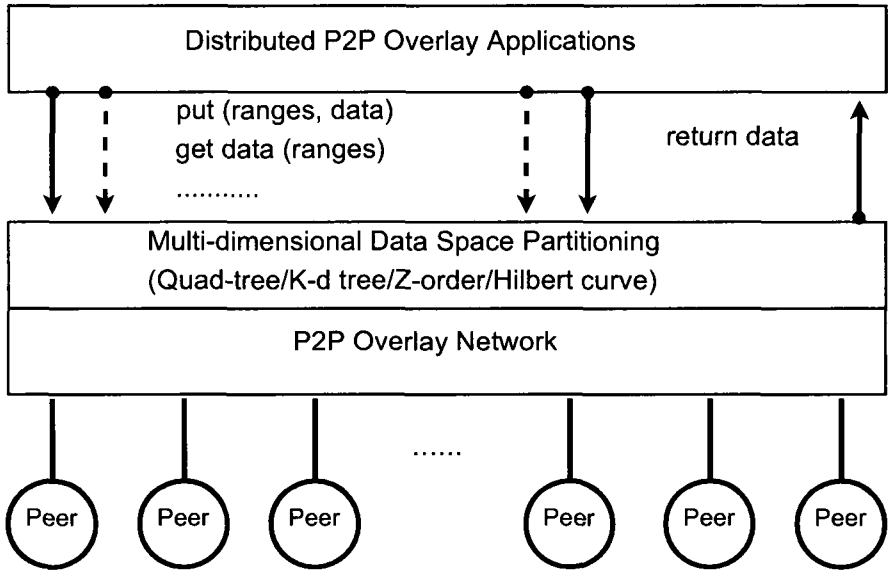


Figure 2.2: Overview of multi-dimensional range query system

range query hard to achieve. It is also hard to present the multi-dimensional data by one-dimensional keys. To solve these problems, systems for multi-dimensional range queries have been designed, such as SONAR[1], SCARP[5], MURK[5], ZNet[28], Skipindex[27], and Squid[37]. The basic structure of the systems for a multi-dimensional range query is shown in Figure 2.4. The systems for multi-dimensional range queries can all be divided into two parts. One part is a data structure for data space partitioning, which is used to partition the multi-dimensional data space and map the multi-dimensional data down to a single dimension. The other part is an identifier management system to coordinate each data to an identifier without changing the locality of the data, and manage the topology of the identifier system.

2.2.1 SONAR

SONAR is a distributed structure for multi-dimensional range queries. It directly maps the multi-dimensional data space to the multi-dimensional identifier space, and is designed for P2P overlay networks, such as CAN.

The routing table of each node in SONAR system contains only the neighbors of this node, with the size of $logN$ (N is the total number of nodes in the structure). Routing uses the greedy algorithm, the node moves to its successor which has the minimal distance to the target, and does this recursively until it gets to the target. SONAR uses k-d tree[2, 19] for maintenance of its topology. When a node wants to join the system, there will be two steps: First it will select a random node, then split a key space of the node, and allocate the free key space to the new node. When a node wants to leave, it not only needs to find another node to fill the free space created by the leaving node, but also needs to consider the load balancing of the system.

The routing table contains only the information of its neighbor nodes, so the updating of the routing table is small and easy to process. SONAR is designed for the range query, so it is straightforward to match the identifiers to the data objects, especially when the data objects have multi-dimensional range space. The maintenance is the main problem of SONAR, since it is always required to maintain the topology of the k-d tree whenever a node joins or leaves. It should also need to keep updating the routing table. Since the routing of SONAR uses the greedy algorithm, if the routing table has not been updated, and the successor has changed, then the chosen route may not be the best or correct way. This kind of routing requires that each node has the latest information, so every node has a stabilizer to maintain the updating of its routing table.

2.2.2 SCARP

SCARP uses the space-filling curve (SFC) to support multi-dimensional queries. It reduces the multi-dimensional data space into one-dimension, and then partitions the data by its range. The routing, after mapping the data space into one-dimensional query, uses skip graph. For SCARP, the load balance is easy to maintain for its one-dimensional query. However as the dimensions of the data space increase, the performance of this data structure will be reduced.

The routing in SCARP can be divided into three steps. First, it converts the multi-

dimensional request query into a set of one-dimensional range queries. Then, it will route each one-dimensional range query. This is performed by some well-known algorithms with query mapping for the space filling curve. In order to reduce the size of the results, SCARP uses "false positives[36]" for each set of results, which may result in sending the requests to non relevant peers. Finally, SCARP uses the routing network skip graph[4, 34].

Skip graph is generated from the skip list[35] and also provides the functionality of a balanced tree. It uses the circular linked list to store the nodes, which can be directly mapped to the data objects. The skip graph is a self-stabilization structure, the deletion and addition of dynamic nodes can be done in $O(logN)$ time, since it is very straightforward when a node wants to join or leave, it only need to tell its neighbors. Skip graph also can tolerate the failure of part of the nodes, because there is more than one routing path between any two nodes. But according to [4], the reliability of the skip graph is still in consideration.

2.2.3 MURK

MURK is very similar to the existing P2P network data structure CAN, which uses rectangles to partition the data space. Each rectangle identifier maps to one node. It uses the k-d tree to manage the mapping and partitioning, and the routing of MURK is straightforward. First, it interconnects nodes in the identifier space, which results in a grid-like structure similar to CAN. After receiving the request in a multi-dimensional range query, the node sends the query to the relevant nodes based on the greedy routing protocol. There are two problems with the original routing in MURK. One is the unbalanced routing load for each node, since every grid may have a different number of neighbors. The second is that the routing algorithm in MURK cannot improve the routing performance effectively, especially in low dimensionality. The routing in MURK has been optimized by using random skip pointers used in CAN, or the skip pointers used in skip graph, so that every node in MURK contains a tale of its skip pointers to some other nodes,

while it still keeps the greedy routing protocol. In the optimized routing, each node will forward the request query to its nearest neighbor, who can be the grid neighbor, or the neighbor pointed by its skip pointer. MURK is good at multi-dimensional data space, but is not suitable for dynamic nodes, since it also needs to maintain the structure of the k-d tree when the nodes are changed.

2.2.4 ZNet

ZNet uses the Z-order[20, 21] as its partition algorithm, and uses the skip graph to manage the overlay network structure. Skip graph generates several skip lists at each partition level, which goes from bottom to top, so the bottom level contains all the nodes in the network. Each node in ZNet maintains its neighbors' information in a routing table. Every entry in the routing table has the information of the neighbor's identities, level number and its neighbors' converage range. The maximum size of the routing table is $O(logN)$ (N is the numer of nodes). In the original method of routing, when the request range query is sent to a node, it will first convert the request into a set of continuous Z-addresses covered by the range query. Each query in the set will then be sent to the node who has the minimum Z-address of its range. However this kind of routing protocol is proven to be inefficient and can not support a dynamic case.

2.2.5 Skipindex

Skipindex is constituted by a chosen data space partitioning structure and the skip graph. First, the data space partitioning structure is used to partition the multi-dimensional data space hierarchically. Skipindex uses the region tree to describe the partitioning of data space. After dividing the data spaces to amount of regions, each region is assigned to a one-dimensional key, in order to get a total order in the whole region. The assigned key captures the hierarchical manner while partitioning. Then, the skip graph is used to store the all the leaf regions of the region tree by using their keys. As we have mentioned

above, the skip graph supports the insertion and lookup of one-dimensional keys. The skip graph can also preserve the logical locality in the key space. When a data point and a region are given, the system can determine if the region contains the destination point is before or after the current region in the total order. The skip graph organizes the routing between the regions. The routing table of each node owns a region at each level in the skip graph contains two parts: the identifiers of other nodes which contain the neighbor regions, and the partitioning history of the identifiers of other nodes which contain the neighbor regions. On average, there are $O(logN)$ levels in skip graph, so the routing performance should be in $O(logN)$.

2.2.6 Squid

The architecture of Squid is based on a data lookup system, it uses the Internet-scale distributed hash table (DHT) to map the data elements to the identifier space. In order to support more complex queries, Squid uses the space-filling curve to reduce the dimensionality of the data space, since the SFCs, like: Hilbert curve[22] or Z-order curve[20, 21], are recursive, self-similar, and locality-preserving. After mapping the data elements to the identifier space, Squid uses Chord, as we mentioned above, to construct the identifier space. Squid uses the following steps to publish a data element.

1. Attach keywords which describe the data elements content to each data element.
2. Use the SFC to partition the data space, and construct the data elements' index.
3. Store the elements at the appropriate node in the overlay structure Chord by using the index of each element.

The index of each node in Squid can identify which range of the data space the node belongs to. Therefore, when a query is requested, Squid checks the range information in the query first, then locates the query to the nearest node, and then routes using the Chord structure. The routing complexity for Squid is the same as the Chord, which is $O(logN)$.

	Space Partitioning	Application Constraint	P2P Overlay Structure	System Constraint
SCARP	Z-order/Hilbert curve	Yes	Skip graph	No
MURK	K-d tree	Yes	CAN	No
ZNet	Z-order curve	Yes	Skip graph	No
Skipindex	K-d tree/R-tree	Yes	Skip graph	No
Squid	Hilbert curve	Yes	CHORD	No
SONAR	K-d tree	Yes	CAN	No

Table 2.3: Comparison of P2P Overlay System

2.2.7 Summary

All the P2P overlay systems we mentioned above can be applied to complex queries, but since they are all originally designed for exact key queries, they can not support multidimensional range query efficiently. Based on the proof of the third party, we can get the comparison results of the existed systems, as shown in the Table 2.3. Multidimensional range queries have brought two important requirements to P2P overlay systems. One is application constraint, which is how to partition and mapping the data space into the identifier space of the distributed system, so that we can easily find the identifier according to the corresponding data object, and do the opposite way the same. The other one is system constraint, which is how to organize and manage the identifiers of the distributed system. This requires the system to preserve the data object location, and easily maintain the topology of the system.

In order to support application constraint, there are some existing data structures, such as Quadtree[18, 19], K-d tree[2, 19], Z-order[20, 21], and Hilbert curve[22], which have been proven to meet the conditions of application constraint.

Quadtree was named by Raphael Finkel and J. L. Bentley in 1974. It is a data structure which describes a class of hierarchical data spaces divided from a two dimensional

data space. It partitions one region recursively into up to four equal subregions. Each subregion finally contains one single type of data. The insertion of a new node in the quadtree is almost the same as the binary tree: compare and choose to go down from root to the right subtree, and insert the node as the leaf. The average time for this is $O(logN)$. To perform the deletion of quadtree is very difficult, because when a node leaves, it has to merge the rest of the nodes in the subtree, but this is not easy. In fact, it seems better to perform a new tree, and insert these nodes into it. Quadtree is efficient tree structure for data storage, search and insertion, ut the deletion and merge of the node in the tree can be difficult.

K-d tree is a data structure for multi-dimensional data space partitioning and mapping. k-d tree uses the binary tree to store the nodes, and each represents a record of the data. Every non-leaf node generates a hyperplane to split the space into subspace. The nodes on each level of k-d tree are generated based on one corresponding dimension of the data space. The insertion of k-d tree is like the insertion of any other search tree. Go through the tree, find the insert location, and add the node as the left or right leaf node, which costs at most $O(logN)$. The main procedure of deletion of k-d tree is to find a node to replace the deleting node, which also costs $O(logN)$ on average. The storage for the multi-dimensional data is small for k-d tree, and it is very efficient and straightforward for the insertion and deletion of the nodes. However, the maintenance to keep the balance of the tree would cost more.

Z-order curve and Hilbert curve are two approaches usually used in data structures for mapping from multi-dimensional data space to one-dimensional space. For Z-order, the space is recursively split into four parts, and a Z-order curve links the quadrants together into a linear order. Each quadrant has a one dimensional Z-value, which makes it easy to find. The Hilbert curve is a very similar approach, but it uses the Hilbert curve instead of the Z-order curve. This makes it more popular than Z-order, because it has better locality-preserving behavior. Z-order curves and Hilbert curves are often used in other one-dimensional data structures after they convert the multi-dimensional data

space into one-dimensional or multi-dimensional data space, like quadtree, which can be more efficient after using these curves.

Although the above data structures use different function to partition and map, Quadtree and K-d tree use the tree structure to manage the nodes, while Z-order curve and Hilbert curve use the space filling curve. We have found that they have the same principle: they all use recursive decomposition to do the partition.

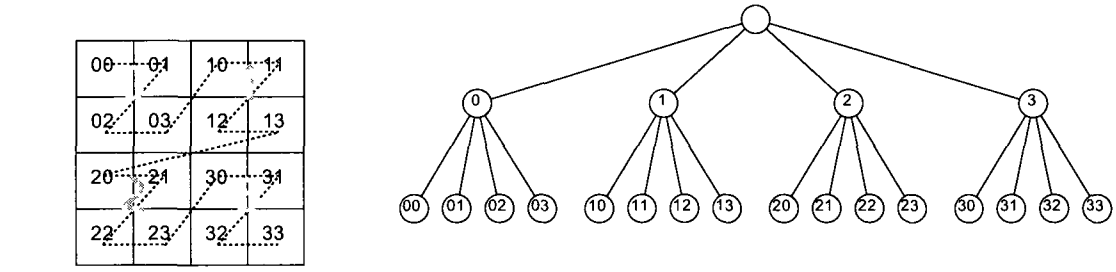
Chapter 3

HD Tree

3.1 Hierarchically Distributed Tree

In Chapter 2, we introduced the previous work in P2P overlay networks. Since there is no existing P2P overlay system which can support multi-dimensional range queries efficiently, a new data structure was proposed the Hierarchically Distributed Tree[14], called the HD Tree.

