Data Distribution Management in Large-Scale Distributed Environments

Yunfeng Gu

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

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

To the memory of my father,

Zhenguang Gu (Apr. 1933 - May 2011)

Abstract

Data Distribution Management (DDM) deals with two basic problems: how to distribute data generated at the application layer among underlying nodes in a distributed system and how to retrieve data back whenever it is necessary. This thesis explores DDM in two different network environments: peer-to-peer (P2P) overlay networks and cluster-based network environments. DDM in P2P overlay networks is considered a more complete concept of building and maintaining a P2P overlay architecture than a simple data fetching scheme, and is closely related to the more commonly known associative searching or queries. DDM in the cluster-based network environment is one of the important services provided by the simulation middle-ware to support real-time distributed interactive simulations. The only common feature shared by DDM in both environments is that they are all built to provide data indexing service. Because of these fundamental differences, we have designed and developed a novel distributed data structure, Hierarchically Distributed Tree (HD Tree), to support range queries in P2P overlay networks. All the relevant problems of a distributed data structure, including the scalability, self-organizing, fault-tolerance, and load balancing have been studied. Both theoretical analysis and experimental results show that the HD Tree is able to give a complete view of system states when processing multi-dimensional range queries at different levels of selectivity and in various error-prone routing environments. On the other hand, a novel DDM scheme, Adaptive Grid-based DDM scheme, is proposed to improve the DDM performance in the cluster-based network environment. This new DDM scheme evaluates the input size of a simulation based on probability models. The optimum DDM performance is best approached by adapting the simulation running in a mode that is most appropriate to the size of the simulation.

List of Publications Related to Thesis

Refereed Journal Papers

- Yunfeng Gu, A. Boukerche: "HD Tree: A Novel Data Structure to Support Multi-Dimensional Range Query for P2P Networks". Journal of Parallel and Distributed Computing, Volume 71, Issue 8 (Aug. 2011), pp. 1111-1124

- Yunfeng Gu, A. Boukerche: "An Efficient Adaptive Transmission Control Scheme for Large-scale Distributed Simulation System". Parallel and Distributed Systems, IEEE Transactions on, Volume 20, Issue 2 (Feb 2009), pp. 246-260

- Yunfeng Gu, A. Boukerche, and Regina B. Araujo: "Performance Analysis of An Adaptive Dynamic Grid-based Approach to Data Distribution Management". Journal of Parallel and Distributed Computing, Vol. 68, Issue 4 (Apr. 2008), pp. 536-547

- A. Boukerche, Yunfeng Gu, and Regina B. Araujo: "An Adaptive Dynamic Grid-Based approach to DDM for Large-scale Distributed Simulation Systems". Journal of Computer and System Sciences, Volume 74 (Sep. 2008), pp. 1043-1054

Refereed Conference papers

- Yunfeng Gu, A. Boukerche: "Error-Resilient Routing for Supporting Multidimensional Range Query in HD Tree". Distributed Simulation and Real Time Applications (DS-RT'11), 2011 IEEE/ACM 15th International Symposium on, 4 - 7 Sep. 2011, MediaCITYUK, Salford, UK. pp. 44-51

- Yunfeng Gu, A. Boukerche: "Hierarchically Distributed Tree". Computers and Communications (ISCC'11), 2011 IEEE 16th Symposium on, 28 Jun. - 1 Jul. 2011, Kerkyra, Greece. pp. 91-96

- Yunfeng Gu, A. Boukerche, Xun Ye, and Regina B. Araujo: "Supporting Multi-dimensional Range Query in HD Tree". Distributed Simulation and Real Time Applications (DS-RT'10), 2010 IEEE/ACM 14th International Symposium on 17-20 Oct, 2010, Fairfax, Virginia, USA. pp. 71-78

- A. Boukerche, Yunfeng Gu: "Performance Modeling of A Grid-based Data Distribution Management Protocol". Computers and Communications, 2008, ISCC 2008, IEEE Symposium on. 6-9 July 2008, Marrakech, Morocco. pp, 455-461

- Yunfeng Gu, A. Boukerche: "An Adaptive Resource Allocation Scheme for Large-scale Distributed Simulation System". Distributed Simulation and Real-Time Applications (DS-RT'07), 2007 IEEE 11th International Symposium on 22-26 Oct. 2007, Greece. pp. 48-56 (Recipient of Best Paper Award)

- A. Boukerche, Yunfeng Gu, and Regina B. Araujo: "Performance Analysis of An Adaptive Dynamic Grid-based Approach to Data Distribution Management". Distributed Simulation and Real-Time Applications (DS-RT'06), 2006 IEEE 10th International Symposium on 2-4 Oct. 2006, Spain. pp. 175-184

- Yunfeng Gu, A. Boukerche, and Regina B. Araujo: "An Adaptive Dynamic Grid-based Approach to Data Distribution Management". Parallel and Distributed Processing Symposium, 2006, IPDPS 2006, 20th International, 25-29 Apr. 2006. Page(s): 5 pp.