It is very difficult for a hierarchical tree structure to survive in a distributed environment, like a P2P network. However, the hierarchical structure has no comparable advantages in presenting the recursively compromised data space. As shown in Chapter 2, the most popular partition algorithms, Quadtree, K-d tree, Z-order, and Hilbert curve, have the same feature in partitioning. They all partition the data space recursively. This kind of partition brought us two obvious problems while dividing the multi-dimensional data space. One problem is how we can manage the data localities while the number of the data items extends exponentially as the partition of the data space goes deeper. The second problem is how we can manage the data while the number of the data items expands exponentially as the dimensionality of the data space goes higher. In order to solve the problems brought by partitioning in application constraint, system constraint of multi-dimensional range queries can be states as: how to organize and manage the



(a)2nd order of Z-order curve partitioning (b)The tree structure to represent the data space

Figure 3.1: Z-order curve and its mapping tree

identifiers in the distributed system with the extending and expanding rate of the data objects, while still preserving the data localities.

Table 2.3 in Chapter 2 shows the third party results for different existing P2P overlay networks show that one of them support system constraint of multi-dimensional range queries efficiently. As shown in Figure 3.1, the hierarchical tree structure, which has been studied for a long time, has the non-comparable advantages of maintaining the data locality preserved from the multi-dimensional space partitioning because of its natural connection with the recursive decomposition scheme adopted. Therefore, we intend to develop a new data structure, which can efficiently support the system constraint of multidimensional range queries, by adapting the hierarchical tree structure to the distributed environment.

3.1.1 Introduction

After studying the existing P2P overlay network systems, we find that if we want to develop a system to support multi-dimensional range queries efficiently, the best structure we could choose is the hierarchical structure. As we know, a hierarchically structured system has difficulty surviving in a distributed environment, like a P2P network. This is why we introduce the new hierarchically distributed structure, HD Tree. The HD Tree is a hybrid structure of the hierarchical structure and the distributed structure. The HD

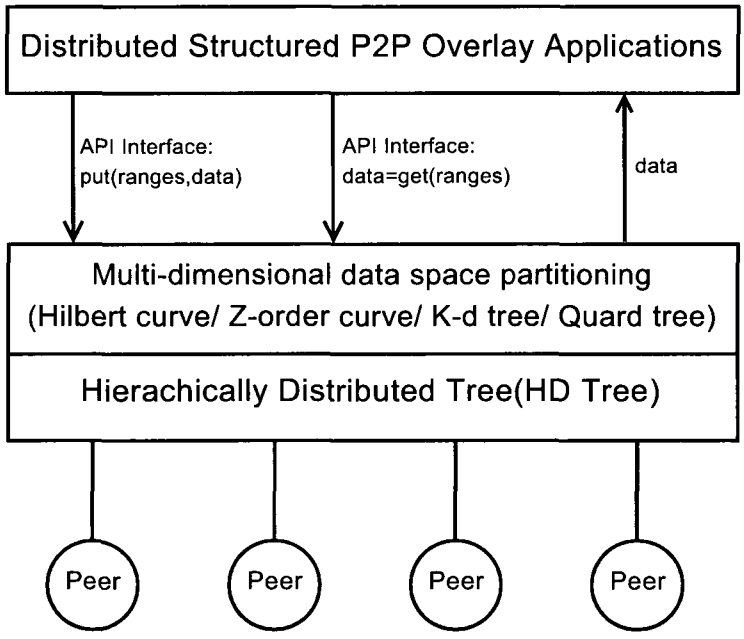


Figure 3.2: The system overview of HD Tree

Tree is built over a complete k-ary tree, which makes it inherit all the basic features of the tree. Besides the hierarchical links of the tree, the HD Tree creates limited number of distributed links, which allows each peer to reach any other peers through not only hierarchical links, but also distributed links.

3.1.2 System Overview

As shown in Figure 3.2, in our application of the HD Tree, we used the existing multi-dimensional data space partition scheme, such as Quadtree, K-d tree, Z-order, and Hilbert curve. Mapping in the HD Tree is very simple, it can be manipulated as direct mapping from the data space to the identifier space, such that each identifier has a nodeID coordinated with a data object. The API interface sends the range query request *put(ranges,data)* to input the data information into the system. The data space partition scheme will then collect and partition the data space into data objects, while still preserving the data objects' locality. When the API interface sends the range query

request *get(ranges)*, the HD Tree system will first find the entry node included in the range, and then all the nodes in the range will be returned based on the entry node.

3.2 Concepts in HD Tree

The HD Tree is a newly developed data structure. In order to better understand it, there are some basic concepts we want show in the following.

3.2.1 Notations in HD Tree

Since the HD Tree structure is built over a complete tree, all the notations in a tree can also be applied here. We used the set (k,h,d,n,n_i) to identify the tree. In the set, k refers to the number of children each parent has, h refers to the height of the tree, d refers to the depth of the current level, n refers to the total node number of the tree, n_i refers to the node number at the depth i.

3.2.2 Concepts in HD Tree

The HD Tree is built by two structures: the hierarchical structure and the distributed structure. The hierarchical structure is a complete k-ary tree structure, so the HD Tree inherits every property of the tree. The distributed structure is built on the hierarchical structure by adding a limited number of distributed links between two hierarchical levels of the tree. By adding the distributed structure, the HD Tree spans each two adjacent sets of the nodes: the set of the nodes at depth i *(*$0 < i \leq h$*)* and the set of the nodes at depth *i-1*. The formal definitions of the HD Tree follows.

HD Tree (as defined in [14]): "the HD Tree is a cyclic graph built over a complete k-ary tree. In addition to the existing links in the complete k-ary tree, each node at depth i - *1 (*$1 < i \leq h$*)* has k or k - *1* extra down links connecting to one node at depth i in each of k sub-trees of the root if that node exists. Each node is able to reach any

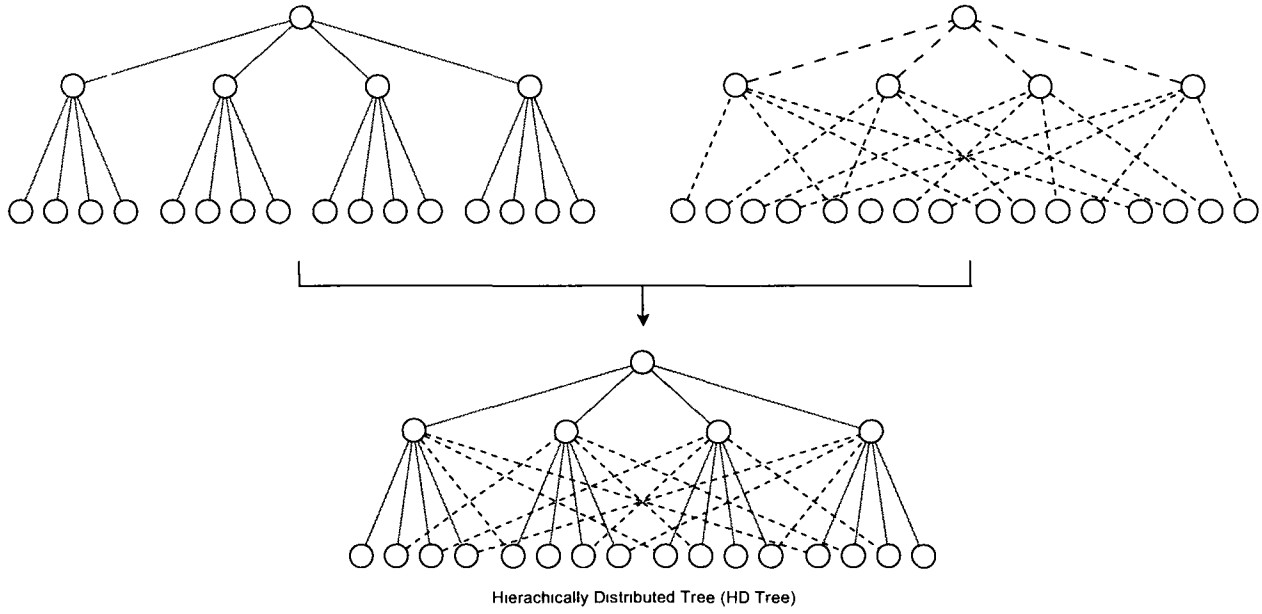


Figure 3.3: A full HD Tree with height = 2

other nodes at the same depth *i - 1* in at most *2(i - 1)* hops via nodes at depth *i - 2* and *i - 1* only."

The construction of the HD Tree is simply coding the nodes sequentially. Because the HD Tree has two different structures, two different coding schemes exist for each of the structures.

There are three rules the coding scheme I of the HD Tree that hierarchical tree has to follow:

- The code for the root is null;
- Each child node takes the code of its parent as the prefix of its own code; and
- All the children of the same parent are coded sequentially.

The result of the coding scheme I is shown in Figure 3.4.

For the coding scheme II of the HD Tree for the distributed tree, the rules are similar as the coding scheme I:

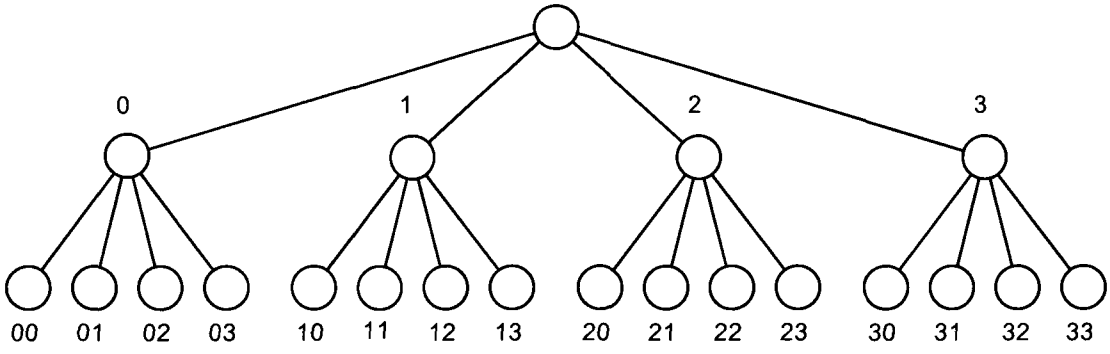


Figure 3.4: Coding scheme I

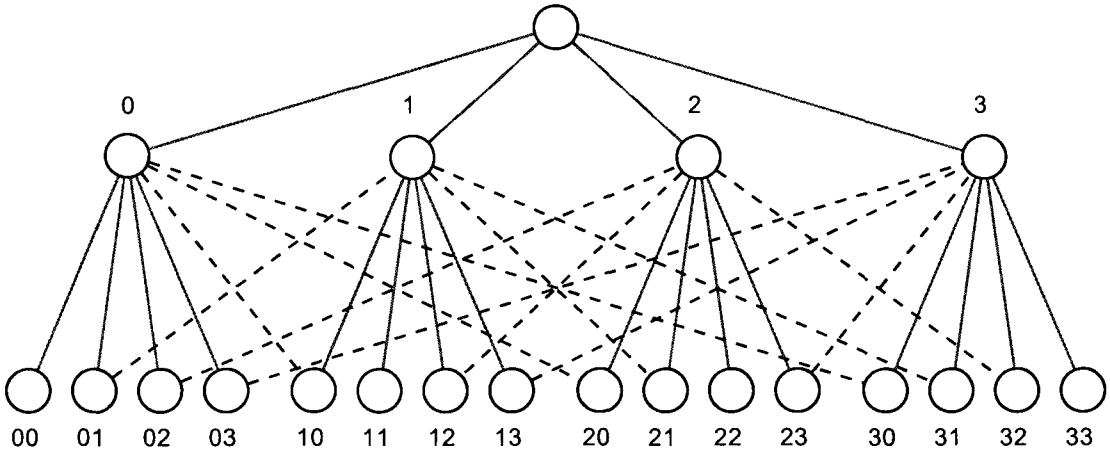


Figure 3.5: Coding scheme II

- The code for the root is null;
- Each child node takes the code of its parent as the suffix of its own code; and
- All the children of the same parent are coded sequentially.

The result of the coding scheme II is shown in Figure 3.5.

HParent: in the HD Tree, the HParent of a node is its original parent in the hierarchical tree structure. As shown in Figure 3.2, the HParents are the parents in the hierarchical tree.

DParent: in the HD Tree, the DParent of a node is the parent connected by the distributed link. As shown in Figure 3.2, the DParents are the parents in the distributed tree.

HChild: in the HD Tree, the child connected by the hierarchical link with its parent is called the HChild. For example, in Figure 3.2, the children connected by the solid lines in the HD Tree are the HChildren of their HParents.

DChild: in the HD Tree, the child connected by the distributed link with its parent is called the DChild. For example, in Figure 3.2, the children connected by the dotted lines in the HD Tree are the DChildren of their DParents.

HD Table (as defined in [14]): "the HD table consists of a H table and a D table. The H table is above the D table. Each element in the H table has a unique entry in the D table. All elements in the HD table are at the same depth in its HD tree. All elements in the same column are siblings, and each column corresponds to a unique parent node such that both coding scheme I and coding scheme II can be applied."

Figure 3.6 shows examples of H tables and D tables of the previous two coding scheme examples in Figure 3.4 and Figure 3.5, and the HD table after combining the H table and D table.

The nodes in the same column of the HD table are siblings. The prefix of the HSiblings in the same column of the H table is the code of their HParent, while the suffix of the DSiblings in the same column of the D table is the code of their DParent, as shown in Figure 3.7. Correspondingly, these HSiblings and DSiblings are the HChildren and DChildren of the parent of that column respectively. The HD table of depth i contains the structure for both the depth i and i-1. Each node in the H table can also be found in the D table. Therefore, the HD table can be also seen as the map between the hierarchical structure and distributed structure of the HD Tree.

HCode and DCode: in the HD Table, every node has two locations, one in the H table and one in the D table. The HCode and DCode are used to represent the two locations.

0	00	10	20	30
1	01	11	21	31
2	02	12	22	32
3	03	13	23	33

(a) H Table at depth 1 (b) H Table at depth 2

0	00	01	02	03
1	10	11	12	13
2	20	21	22	23
3	30	31	32	33

(c) D Table at depth 1 (d) D Table at depth 2

00	00	10	20	30
01	01	11	21	31
02	02	12	22	32
03	03	13	23	33

(e) HD Table at depth 1

00	00	01	02	03
10	10	11	12	13
20	20	21	22	23
30	30	31	32	33

(f) HD Table at depth 2

Figure 3.6: Examples of HD Table

As shown in Figure 3.7, the node with the HCode 131 is the HChild of its HParent 13, it can also be the DChild of its DParent 31. In this way, it maps the HChild of 31 with the HCode 311, which is the DCode of the node whose HCode is 131. In general, the DCode of a node, which is the ith HChild of its HParent, is the HCode of the ith HChild of its DParent.

HPeer and DPeer: in the HD Tree, a node at depth i has a DPeer, which can be referred to any other node at the same depth. It also has a HPeer, which is referred to a node at the depth *i-1*. Therefore, a DPeer could be a HChild, a DChild, a HSibling or a DSibling. An HPeer could be either a HParent or a DParent.

SPeer: in the HD Tree, an SPeer is referred to a special set of nodes, which maps to itself. In other words, an SPeer has the same HCode and DCode, and also has same HParent and DParent. As shown in Figure 3.7, the nodes in bold are SPeers.

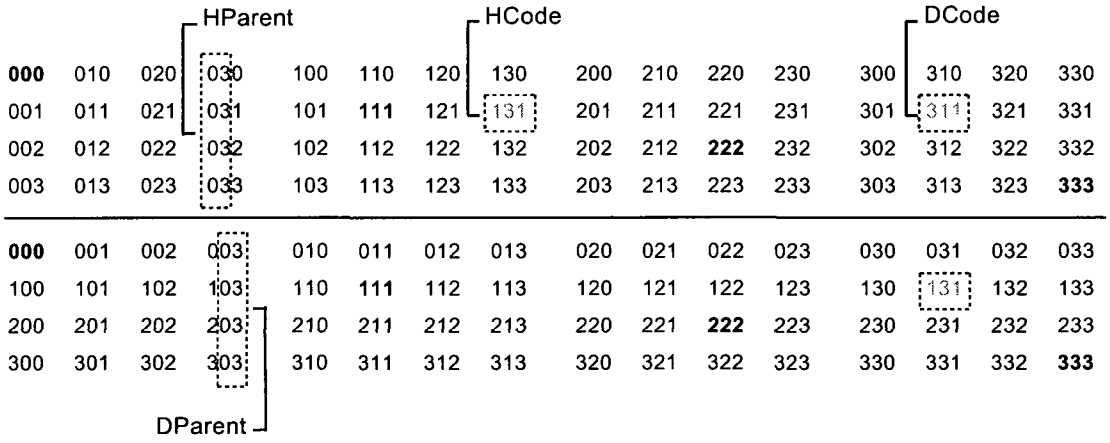


Figure 3.7: The example of HD table at depth 3

3.3 Basic Computations in HD Tree

According to the definitions of HCode and DCode, two basic computations are found in the HD Tree: One is the computation from the HCode to DCode, and the other is the computation from DCode to HCode.

- HCode to DCode

  Given the HCode of a node at depth d in the HD Tree: $C_H = c_0c_1c_2...c_{d-1}$, we can get the DCode of the node is: $C_D = c_1c_2...c_{d-1}c_0$. In general, the way to transfer HCode to DCode in the HD Tree is simply to get the first digit of the HCode, and move it to the end.

- DCode to HCode

  Given the DCode of a node at depth d in the HD Tree: $C_D = c_0c_1c_2...c_{d-1}$, we can get the HCode of the node is: $C_H = c_{d-1}c_0c_1...c_{d-2}$. In general, the way to transfer DCode to HCode in the HD Tree is simply to get the last digit of the DCode, and move it to the front.

3.4 Fault Tolerance and Load Balancing

In the HD Tree, every node has two parents. This ensures that there is more than one route between every two nodes in the HD Tree. When one parent of a node fails, the node can still send the request query through the other parent. In the multi-dimensional data space, the data elements are sparse, instead of uniform. When we use the space-filling curve or other space-partitioning data structure to divide and map the data space, the identifier space will also be sparse, since the partitioning can best preserve the locality of each data element. Even though we have used the hierarchical tree structure, the distributed structure is also added in the HD Tree, so the HD Tree can balance the work load between the nodes using hierarchical structure and distributed structure.

3.5 Supporting Multi-dimensional Range Query

As we have mentioned, the space-filling curve or other space-partitioning data structure to divide and map the data space can best preserve the data localities. Moreover, the HD Tree structure is based on the hierarchical tree structure, which is the most appropriate structure for the recursive decomposition of the partition structures used. The tree structure allows the data elements close to each other to also be near each other in the identifier space. When a range query is requested, it is very easy for parents to broadcast the query to their children in the range.

Chapter 4

Basic Routing Algorithms in HD Tree

In accordance with the hierarchical and distributed structures in an HD Tree, there are two different types of routes. The hierarchical route is the route between any two nodes via intermediate nodes along the hierarchical structure of the HD Tree. Meanwhile, the distributed route is the route between two nodes at the same depth i via intermediate nodes at the depth $i - 1$ and i. The reason to create the distributed route is that the nodes in the range query request is always at the same depth, so the distributed routings can help us to get a easy way to find all the nodes in the range. Therefore, the sequence of nodes in any distributed route has the pattern of "-parent-child-" repeatedly occuring right after the source node, like

source - parent - child - parent - child - ... - parent - child (destination)

Both hierarchical routing and distributed routing in the HD Tree are distributive algorithms. However, hierarchical routing directs the request along the hierarchical route, while, distributed routing directs the request via the distributed route.

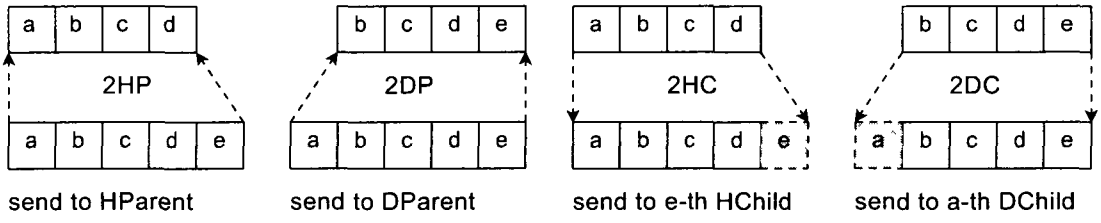


Figure 4.1: Basic Operations in Hierachical Routing

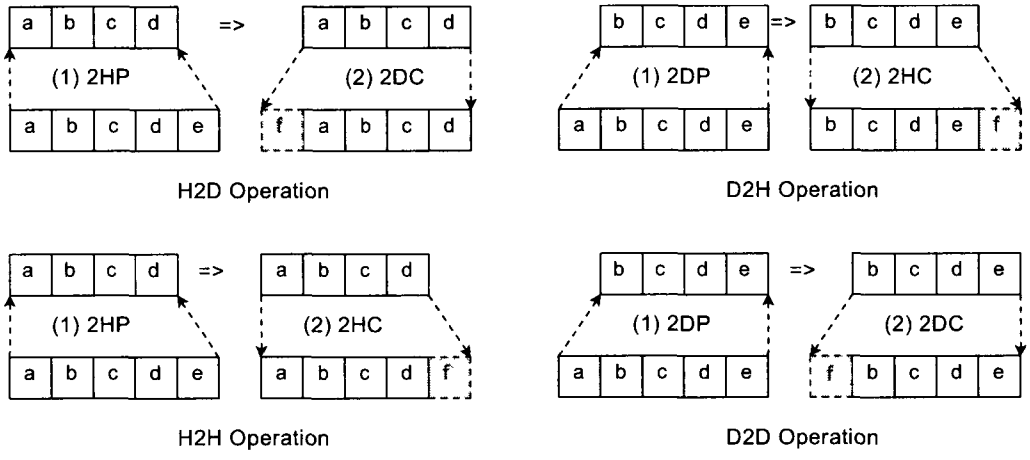


Figure 4.2: Basic Operations in Distributed Routing

4.1 Basic Operations in HD Tree

Basic operations in hierarchical routing are well-known tree operations including "send to parent" and "send to child", with only one-top time complexity. However, in the HD Tree, a node has more options. "Send to HParent" (2HP) or "Send to DParent" (2DP) truncates one digit in the current HCode at the right most side or the left most side. On the other hand, "Send to HChild" (2HC) or "Send to DChild" (2DC) adds one more digit to the current HCode at the right most side or the left most side. The value of new digit is determined by the sequence number of the HChild or DChild chosen. These four basic hierarchical operations are shown in Figure 4.1.

Different from what we have seen in hierarchical routing, the basic operations in

Algorithm 4.1 2HP (*DEST, len*)

INITIATOR	**HPARENT**
BEGIN	receiving (*DEST, len*)
send (DEST, len) to HParent;	BEGIN
become non INITIATOR;	*if d > len*
END	*pass (DEST, len) to HParent;*
	else
	CHK_ROUTE();
	END

distributed routing are two-hop time operations which always involve a parent node and a child node, and result in a digit-shift or digit replacement in the current HCode. A digit-shift operation is accomplished by two hierarchical operations, 2DP and 2HC, which can be simplified as D2H, or 2HP and 2DC, which can be simplified as H2D. In the digit-shift operation, a message is passed between an HSibling and a DSibling, while in the digit replacement operation, it is passed between either HSiblings or DSiblings. 2DP and 2DC (D2D), and 2HP and 2HC (H2H) are such two basic distributed operations. Figure 4.2 depicts these four basic operations in distributed routing.

4.2 Hierarchical Routing Operations

Hierarchical routing in the HD Tree is simply a series of repeated operations of some or all of the four basic hierarchical routing operations: 2HP, 2DP, 2HC, and 2DC. We explore the following four hierarchical routing algorithms coordinated with each basic hierarchical routing operation.

The details about the implementations of the 2HP, 2DP, 2HC, and 2DC routing operations are described in the following sections. (Given the HCode of the destination node *DEST*, the length of its HCode *len*, and d is the depth of current node.)

Algorithm 4.1 shows the pseudocode for the implementation of 2HP routing operation.

Algorithm 4.2 2DP (*DEST, len*)

INITIATOR	**DPARENT**
BEGIN	receiving (*DEST, len*)
send (DEST, len) to DParent;	BEGIN
become non INITIATOR;	*if* $d > len$
END	*pass (DEST, len) to DParent;*
	else
	CHK_ROUTE();
	END

Algorithm 4.3 2HC (*DEST, i, len*)

INITIATOR	**HCHILD**
BEGIN	receiving (*DEST, i, len*)
get i-th digit c_i *from DEST;*	BEGIN
send (DEST, $i+1$*, len) to*	*if* $d < len$
c_i*-th HChild;*	*get i-th digit* c_i *from DEST;*
become non INITIATOR;	*pass (DEST,* $i+1$*, len) to*
END	c_i*-th HChild;*
	else
	CHK_ROUTE();
	END

Algorithm 4.4 2DC (*DEST, i, len*)

INITIATOR	**DCHILD**
BEGIN	receiving (*DEST, i, len*)
get i-th digit c_i from DEST;	BEGIN
send (DEST, $i-1$, len) to	*if $d < len$*
c_i-th DChild;	*get i-th digit c_i from DEST;*
become non INITIATOR;	*pass (DEST, $i-1$, len)*
END	*to c_i-th DChild;*
	else
	CHK_ROUTE();
	END

In 2HP routing, the message contains the *DEST* and *len*. When a node receives the message from its HChild, it will first check if its depth is larger than *len*. If it is true, it will forward the message to its HParent. If not, the routing will stop, and a *CHK_ROUTE()* function will be started to check the mistakes in the routing.

Similarly, we also give the pseudocode for the implementation of 2DP routing operation in Algorithm 4.2. The process of the 2DP route is almost the same as 2HP, but instead of sending the message to the HParent, in 2DP routing a node will always send the message to its DParent.

The pseudocode for the implementation of 2HC routing operation is shown in Algorithm 4.3. The message in 2HC routing contains three parts: the *DEST*, the *len*, and a pointer pointed to current digit of the *DEST*, which starts from the start digit of the *DEST*. When a node receives a message, it will check if its depth is smaller than the *len*. If it is true, it will forward the message to its coordinate HChild based on the current digit of the *DEST*, and move the pointer to the next digit of the *DEST*. If not the routing will stop, and a *CHK_ROUTE()* function will be started to check the mistakes in the routing.