Acknowledgements

This thesis is dedicated to the memory of my father, who had been expecting me till the last minute of his life in this world.

First and foremost, I am immensely grateful to my supervisor Dr. Azzedine Boukerche. This research work would not have been possible without his professionalism, guidance, encouragement, moral and financial support throughout the entire period of my Ph.D thesis work.

I would like to thank Hengheng Xie and Zhenxia Zhang, for being there for me whenever I needed someone to argue, discuss, and validate my work with.

My special thanks also goes to Robson Eduardo De Grande for providing consistent technical support to our working environment, and Anahit Martirosyan for helping me with my thesis writing.

Contents

1 Introduction **1**

1.1 Addressing DDM in Different Network Environments 3

1.2 DDM in P2P Overlay Networks 5

1.2.1 Background 5

1.2.2 Problem Statement 8

1.2.3 Research Objective 8

1.2.4 Main Contributions 9

1.3 DDM in Cluster-Based Network Environments 10

1.3.1 Background 10

1.3.2 Problem Statement 11

1.3.3 Research Objective 12

1.3.4 Main Contributions 12

1.4 Thesis Organization 13

2 DDM in P2P Overlay Networks **14**

2.1 Related Works 14

2.1.1 Multi-dimensional Data Space Partitioning and Mapping 14

2.1.2 Corresponding Overlay Network Support 16

2.1.3 Why HD Tree? 19

2.2 Proposal of New Approach 20

2.2.1 HD Tree 20

2.2.1.1 Notations in Tree 20
2.2.1.2 Definition of HD Tree 21
2.2.1.3 HD Table 23
2.2.2 Properties of HD Tree 24
2.2.2.1 Notations in HD Tree 25
2.2.2.2 HCode and DCode 26
2.2.2.3 SPeers 26
2.2.2.4 System Parameters 27
2.2.3 Basic Routing Strategies 27
2.2.3.1 Hierarchical and Distributed Routing Mechanism 27
2.2.3.2 Combined Routing Mechanism 33
2.2.4 Supporting Multi-dimensional Range Queries 38
2.2.4.1 Data Space Partitioning and Mapping 39
2.2.4.2 Multi-dimensional Range Queries 40
2.2.5 Error Resilient Routing 42
2.2.5.1 Connecting SPeers 42
2.2.5.2 Error Resilient Routing Mechanism 43
2.2.5.3 2SDP and 2SDC 46
2.2.6 Maintenance Operations 48
2.2.6.1 Entry Table 48
2.2.6.2 Join and Leave Operations 49
2.2.6.3 Performance Analysis 51
2.2.7 Dynamic Load Balancing 51
2.2.7.1 H2H and H2D Local Adaptation 52
2.2.7.2 D2H Repartitioning 52
2.3 Experimental Results 54
2.3.1 Simulation Environment 54
2.3.2 Average Routing Performance 55

2.3.3 Performance of Range Queries 58
2.3.4 Performance of Join and Leave 64
2.3.5 Performance of D2H Repartitioning 66
2.4 Summary 69

3 DDM in Cluster-Based Network Environments 73
3.1 Basic Concepts 73
3.1.1 Routing Space 73
3.1.2 Objects, Federates and Federation 74
3.1.3 Behavior of Entities 75
3.1.3.1 Distribution of Objects 75
3.1.3.2 Movement of Objects 75
3.1.3.3 Publishing and Subscribing 75
3.1.4 Other Important Concepts 76
3.1.4.1 Timestep 76
3.1.4.2 Publisher and Subscriber 76
3.1.4.3 Publication and Subscription Region 76
3.1.4.4 Maximum Publication and Subscription Region 77
3.1.4.5 Matching 77
3.1.5 Performance of DDM Implementations 78
3.2 Related Works 79
3.2.1 Region-based DDM Approach 79
3.2.2 Grid-based DDM Approaches 80
3.2.2.1 Fixed Grid-based DDM Approach 80
3.2.2.2 Dynamic Grid-based DDM Approach 80
3.2.3 Other DDM Schemes 82
3.3 Foundations of Grid-based DDM 83
3.3.1 Simulation Scenario 83