Similar to 2HC routing, the pseudocode for the implementation of 2DC routing op-

eration is given in Algorithm 4.4. The difference is that the pointer starts from the end digit of the *DEST*, so if the depth of the current node is smaller than the *len*, it will move the pointer to the last digit of the *DEST*. Moreover, instead of sending the message to the HChild, in 2DC routing a node will always send the message to one of its DChildren.

4.3 Distributed Routing Algorithms

Distributed routing is essentially a series of digit operations that are continuously applied in one direction. If each new digit is chosen properly, by following every digit in the HCode of the destination node sequentially, the destination node is guaranteed to be reached in at most d steps of 2-hop time operations. This is the basic principle that what blind distributed routing algorithm (BLINDDR)[14] is based on.

There are four types of distributed routing algorithms in the HD Tree. Each can be applied to one of four matching patterns, which are described in detail in the following sections. D2H routing and H2D routing are two series of digit-shift operations towards the destination code. One consists of only D2H operations, and shifts the current HCode towards the left; and the other consists of only H2D operations, and shifts the current HCode towards the right. In addition, in the last step of D2H routing and H2D routing, if we apply D2D and H2H instead D2H and H2D operations respectively, we obtain the other two distributed routing algorithms, D2D routing and H2H routing, which are named based on their last step digit operations.

4.3.1 Matching Pattern

The BLINDDR is only able to deliver the routing request to the destination with the worst case time complexity, because BLINDDR generates all d digits of destination code without making use of any part of the HCode of the current node. However, as we described previously, distributed routing in the HD Tree is a series of digit-shift operations, which are sequentially applied to either the right most or the left most side

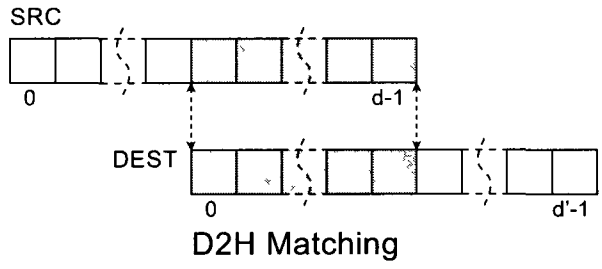


Figure 4.3: D2H Matching Pattern

of the current HCode towards the destination HCode. Optimization may be achieved by comparing HCodes between the source and the destination nodes. If matching can be found, some digit operations might be saved, which means that the destination could be reached via a shorter path. Basically, there are four matching patterns that can be used for this purpose, namely, D2H matching, H2D matching, D2D matching, and H2H matching. These matching patterns correspond to each of the four distributed routing algorithms that we have described.

The details about D2H, H2D, D2D, and H2H routing algorithms with matching patterns are described in the following sections. (*DEST* denotes the HCode of a destination node, *SRC* denotes the HCode of a source node.)

4.3.2 D2H Routing Algorithm

D2H routing starts from the HSibling. The message in the route contains two parts: the *DEST* and i-th number of the current digit it points to in the *DEST*, which starts from the first digit of the *DEST*. When a node receives a message, it first checks if it is the destination node. If the current node is the destination node, the route will stop. If the current node is not the destination node, it will send the message to its DParent. When the DParent receives the message, it will increase the number i in the message, and then forward the message to its appropriate HChild based on the i-th digit of *DEST*. The HChild who receives the message will recursively complete the whole check and forward

000	010	020	030	100	110	120	130	200	210	220	230	300	310	320	330
001	011	021	031	101	111	121	131	201	211	221	231	301	311	321	331
002	012	022	032	102	112	122	132	202	212	222	232	302	312	322	332
003	013	023	033	103	113	123	133	203	213	223	233	303	313	323	333

000	001	002	003	010	011	012	013	020	021	022	023	030	031	032	033
100	101	102	103	110	111	112	113	120	121	122	123	130	131	132	133
200	201	202	203	210	211	212	213	220	221	222	223	230	231	232	233
300	301	302	303	310	311	312	313	320	321	322	323	330	331	332	333

(SRC) 012 - 12 - 121 - 21 - 212 - 12 - 123 (DEST)

Figure 4.4: D2H Routing without Matching

000	010	020	030	100	110	120	130	200	210	220	230	300	310	320	330
001	011	021	031	101	111	121	131	201	211	221	231	301	311	321	331
002	012	022	032	102	112	122	132	202	212	222	232	302	312	322	332
003	013	023	033	103	113	123	133	203	213	223	233	303	313	323	333

000	001	002	003	010	011	012	013	020	021	022	023	030	031	032	033
100	101	102	103	110	111	112	113	120	121	122	123	130	131	132	133
200	201	202	203	210	211	212	213	220	221	222	223	230	231	232	233
300	301	302	303	310	311	312	313	320	321	322	323	330	331	332	333

(SRC) 012 - 12 - 123 (DEST)

Figure 4.5: D2H Routing with Matching

steps, until the message gets to the destination node. The pseudocode for the D2H algorithm is shown in Algorithm 4.5.

As shown in Figure 4.3, D2H matching compares the digits at the end of the *SRC* to the same number of digits at the beginning of the *DEST*.

Figure 4.4 shows the complete process of D2H routing without matching in the HD Table. However, if the source HCode is "*abcde*" and the destination HCode is "*defgh*" (D2H matching), the example in Figure 4.4 can be changed as Figure 4.5 shows. We have 2-digit-shift operations that can be saved in D2H routing:

(SRC) abcde - bcde - bcdef - cdef - cdefg - defg - defgh (DEST)

Algorithm 4.5 D2H (*DEST, i*)

INITIATOR	**DPARENT**	**HCHILD**
BEGIN	receiving (*DEST, i*)	receiving (*DEST, i*)
send (DEST, i) to DParent;	BEGIN	BEGIN
become non INITIATOR;	*get i-th digit c_i from DEST;*	*if $i < d$*
END	*pass (DEST, $i + 1$) to c_i-th HChild;*	*pass (DEST, i) to DParent;*
	END	*else*
		CHK_ROUTE();
		END

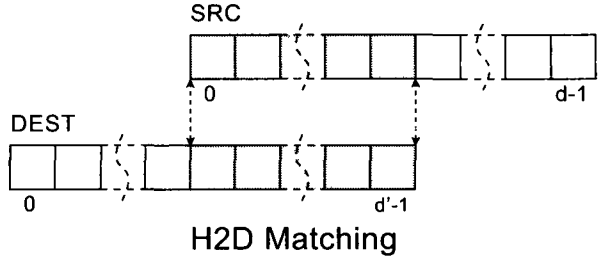


Figure 4.6: H2D Matching Pattern

4.3.3 H2D Routing Algorithm

Different from D2H routing, H2D routing starts from the DSibling. The message in the route also contains two parts; the *DEST* and i-th number of the current digit it points to in the *DEST*, which starts from the end digit of the *DEST*. When a node receives a message, it first checks if it is the destination node. If the current node is the destination node, the route will stop. If the current node is not the destination node, it will send the message to its HParent. When the HParent receives the message, it will decrease the number i in the message, and then forward the message to its appropriate DChild based on the i-th digit of the *DEST*. The DChild who receives the message will recursively complete the whole check and forward steps, until the message gets to the destination node. The pseudocode for the H2D algorithm is shown in Algorithm 4.6.

Algorithm 4.6 H2D (*DEST, i*)

INITIATOR	**HPARENT**	**DCHILD**
BEGIN	receiving (*DEST, i*)	receiving (*DEST, i*)
send (DEST, i) to HParent;	BEGIN	BEGIN
become non INITIATOR;	*get i-th digit c_i from DEST;*	*if $i \geq 0$*
END	*pass (DEST, $i-1$) to c_i-th DChild;*	*pass (DEST, i) to HParent;*
	END	*else*
		CHK_ROUTE();
		END

As shown in Figure 4.6, the H2D matching is the reverse of the D2H matching. It compares the digits at the beginning of the *SRC* to the same number of digits at the end of the *DEST*.

Figure 4.7 shows the complete process of H2D routing without matching in the HD Table. However, if the source HCode is "*defgh*" and the destination HCode is "*abcde*" (D2H matching), the example in Figure 4.7 can be changed as shown in Figure 4.8. We have 2-digit-shift operations that can be saved in H2D Routing. This example for H2D routing with H2D matching is the reverse operation of the D2H Routing:

(SRC) defgh - defg - cdefg - cdef - bcdef - bcde - abcde (DEST)

000	010	020	030	100	110	120	130	200	210	220	230	300	310	320	330
001	011	021	031	101	111	121	131	201	211	221	231	301	311	321	331
002	012	022	032	102	112	122	132	202	212	222	232	302	312	322	332
003	013	023	033	103	113	123	133	203	213	223	233	303	313	323	333

000	001	002	003	010	011	012	013	020	021	022	023	030	031	032	033
100	101	102	103	110	111	112	113	120	121	122	123	130	131	132	133
200	201	202	203	210	211	212	213	220	221	222	223	230	231	232	233
300	301	302	303	310	311	312	313	320	321	322	323	330	331	332	333

(SRC) 012 - 01 - 101 - 10 - 010 - 01 - 301 (DEST)

Figure 4.7: H2D Routing without Matching

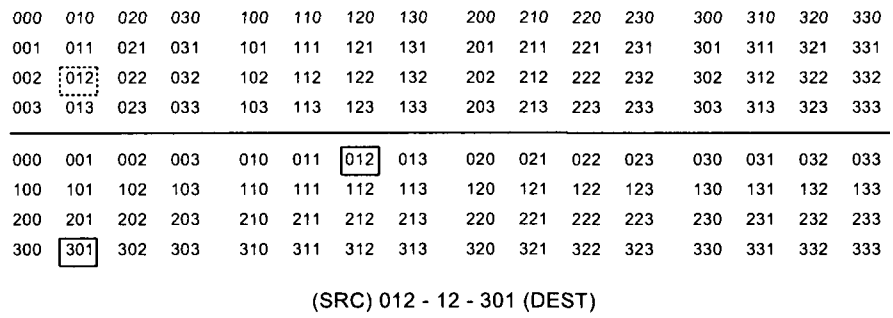


Figure 4.8: H2D Routing with Matching

4.3.4 D2D Routing Algorithm

D2D routing is very similar to D2H routing, as it also starts from the HSibling. The message in the route contains two parts, the *DEST* and i-th number of the current digit it points to in the *DEST*, but the i starts at the second digit from the beginning of the *DEST* instead of the first digit. D2D routing is the same as D2H routing for the most part. When a node receives a message, it first checks if it is the destination node. If the current node is the destination node, the route will stop. If the current node is not the destination node, it will send the message to its DParent. When the DParent receives the message, it will increase the number i in the message and modulo it by the total number of digits of the *DEST*. If i becomes 0, it means the i is currently pointed to the first digit of the *DEST*, the DParent will send the message to its appropriate DChild instead of HChild, and the DChild is curtain of the destination node of the request query. If the i is not *0*, the DParent will do the same thing as in D2H routing, it will forward the message to its appropriate HChild. The HChild who receives the message will recursively complete the whole check and forward steps, until the message gets to the destination node. The pseudocode for the D2D algorithm is shown in Algorithm 4.7.

As shown in Figure 4.9, D2D matching is an extension of D2H matching. D2D matching also compares the digits at the end of the *SRC* to the same number of digits at the beginning of the *DEST*, but without considering the first digit of the *DEST*.

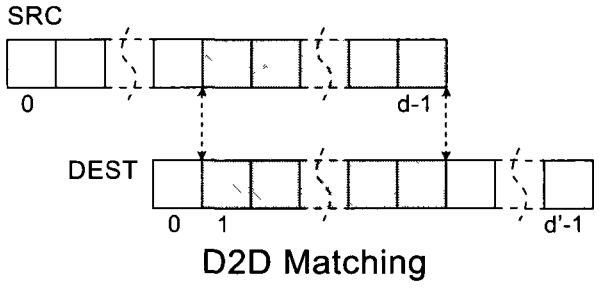


Figure 4.9: D2D Matching Pattern

000	010	020	030	100	110	120	130	200	210	220	230	300	310	320	330
001	011	021	031	101	111	121	131	201	211	221	231	301	311	321	331
002	012	022	032	102	112	122	132	202	212	222	232	302	312	322	332
003	013	023	033	103	113	123	133	203	213	223	233	303	313	323	333
000	001	002	003	010	011	012	013	020	021	022	023	030	031	032	033
100	101	102	103	110	111	112	113	120	121	122	123	130	131	132	133
200	201	202	203	210	211	212	213	220	221	222	223	230	231	232	233
300	301	302	303	310	311	312	313	320	321	322	323	330	331	332	333

(SRC) 012 - 12 - 121 - 21 - 212 - 12 - 312 (DEST)

Figure 4.10: D2D Routing without Matching

000	010	020	030	100	110	120	130	200	210	220	230	300	310	320	330
001	011	021	031	101	111	121	131	201	211	221	231	301	311	321	331
002	012	022	032	102	112	122	132	202	212	222	232	302	312	322	332
003	013	023	033	103	113	123	133	203	213	223	233	303	313	323	333
000	001	002	003	010	011	012	013	020	021	022	023	030	031	032	033
100	101	102	103	110	111	112	113	120	121	122	123	130	131	132	133
200	201	202	203	210	211	212	213	220	221	222	223	230	231	232	233
300	301	302	303	310	311	312	313	320	321	322	323	330	331	332	333

(SRC) 012 - 12 - 312 (DEST)

Figure 4.11: D2D Routing with Matching

Algorithm 4.7 D2D (*DEST, i*)

INITIATOR

BEGIN

i = i MOD d;

send (DEST, i) to DParent;

become non INITIATOR;

END

HCHILD

receiving (*DEST, i*)

BEGIN

pass (DEST, i) to DParent;

END

DPARENT

receiving (*DEST, i*)

BEGIN

get i-th digit c_i from DEST;

i = (i++) MOD d;

if i ≠ 0

pass (DEST, i) to c_i-th HChild;

else

pass (DEST, i) to c_i-th DChild;

END

DCHILD

receiving (*DEST, i*)

BEGIN

CHK_ROUTE();

END

Figure 4.10 shows the complete process of D2D routing without matching in the HD Table. However, if the source HCode is "*abcde*" and the destination HCode is "*hdefg*" (D2H matching), the example in Figure 4.10 can be changed as Figure 4.11 shows. We have 2-digit-shift operations that can be saved in D2D routing:

(SRC) abcde - bcde - bcdef - cdef - cdefg - defg - hdefg (DEST)

4.3.5 H2H Routing Algorithm

H2H routing is very similar to H2D routing, as it starts from the DSibling as well. The message in the route contains two parts, the DEST and i-th number of the current digit

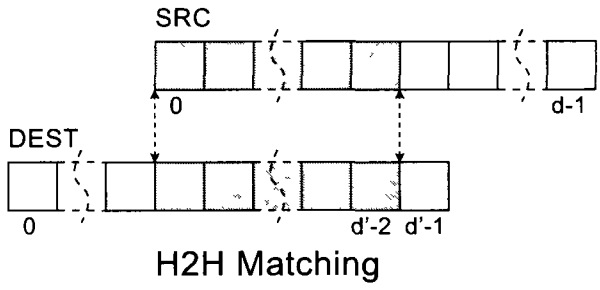


Figure 4.12: H2H Matching Pattern

it points to in the *DEST*, but the i starts from the second last digit of the *DEST* instead of the end digit. H2H routing is the same as the H2D routing in most aspects. When a node receives a message, it first checks if it is the destination node. If the current node is the destination node, the route will stop. If the current node is not the destination node, it will send the message to its HParent. When the HParent receives the message, it will decrease the number i in the message and modulo it by the total number of digits of the *DEST*. If the i becomes *d-1*(d is the depth of current node), it means the i is currently pointed to the last digit of DEST, the HParent will send the message to its appropriate HChild instead of DChild, and the HChild is sure of the destination node of the request query. If the i is not *d-1*, the HParent will do the same thing as in H2D routing, it will forward the message to its appropriate DChild. The DChild who receives the message will recursively complete the check and forward steps, until the message gets to the destination node. The pseudocode for the H2H algorithm is shown in Algorithm 4.8.

As shown in Figure 4.12, H2H matching is an extension of H2D matching. H2H matching also compares the digits at the beginning of the *SRC* to the same number of digits at the end of the *DEST*, but without considering the last digit of the *DEST*.

Figure 4.13 shows the complete process of H2H routing without matching in the HD Table. However, if the source HCode is "*defgh*" and the destination HCode is "*bcdea*" (D2H matching), the example in Figure 4.13 can be changed as Figure 4.14 shows. We

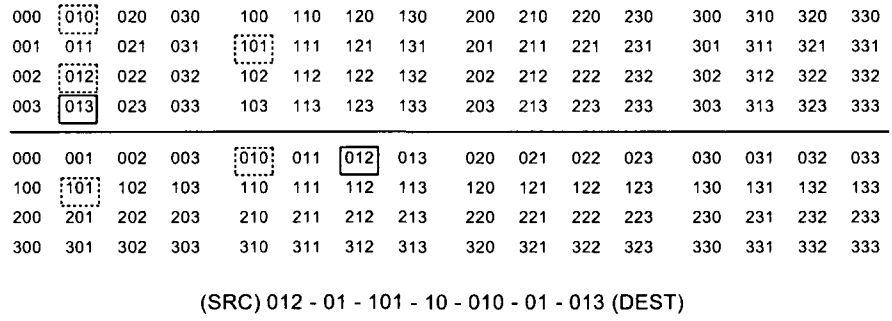


Figure 4.13: H2H Routing without Matching

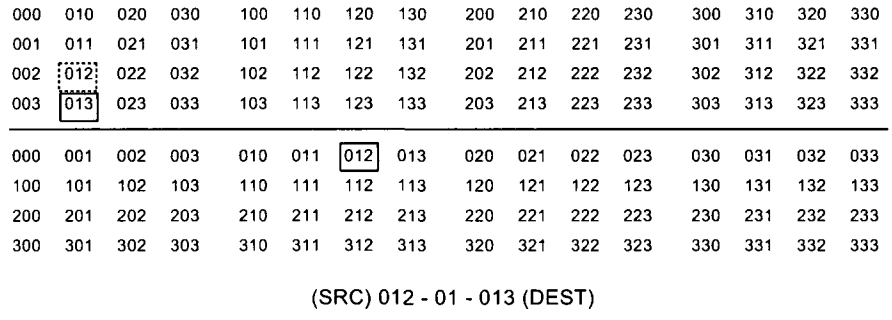


Figure 4.14: H2H Routing with Matching

Algorithm 4.8 H2H (*DEST, i*)

INITIATOR

BEGIN

i = *i MOD d;*

send (DEST, i) to HParent;

become non INITIATOR;

END

DCHILD

receiving (*DEST, i*)

BEGIN

pass (DEST, i) to HParent;

END

HPARENT

receiving (*DEST, i*)

BEGIN

get i-th digit c_i *from DEST;*

$i = (i - -)\ MOD\ d$;

if $i \neq d - 1$

pass (DEST, i) to c_i*-th DChild;*

else

pass (DEST, i) to c_i*-th HChild;*

END

HCHILD

receiving (*DEST, i*)

BEGIN

CHK_ROUTE();

END

have 2-digit-shift operations that can be saved in H2H routing:

(SRC) defgh - defg - cdefg - cdef - bcdef - bcde - bcdea (DEST)

4.4 Summary of Routing Algorithms in the HD Tree

In this section, we will have a brief summary of hierarchical routing algorithms and distributed routing algorithms we have instroduced above.

4.4.1 Hierarchical Routing Algorithms

The four basic hierarchical routing algorithms are very basic routings in the HD Tree. By using the four routings sequentially, any two nodes in the HD Tree can reach each other. Since most of the requests need to go through the root in order to get to the destination node, the hierarchical routing algorithms generate high volume traffic for nodes closer to the root. Moreover, in the hierarchical routing algorithms, there is only one route between any two nodes in the HD Tree. When a node in the route is unavailable, the routing will be suspended. This illustrates that the hierarchical routing algorithms have very limited options to cope with a node failure.

4.4.2 Distributed Routing Algorithms

The four basic distributed routing algorithms we presented above are designed based on the special hierarchically distributed structure in the HD Tree. Obviously, there is more than one route between the source node and destination node in distributed routing. When the following node in a route is unavailable, the message will simply be redirected to another route to get to the destination node. Therefore, distributed routing is fault tolerable. Also, the distributed routing algorithms complete the routing by using only two levels in the HD Tree: the current level of the source and destination nodes and the level of their parents. This can greatly help the HD Tree structure to balance the work load in the nodes at the higher level. On the other hand, distributed routing algorithms can only be applied between two nodes who are at the same depth in the HD Tree. Meanwhile, only the current level of the source and destination nodes and their parent level are used in distributed routings. The distributed routing algorithms can also help to balance the work load of the nodes closer to the root. Since the distributed routing algorithms can only be applied between nodes at the same depth in the HD Tree, they can only be used in limited conditions.

000	010	020	030	00	110	120	130	200	210	220	230	300	310	320	330
001	011	021	031	01	111	121	131	201	211	221	231	301	311	321	331
002	012	022	032	02	112	122	132	202	212	222	232	02	312	322	332
003	013	023	033	03	113	123	133	203	213	223	233	303	313	323	333
000	001	002	003	010	011	012	013	020	021	022	023	030	031	032	033
100	101	102	103	110	111	112	113	120	121	122	123	130	131	132	133
200	201	202	203	210	211	212	213	220	221	222	223	230	231	232	233
300	301	302	303	310	311	312	313	320	321	322	323	330	331	332	333

$J_0 = \{000\}; J_1 = J_0 \cup \{001, 002, 003\}; J_2 = J_1 \cup \{010, 011, 012, 013\} \cup \{020, 021, 022, 023\} \cup \{030, 031, 032, 033\}$

Figure 4.15: D2H When Source is a SPeer

4.5 Performance Analysis

In distributed routing, all siblings of the source node are 2-hops away, all siblings of the siblings of the source are 4-hops aways, and so on. These 2i-hops away sets of siblings might have one of four matching patterns we described previously. By checking and choosing the maximum matching pattern, the shortest path can always be found between the source and destination pair, and the average performance of distributed routing can be evaluated further.

Theorem 1: The average case time complexity of distributed routing:

$$T_{avg}(BasicDR) \leq 2d - 2/(k-1) + 2/k^d(k-1)$$

Proof: Assume SPeer$S$ = 00...00 is the source node at depth d, the destination D is chosen randomly among all nodes at the same depth. In Part 1, we analyze the time complexity based on the D2H algorithm when the source is a SPeer, and then we prove that the SPeer has the worst average case time complexity in Part 2.

Part 1:

Let $J_0 = S$, and J_i be the union of HSibling sets of all members in J_i—1 via DParent

000	010	020	030	100	110	120	130	200	210	220	230	300	310	320	330
001	011	021	031	101	111	121	131	201	211	221	231	301	311	321	331
002	012	022	032	102	112	122	132	202	212	222	232	302	312	322	332
003	013	023	033	103	113	123	133	203	213	223	233	303	313	323	333
000	001	002	003	010	011	012	013	020	021	022	023	030	031	032	033
100	101	102	103	110	111	112	113	**120**	**121**	**122**	**123**	130	131	132	133
200	201	202	203	210	211	212	213	220	221	222	223	230	231	232	233
300	301	302	303	310	311	312	313	320	321	322	323	330	331	332	333

$J_0 = \{012\}; J_1 = J_0 \cup \{210, 211, 212, 213\}; J_2 = J_1 \cup \{200, 201, 202, 203\} \cup \{220, 221, 222, 223\} \cup \{230, 231, 232, 233\}$

Figure 4.16: D2H When Source is a non-SPeer

only,

$\Rightarrow$the probability of $D \subseteq J_0$ is $1/k^d$, it takes 0 hop time.

In D2H, all DPeers that are 2 hops away from S are in $J_1 - J_0$, and $|J_1 - J_0| = k - 1$.

$\Rightarrow$the probability of $D \subseteq J_1 - J_0$ is $(k-1)/k^d$.

In D2H, all DPeers that are 4 hops away from S are in $J_2 - J_1$, and $|J_2 - J_1| = k(k-1)$.

$\Rightarrow$the probability of $D \subseteq J_2 - J_1$ is $k(k-1)/k^d$.

In D2H, all DPeers that are $2i$ hops away from S are in $J_i - J_{i-1}$, and $|J_i - J_{i-1}| = k^{i-1}(k-1)$.

$\Rightarrow$the probability of $D \subseteq J_i - J_{i-1}$ is $k^{i-1}(k-1)/k^d$.

In D2H, all DPeers that are $2d$ hops away from S are in $J_d - J_{d-1}$, and $|J_d - J_{d-1}| = k^{d-1}(k-1)$.

$\Rightarrow$the probability of $D \subseteq J_d - J_{d-1}$ is $k^{d-1}(k-1)/k^d$.

$\Rightarrow T_{avg}(D2H(SPeer)) = \frac{2(k-1)}{k^d+1} \sum_{i=1}^{d} ik^i$.

$\Rightarrow T_{avg}(D2H(SPeer)) = 2d - \frac{2}{k-1} + \frac{2}{k^d(k-1)}$.

Part 2:

If a SPeer is the source node, it has the maximum average time complexity in the D2H algorithm because $J_{i-1} \subset J_i$ is always true for $0 < i < d$, with leads to the worst

average case time complexity compared with any other non-SPeer source nodes at the same depth. Figure 4.5 and 4.6 demonstrate the difference between the SPeer and the non-SPeer in D2H distributed routing.

The analysis for H2D, D2D, and H2H routing returned the same result as the analysis for D2H routing. Therefore, by Part 1 and Part 2

$$T_{avg}(BasicDR) \leq 2d - 2/(k-1) + 2/k^d(k-1)$$

From the analysis of SPeer in this proof, it is not hard to conclude that distributed routing of SPeer has the same routing performance as its equivalent tree routing.

Fact: Distributed routing achieves better routing performance than its equivalent tree routing does.

Chapter 5

Supporting Multi-dimensional Range Query

5.1 Supporting Multi-dimensional Range Query

Multi-dimensional range queries are to find a minimum set of nodes included in a requested range in a multi-dimensional data space. The routing algorithms introduced above can help the HD Tree to better support multi-dimensional range queries. Here the Z-order curve is chosen for the multi-dimensional data space partitioning. As an example, in Figure 5.1 the two-dimensional data space is in 2-order partitioning using the Z-order curve. The range the query requested are shown by the nodes in grey.

Since this thesis has shown previously that the tree structure can best represent the recursive decomposition scheme to preserve the data locality, I would use a tree structure to compare to an HD Tree. Figure 5.2 shows the route for the range query request in a tree structure. If the entry node is node 01, it is easy to get to node 03, since they belongs to one parent node. In order to find the nodes 12, 21 and 30, the node 01 has to send the message back to the root node, then the root node broadcast the message to all its children, and they keeps on forwarding the message down to get to the destination nodes. Finally all the responses to the query are sent back by the same path as the

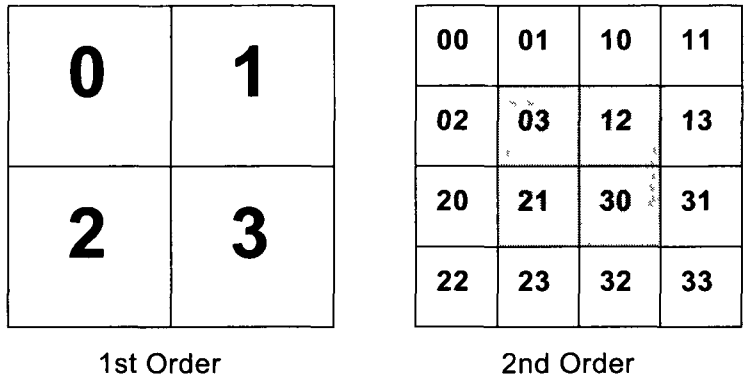


Figure 5.1: Two Iterations of data space partitioning using Z-order curve

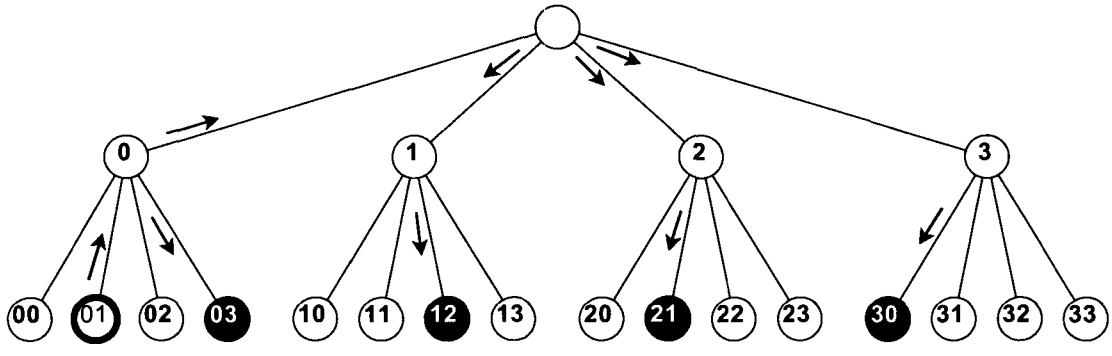
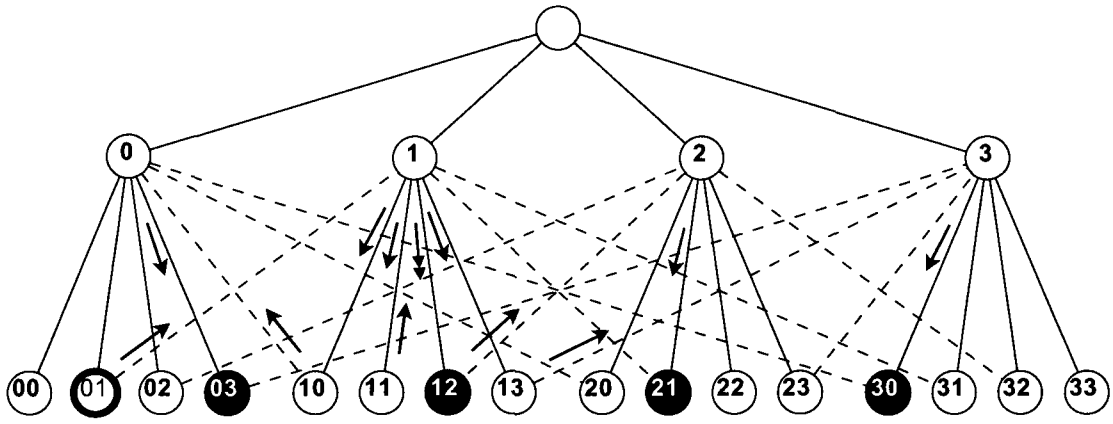


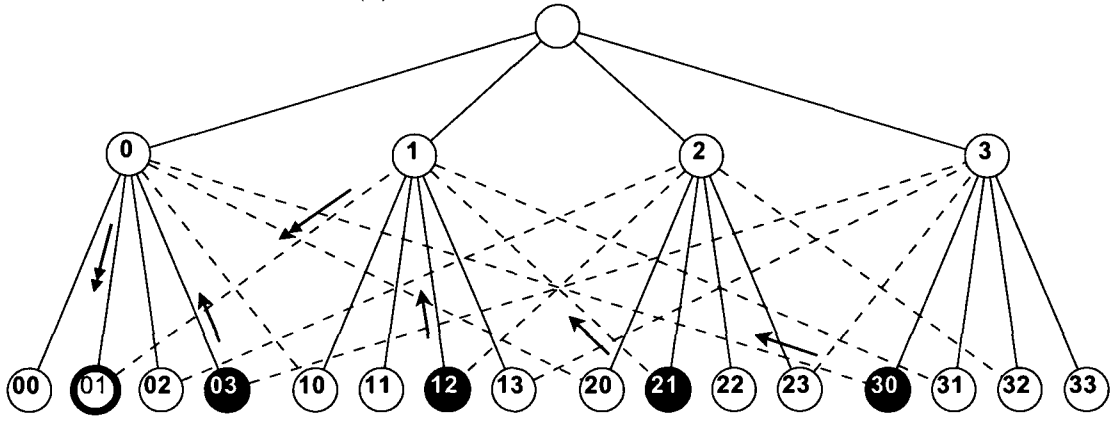
Figure 5.2: Range query in tree structure[14]

incoming query.

If apply the range query in an HD Tree, the route to find the nodes in the range is shown as in Figure 5.3a by choosing the D2H routing algorithm. Node 01 first sends the message to its DParent node 1. Node 1 broadcasts the message to its HChildren, and node 12 is found, since node 10, 11, 13 are not in the range, they forward the messages to their DParent 0, 2, 3, so that, all the node at depth 2 has been covered. Finally, the DParents forward the messages to their HChildren: node 03, 21 and 30. Thus all the nodes in the range has been found. Figure 5.3b shows the routes of the response queries from the destination nodes. Each of the destination node chooses the best distributed routing algorithm to send the query back. It ensures that the response queries have the



(a) Request of the range query



(b) Response to the range query

Figure 5.3: Range query in HD Tree[14]

minimum hops to go back to the source node (node 01 in this case). In this example, node 03 uses the H2H routing, it sends the message to its HParent node 0, and then the node 0 forwards the message to its HChild 01. Node 12 uses the H2D routing, it sends the message to its HParent node 1, and then the node 1 forwards the message to its DChild node 01. Node 21 uses the D2D routing, it sends the message to its DParent node 1, and then the node 1 forwards the message to its DChild node 01. Node 30 uses the D2H routing, it sends the message to its DParent node 0, and then the node 0 forwards the message to its HChild node 01.

Based on the comparison of the queries in the tree structure and the HD Tree structure, we can conclude that the routings in the HD Tree do not need to go through the root node, and can achieve a less time complexity than the tree routing.

From the above discussion, we can see that the whole query for a request may not use only one routing algorithm. As shown in Figure 5.3, the node 01 uses the D2H to send the request query to the destination nodes, but the destination node 12 uses the H2D routing to send the respond message back. Every node received a message will choose the best distributed routing algorithm to go based on the matching pattern. Therefore, based on my thesis work, we introduced two generalized routing algorithms X2X[43] and DROCR[43].

X2X routing algorithm is a greedy distributed routing algorithm in the HD Tree, which can only be applied between two nodes at the same depth of the HD Tree. It is a combination of the four distributed routing algorithms D2H, H2D, H2H, and D2D. When a query is requested, every node received the message do the matching between the destination node and itself, and then choose the appropriate distributed routing algorithm with the longest matching pattern to do the routing. It is not surprising that the X2X routing algorithm can provide better routing performance than the four basic distributed routing algorithms.

The distributed routing algorithms in the HD Tree can only be applied between nodes at the same depth in the HD Tree, with good routing performance. The hierarchical routing algorithms can be applied between any two nodes in the HD Tree, but with poor routing performance. The Distributed Routing Oriented Combined Routing algorithm (DROCR) for the HD Tree is proposed to find an appropriate way to combine the hierarchical routing algorithms with the distributed routing algorithms. The DROCR algorithm tends to use distributed routing as much as possible. It applies hierarchical routing only if the source node and the destination nodes are not at the same depth, or the matching found between them does not belong to any of the four matching patterns. In the DROCR algorithm, a source node first checks if the matching between it and

the destination node can satisfy any of the four matching patterns for the distributed routing algorithms. If matching patterns are satisfied, the DROCR simply chooses the distributed routing algorithm which coordinates best to the matching pattern. If none of the matching patterns can be satisfied, the hierarchical routing algorithms will be used to change the matching pattern and generate a new matching pattern. The 2HP and 2DP routing algorithms are used to truncate the non-matching part of source code, while the 2HC and 2DC routing algorithms are used to extend existing matching pattern or generate new matching patterns. The DROCR algorithm can achieve more performance gain, and performs better than the X2X algorithm in most cases.

5.2 Fault Tolerance and Load Balancing

Compared to tree structure, the HD Tree can achieve better fault tolerance. One node failure in an HD Tree would not occure the failure of the query except for some SPeers, which, as shown in the section 4.5, are the weak points in the HD Tree. Based on the structure of the HD Tree, there are five different kind of nodes, which have different size of the routing table.

1. Leaf node, if it is SPeer, the number of its neighboring nodes is 1,which is 1 parent.

2. Leaf node, if it is not SPeer, the number of its neighboring nodes is 2, which includes 1 HParent and 1 DParent.

3. Non-leaf node, if it is SPeer, the number of its neighboring nodes of is $2k$, which includes 1 parent and $2k-1$ children with 1 SPeer child in it.

4. Non-leaf node, if it is not SPeer, the number of its neighboring nodes of is $2k+2$, which includes 1 HParent, 1 DParent and $2k$ children.

5. Root node, the number of its neighboring nodes of is k, which includes k children.

The four different distributed routing algorithms in the HD Tree can be easily adapted to fault rolerable purpose. If a following node of the current node in a query is unavailable, the query can be simply redirected to any of the available neighboring nodes of the current node. After the current node sends the message to one of its available neighboring nodes, the node received the message will do the matching comparing between the destination node and itself, and choose the appropriate distributed routing algorithm to do the query based on its matching result. Therefore, if the current node is a leaf node, if it is also SPeer, it cannot survive in nodes failure, else if it is not SPeer, it can survive in 1 node failure. If the current node is a non-leaf node, if it is also SPeer, it can survive in at least $2k-3$ nodes failure, which includes an available non-SPeer child, else if it is not SPeer, it can survive in $2k+1$ nodes failure. If the current node is the root node, it can survive in $k-1$ nodes failure. So that the HD Tree can garantee that none of the nodes at depth less than $h-1$ will be isolated with $k-1$ nodes failure in the entire system.

As shown in the examples in section 5.1, we can see that the tree structure has a load balancing problem, with the nodes closer to the root having a heavier work load, because almost all requests go through the root in order to get to the destination nodes, especially when many requests are applied in the system at the same time. In the HD Tree, since we can use both the distributed routing algorithms and the hierarchical routing of the tree, we can balance the work load in the system. If most of the requests are about the nodes on the same depth in the HD Tree, we can use the hierarchical routing to perform some of the requests to allocate the work loads to the nodes on higher levels. Else if most of the requests are about the nodes on different depths in the HD Tree, we can choose the distributed routing to perform most of the query. So that the work loads will be mainly on the nodes of two neighboring depths in the HD Tree.

Chapter 6

Simulation Result

We chose the network simulator- ns2(release 2.33)[38] to do simulations. Ns2 is well-known for its simulation of routing and multicast protocols, and searching in the ad-hoc network. The reason why we chose ns2 is to ease any following studies of the HD Tree, in which we might adopt different scenarios such as lower layer network infrastructure, higher layer data space partitioning, and the P2P overlay application layer. By calculating the average routing hops between any two nodes in the HD Tree uses the four distributed routing algorithms presented in Chapter 4, and the two advanced routing algorithms given in the Chapter 5, we show that the HD Tree can provide efficient support for multi-dimensional range queries.

6.1 Simulation

6.1.1 Simulation Introduction

In this simulation, I have chosen the Z-order space-filling curve for multi-dimensional data space partitioning. In order to accommodate the dimensionality of the data space change, every node in the HD Tree manages a set of virtual nodes.

At the P2P overlay application layer, I simplify the data space in real application

scenarios to an obstructive multi-dimensional data space with [0, 1] range at each dimension. HChildren at depth d decompose a certain part of the abstract data space, which was previously decomposed by its HParent, half at each dimension according to the sequence determined by the Z-order spacing filling curve. Therefore, the HCode of a node actually indicates which part of the abstract data it is currently managing.

To get the average routing performance in the experiment, the average time is evaluated among all nodes at the same depth in the HD Tree. Because the worst case time is always *2d* hops at each depth, the routing performance is computed as the rate of the average time over the worst case time . Five distributed routing algorithms, H2D, D2H, H2H, D2D, and X2X and the DROCR algorithm are evaluated.

Due to the lack of hardware equipment, such as a large number of connected computers, experiments could not be carried out in a real testing environment. In order to simulate the routings in the HD Tree, we created a virtual wired P2P network in ns2. Since the HD Tree is on the application level, and we were only concerned about the routing in this level, TCP was chose to be the network routing protocol, which is originally supported by the ns2.

Time did not permit the implementation of the application for a large scale P2P network, so the simulation of the routings was only run at most to the *12*-th level of the HD Tree with *2*_ary. In other words, the largest network we tested has $2^{13} - 1 = 8191$ nodes.

6.1.2 Simulation Enviornment

In order to build the HD Tree structure over the ns2, I created a new *TCPApplication* which is *HDTApp*, as shown in the Figure 6.1. There is an *Agent* which belongs to every node, and I attached an *HDTApp* to each *Agent*. The *HDTApp*s are used to create virtual connections to other nodes, and communicate message between them. More than one *HDTApp* can be attached to one node. In order to simplify the management of sending and receiving messages in the HD Tree structure, we created four *HDTApp*s on each

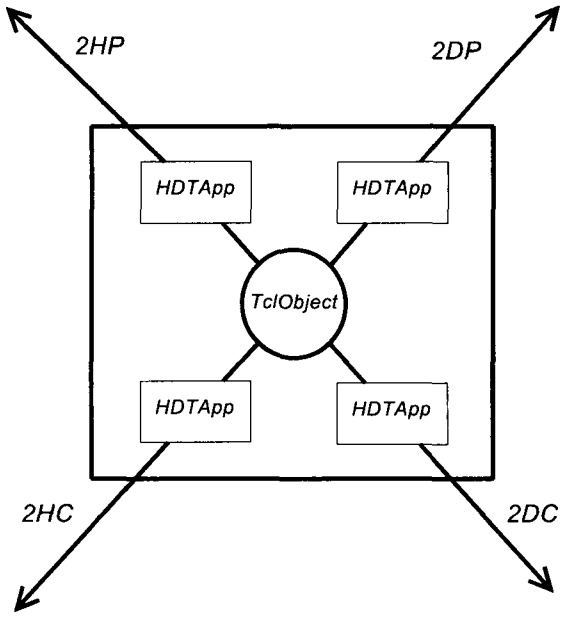


Figure 6.1: Simulation Topology

node, one for each basic hierachical routing operation as discussed in Chapter 4. I also created a *TclObject* on each node to manage the routing, and to coordinate the four *HDTApp*s. In order to support this structure, we added the following code.

- To create the structure of the *HDTApp*, the following codes are generated in the headfile: *hdt-app.h* under the folder *./apps/*:

```
class HDTApp : public TcpApp
  {
    private:
    protected:
      int support_HDT_;
      virtual int command(int argc, const char*const* argv);
      virtual void start();
      virtual void stop();
```

```
public:
    HDTApp();
    ~HDTApp();
    virtual int supportHDT() { return 1; }
    virtual void enableHDT() { support_HDT_ = 1; }
    virtual void send(int nbytes, AppData *data);
    virtual void recv(int nbytes);
};
```

- To create the structure of the *P2P*, the following codes are generated in the headfile: *ns-p2p.h* under the folder *./common/* :

```
class P2P : public TclObject
{
  private:
  protected:
    virtual int command(int argc, const char*const* argv);
    virtual void start() {}
    virtual void stop() {}
    virtual void initP2P(unsigned int) {}
  public:
    P2P();
    ~P2P();
    virtual void receive(char* s){}
};
```

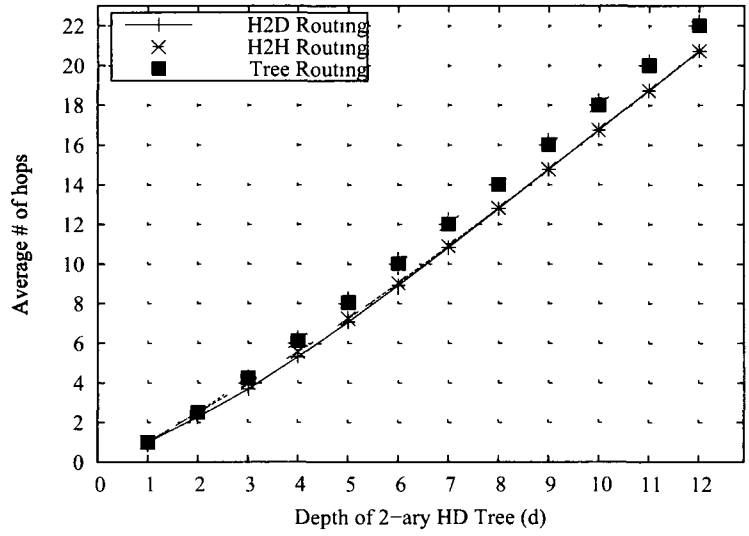


Figure 6 2 Performance of H2H and H2D routing

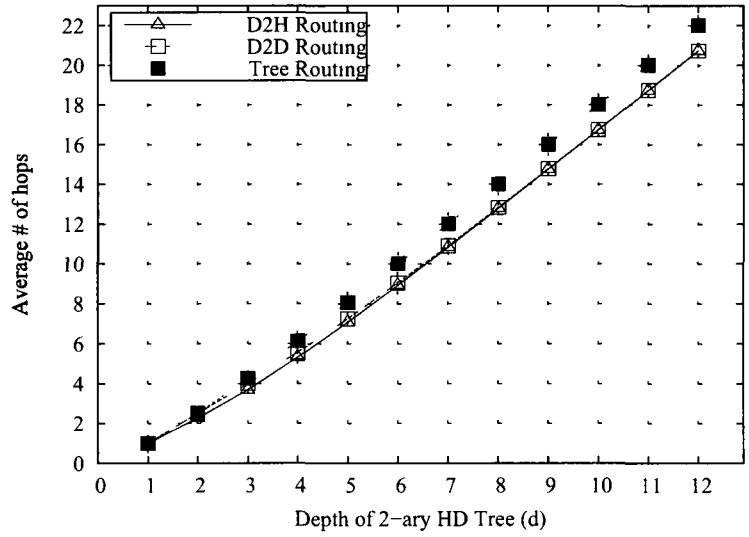


Figure 6 3 Performance of D2D and D2H routing

6.2 Performance Evaluation

6.2.1 Performance of H2D, D2H, H2H and D2D

I ran a group of experiments to get the average routing performance of H2D, D2H, H2H and D2D, as shown in Figure 6.2 and Figure 6.3. In order to get the average performance, we evaluated the routing among all nodes at each depth in the HD Tree from depth 1 to 12. As I have shown in section 4.5, the worst case of the HD Tree is the SPeer, which has the performance of the equivalent tree structure. Figures 6.2 and 6.3 show the average hops of the distributed routing algorithms and compares them with the average hops of the tree routing. If we apply each distributed routing algorithm separately, they are all able to achieve better performance than what we can achieve in an equivalent tree structure. During the analysis of the performance of the four distributed routing algorithms, I have shown that SPeers always have smaller matching sets than any of the four distributed routing algorithms. Therefore, tree routing always needs more hops to get to the destination node than distributed routing on average. In addition, D2H and H2D have the same routing performance, as well as D2D and H2H, because they are two pairs of reversed distributed operations of each other.

6.2.2 Performance of DROCR and X2X

As shown in Figure 6.4, we compared the average hops of X2X, D2H, H2D, H2H, D2D and tree routings. It is not surprising that we achieved a significant performance gain over the equivalent tree structure because X2X is a combination of the four distributed routing algorithms. X2X always chooses the shortest distributed route by comparing among all D2H, H2D, D2D, and H2H matching.

Figure 6.5 shows the comparison results of each routing algorithm with tree routing based on their average routing hops. At depth 1, the routing is the same as the tree routing, since there is no distributed structure used on this level, and all the peers at this depth are SPeers. From depth 2, the distributed structure is applied, and the distributed

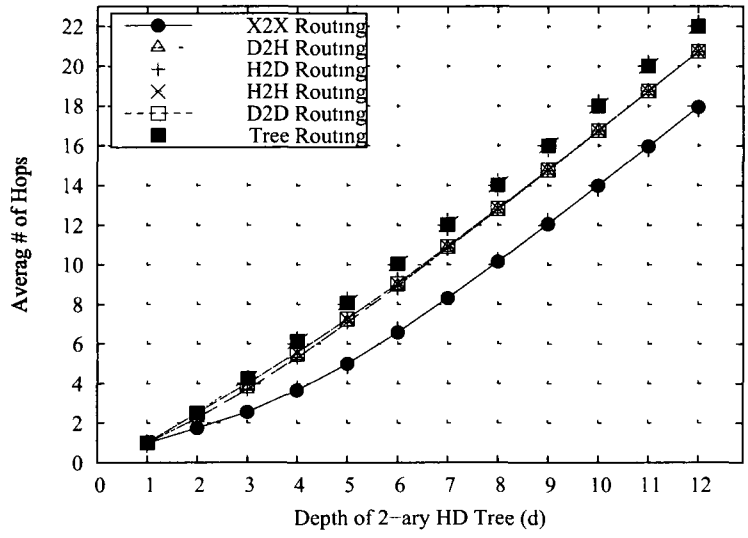


Figure 6.4: Performance comparison of distributed routings[43]

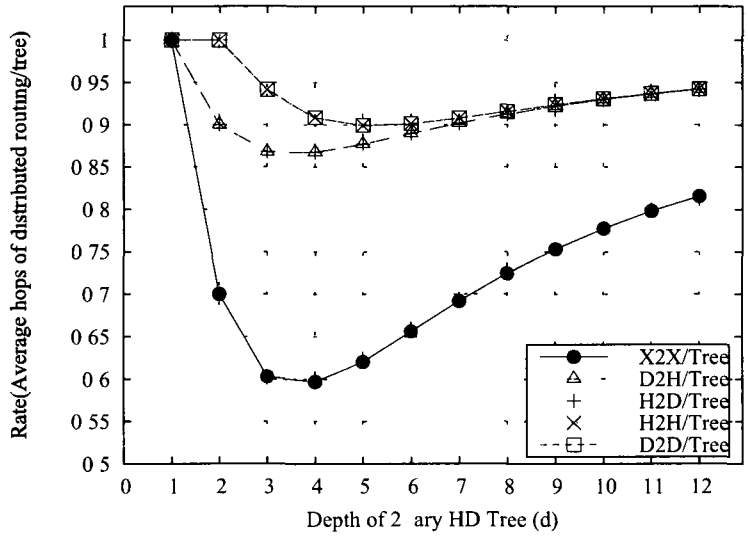


Figure 6.5 Performance comparison of distributed routings over tree[43]

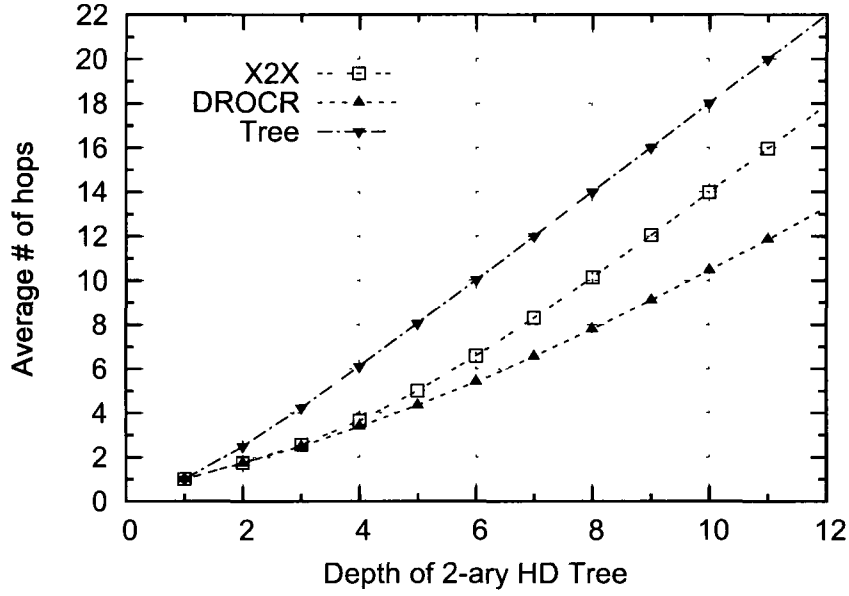


Figure 6.6: Average hops of DROCR[43]

routing algorithms perform better than the tree routing, except for D2D and H2H routing algorithms. Because at this depth, H2H routing algorithm only use the hierarchical links, while D2D routing algorithm only use the distributed links, they have the same routing performance as the tree routing. We can see that the distributed routing algorithms achieve the best performance at depth 4. At the highest depth in the experiment the four distributed routing algorithms: D2H, H2D, H2H and D2D can save about 6% of the tree routing, while the X2X can save about 20% of the tree routing.

The routing performances of the DROCR in the HD Tree are shown in Figures 6.6 and 6.7. We can see from these two figures that the DROCR can gain much more efficient routing performance, even over the X2X routing algorithm. Figure 6.7 shows that the DROCR algorithm saves almost 40% of the routing performance of tree routing at the highest depth, because DROCR combines hierarchical routing and distributed routing. As long as there is at least one matching found between a source node and a destination node, the longest matching pattern can always be used for routing.

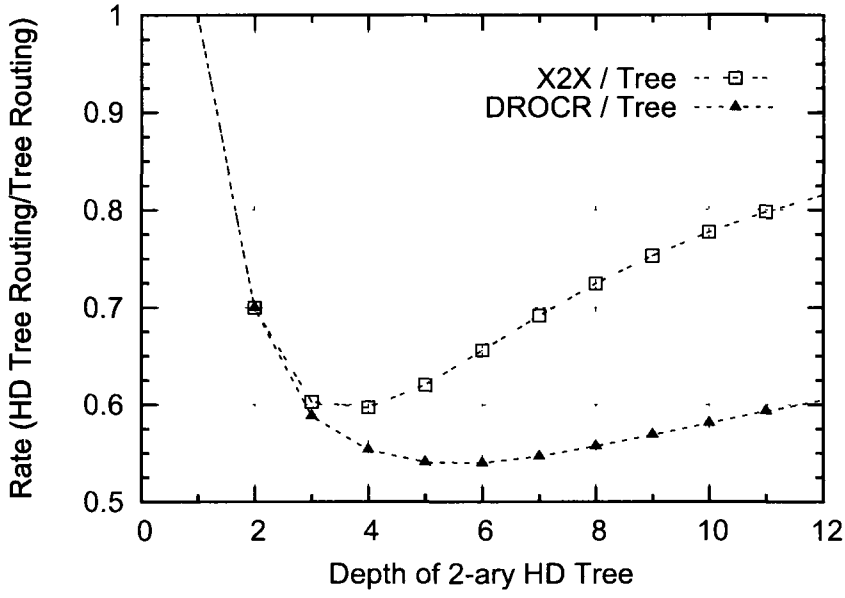


Figure 6.7: Average performance of DROCR[43]

6.2.3 Summary

Since all the routing algorithms I have introduced and simulated are designed exactly for the HD Tree structure, it is hard to adapt these to other P2P overlay networks. I can still compare the routing algorithms with other P2P overlay networks based on the results of the third party experiments, as shown in Table 6.1. We can see that our routing algorithms can achieve at least the same routing performance as other existing P2P overlay networks. Moreover, the HD Tree structure can also provide better support to multi-dimensional range queries by using the hierarchically distributed structure to manage the identifier space at the application level. Other P2P overlay networks directly adapted the data structures designed for the key queries for the identifier management, which has been proven is not suitable for multi-dimensional range queries. In conclusion, the HD Tree is able to maintain data locality among data items with an exponential rate of extension because of the scalability of its hierarchical structure.

	P2P Overlay Structure	Routing Performance	Routing State
SCARP	Skip graph	$O(logN)$	$logN$
MURK	CAN	$O(d \times N^{1/d})$	$2d$
ZNet	Skip graph	$O(logN)$	$logN$
Skipindex	Skip graph	$O(logN)$	$logN$
Squid	CHORD	$O(logN)$	$logN$
SONAR	CAN	$O(d \times N^{1/d})$	$2d$
HD Tree	HD Tree	$O(logN)$	$2k+1$

Table 6.1: Comparison of routing performance in different systems

Chapter 7

Conclusion and Future Work

Due to application constraint and system constraint that multi-dimensional range queries have brought to P2P overlay networks, the HD Tree is designed to give efficient support for multi-dimensional range queries. In this thesis, four basic routing algorithms H2D, D2H, H2H, and D2D, have been proposed for the HD Tree. In this chapter, a summary of our thesis work with the discussion of some contributions of our work will be provided. Possible directions for future research will be pointed out.

7.1 Conclusion

In this thesis, I explored two basic routing strategies in HD Tree, hierarchical routing and distributed routing. The hierarchical route is the route between any two nodes via some intermediate nodes along the only hierarchical structure of the HD Tree. Meanwhile, the distributed route is the route between two nodes at the same depth i via some intermediate nodes at the depth $i - 1$ and i only. For hierarchical routing I showed the four basic hierarchical routing operations: 2HP, 2DP, 2HC, and 2DC. For distributed routing, I proposed four distributed routing algorithms: H2D, D2H, H2H, and D2D, and four matching patterns for each of the distributed routing to skip some route steps. The computations for all the routing algorithms are very simple digit switching.

One of the motivations for designing the routing algorithms for the HD Tree is to better prove that the HD Tree can efficiently support multi-dimensional range queries. In Chapter 4, we gave a detailed discussion about how the routing algorithms can support multi-dimensional range queries.

Experiments to evaluate the performance of the four distributed routings, H2D, D2H, H2H, and D2D, carried out with the ns2 network simulator. These experiments were detailed in Chapter 6. First we built the HD Tree P2P overlay structure in ns2. Currently, the HD Tree overlay is built over the traditional wired network infrastructure, and I employed the Z-order space filling curve for data space partitioning. Each node in the HD Tree manages a certain number of virtual nodes in order to accommodate the dimensionality changes. In addition, I showed the average performance of the two advanced routing algorithms, X2X and DROCR, and compared them with the other routing algorithms. Finally, the comparison of the HD Tree and other existing P2P overlay networks was given. Based on the simulation results, we can conclude that: (1) the HD Tree is able to maintain data locality among data items with an exponential extending rate because of the scalability of its hierarchical structure; and (2) the HD Tree is able to maintain data locality among data items with an exponentially expanding rate because of its natural connection to recursive decomposition employed for data space partitioning.

7.2 Future Work

There are several possible ways for future research as follows:

- More experiments might be done on two cases: 1) The performance of a series of range queries in a fixed multi-dimensional data space. 2) The performance of a fixed range query in different multi-dimensional data space. These experiments would be conducted in order to prove that the HD Tree is able to maintain data locality among data items with an exponentially expanding rate because of its natural connection to recursive decomposition employed for data space partitioning.

- Continue to research on advanced routing algorithms that can give better performance by combining the hierarchical and distributed routing algorithms we proposed in this thesis.
- Experiments to prove the HD Tree can support fault tolerance and dynamic load balancing might be shown by using the hierarchical and distributed routing algorithms.

Bibliography

[1] T. Schutt, F. Schintke, and A. Reinefeld, *A Structured Overlay for Multi-dimensional Range Queries* . Euro-Parallel Processing , 2007, pp. 503-513.

[2] J. Louis Bentley , *Multidimensional Binary Search Trees Used for Associative Searching* . Commun. ACM, Vol. 18, No. 9., 1975, pp. 509-517.

[3] S. Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker, *A Scalable Content-Addressable Network* . In SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, Vol. 31, No. 4, October 2001, pp. 161-172.

[4] James Aspnes, Gauri Shah. *Skip graphs* . ACM Trans. Algorithms, Vol. 3, No. 4, 2007, pp. 384-393.

[5] Prasanna Ganesan Beverly Yang Hector GarciaMolina, *One Torus To Rule Them All: Multidimensional Queries In P2P Systems* . Seventh International Workshop On The Web And Databases, June 2004, pp. 19-24.

[6] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, *Freenet: A distributed anonymous information storage and retrieval system* . Lecture Notes in Computer Science, vol. 2009, 2001, pp. 46-66.

[7] Hans-Emil Skogh, Jonas Haeggstrom, Ali Ghodsi, Rassul Ayani, *Fast Freenet: Improving Freenet Performance by Preferential Partition Routing and File Mesh Prop-*

agation . Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops, vol. 2, 2006, pp. 9-17.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, *Making Gnutella-like P2P Systems Scalable* . in ACM SIGCOMM, 2003.

[9] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, *Measurements, analysis, and modeling of bittorrent-like systems* . Internet Measurement Conference, 2005, pp. 35-48.

[10] I. Stoica, R. Morris et al., *Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications* . IEEE/ACM Trans. Vol. 11, No., Feb, 2003, pp. 17–32.

[11] A. Rowstron and P. Druschel, *Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems* . in Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg. Springer-Verlag, 2001, pp. 329-350.

[12] Wikipedia the free encyclopedia. P2P. http://en.wikipedia.org/wiki/Peer-to-peer, 2009.

[13] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, *Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web* . in In ACM Symposium on Theory of Computing, 1997, pp. 654-663.

[14] Yunfeng Gu, *Hierarchically distributed tree* . Technical Report, University of Ottawa, Canada. TR-10. (In preparation).

[15] R. Blanco, N. Ahmed, D. Hadaller, L. G. A. Sung, H. Li, M. A. Soliman, *A Survey of Data Management in Peer-to-Peer Systems* . Technical Report CS-2006-18 (20 June 2006), pp. 1-51.

[16] Wikipedia the free encyclopedia. SHA-1. http://en.wikipedia.org/wiki/SHA-1.

[17] Eng Keong Lua , Jon Crowcroft , Marcelo Pias , Ravi Sharma , Steven Lim, *A Survey and Comparison of Peer-to-Peer Overlay Network Schemes* . IEEE Communications Surveys and Tutorials, Vol. 7 (2005), pp. 72-93.

[18] R. A. Finkel, J. L. Bentley, *Quad Trees: A Data Structure for Retrieval on Composite Keys* . Acta Informatica, Vol. 4, No. 1. (1 March 1974), pp. 1-9.

[19] Hanan Samet, *The Quadtree and Related Hierarchical Data Structures* . ACM Comput. Surv., Vol. 16, No. 2. (June 1984), pp. 187-260.

[20] G.M. Morton, *A computer oriented geodetic data base and a new technique in file sequencing* . Technical report, Ottawa, Canada, IBM Ltd. 1966.

[21] J. A. Orenstein, T. H. Merrett, *A class of data structures for associative searching* . In PODS '84: Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems (1984), pp. 181-190.

[22] H. V. Jagadish, *Linear clustering of objects with multiple attributes* . In SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data (1990), pp. 332-342.

[23] Hyuncheol Kim, Younghwa Kim , Seungae Kang , Kwangjoon Kim, *Restricted Path Flooding Scheme in Distributed P2P Overlay Networks* . ICISS. International Conference on Information Science and Security (2008), pp. 58-61.

[24] Qianbing Zheng, Xicheng Lu, Peidong Zhu, Wei Peng, *An efficient random walks based approach to reducing file locating delay in unstructured P2P network* . IEEE GLOBECOM '05, Global Telecommunications Conference (2005), pp. 980-984.

[25] Xueyan Tang, Jianliang Xu, Wang-Chien Lee, *Analysis of TTL-Based Consistency in Unstructured Peer-to-Peer Networks* . IEEE Transactions on Parallel and Distributed Systems(2008), pp. 1683-1694.

[26] B. Karp et al., *Spurring Adoption of DHTs with OpenHash, a Public DHT Service.* Peer-to-Peer Systems III (IPTPS 2004), Berkeley, CA, USA, Feb. 2004, pp. 195-205.

[27] C. Zhang et al., *SkipIndex: Towards a Scalable Peer-to-Peer Index Service for High Dimensional Data.* Vol. TR-703-04, Princeton University 2004.

[28] Y. Shu et al., *Supporting Multi-dimensional Range Queries in Peer-to-Peer Systems.* In 5th IEEE Int. Conf. on Peer-to-Peer Computing, 2005. P2P 2005, pp. 173-180.

[29] J. Kubiatowicz et al., *Oceanstore: An Architecture for Global-Scale Persistent Storage.* Proc. ACM ASPLOS, Nov. 2002.

[30] W. J. Bolosky et al., *Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs.* Proc. 2000 ACM SIGMETRICS Int'l. Conf. Measurement and Modeling of Comp. Sys., June 2000, pp. 34-43.

[31] M. Waldman, A. D. Rubin, and L. F. Cranor, *Publius: A Robust, Tamper-evident, Censorship-resistant, Web Publishing System.* Proc. 9th USENIX Security Symp., Denver, CO, USA, 2000.

[32] F. Dabek et al., *Wide-area Cooperative Storage with CFs.* Proc. 18th ACM Symp. Operating Systems Principles, 2001, pp. 202–15.

[33] R. Cox, A. Muthitacharoen, and R. Morris, *Serving DNS using Chord.* Proc. First Int'l. Wksp. Peer-to-Peer Systems, Mar. 2002.

[34] N. J. A. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet, *A scalable overlay network with practical locality properties.* Proc. USITS, 2003.

[35] Wikipedia the free encyclopedia. Skip list. http://en.wikipedia.org/wiki/Skip_list, 2008.

[36] Wikipedia the free encyclopedia. Falso positive. http://en.wikipedia.org/wiki/False_positive_paradox, 2009.

[37] Cristina Schmidt, Manish Parashar, *Enabling Flexible Queries with Guarantees in P2P Systems* . IEEE Internet Computing, vol. 8, no. 3, May/June 2004, pp. 19-26.

[38] VINT. The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[39] Gnutella development forum. http://groups.yahoo.com/group/the gdf/files/, 2001.

[40] Gnutella ultrapeers. http://rfcgnutella.sourceforge.net/Proposals/Ultrapeer/, 2002.

[41] Official BitTorrent website. http://www.bittorrent.com/, 2008.

[42] Official BitTorrent specification. http://www.bittorrent.org/, 2009.

[43] Yunfeng Gu, Azzedine Boukerche, Xun Ye, and Regina B. Araujo, *Supporting Multi-dimensional Range Query in HD Tree* . The 14th IEEE DS-RT, Fairfac, Virginia, USA, Oct. 17-20, 2010 (in press).