3.3.2 Matching Accuracy 84
3.3.3 Timestep 85
3.3.4 Uniform Distribution Patterns 85
3.4 Matching Model 86
3.4.1 Static Data 86
3.4.1.1 Definition of Static Data 87
3.4.1.2 Inside Static Data 87
3.4.2 Performance Analysis 88
3.4.2.1 Evaluating Static Data 88
3.4.2.2 Evaluating DDM Performance 89
3.4.3 Matching Model 90
3.4.3.1 Definitions 90
3.4.3.2 Publishing/Subscribing Probability 92
3.4.3.3 Matching Probability 94
3.4.3.4 Lower Boundary Analysis 94
3.4.3.5 Adaptive Resource Allocation Control Scheme 97
3.5 Switching Model 98
3.5.1 Dynamic Data 98
3.5.1.1 Two Phases of Transmission 99
3.5.1.2 States of a Cell 100
3.5.1.3 Definition of Dynamic Data 101
3.5.1.4 Inside Dynamic Data 101
3.5.2 Performance Analysis 104
3.5.2.1 The State-transition of a Cell 104
3.5.2.2 Evaluating New Pub/Sub Data and UnPub/UnSub Data 106
3.5.2.3 Evaluating TellPubJoin, TellSubJoin Data 106
3.5.2.4 Evaluating TellLeave Data 107
3.5.2.5 Evaluating RefreshCList Data 108

3.5.2.6 Evaluating Transmission Performance 108
3.5.3 Switching Model 109
3.5.3.1 State-transition Probabilities at Object Level 110
3.5.3.2 State-transition Probabilities at Federate Level 112
3.5.3.3 State-transition Probabilities at Federation Level 114
3.5.3.4 Lower Boundary Analysis 115
3.6 Experimental Results 117
3.6.1 Performance of ARAC Scheme 117
3.6.1.1 Simulation Environment 117
3.6.1.2 Performance Evaluation 118
3.6.1.3 Storage Allocation Rate 119
3.6.1.4 Performance Comparisons 120
3.6.2 Performance of ATC Scheme 122
3.6.2.1 Simulation Environment 122
3.6.2.2 Performance Evaluation 123
3.6.2.3 Performance Comparisons 124
3.7 Summary 125

4 Conclusions and Future Works **127**
4.1 Conclusions 127
4.2 Future Works 129

List of Tables

1.1 Comparison of DDM in P2P and Cluster-based Network Environments . 4

2.1 Comparison of P2P Overlay Systems I 15

2.2 Comparison of P2P Overlay Systems II 17

3.1 State-transition at Object and Federate Level 104

3.2 State-transition at Federation Level 107

List of Figures

1.1 DDM in P2P Overlay Networks and Cluster-based Network Environments 2

2.1 Coding of a Complete $4-ary$ Tree 21
2.2 Constructing a $4-ary$ HD Tree 22
2.3 Constructing a $2-ary$ HD Tree 23
2.4 H table, D table, and HD table 24
2.5 HParent, DParent, Mapping Entry, and SPeer 25
2.6 Basic Hierarchical Operations 28
2.7 Basic Distributed Operation - D2H 28
2.8 Basic Distributed Operation - H2D 29
2.9 Basic Distributed Operations - D2D & H2H 29
2.10 D2H Routing When Source is a SPeer 31
2.11 D2H Routing When Source is a Non-SPeer 32
2.12 Hierarchical Routing without Match 33
2.13 Hierarchical Routing with Match 33
2.14 Distributed Routing without Match 35
2.15 Distributed Routing with Match 35
2.16 System Overview 38
2.17 Direct Mapping from Data Space to Identifier Space 39
2.18 Range Query in Tree 40
2.19 D2H Query in HD Tree 41

2.20 Response to D2H Query 41
2.21 Connecting SPeers 42
2.22 Comparison of SPeer's Routing Operations 46
2.23 Entry Table at Node 23 in $4-ary$ HD Tree 48
2.24 Entry Table at Node 02 in $4-ary$ HD Tree 49
2.25 H2H and H2D Local Adaptation 51
2.26 D2H Load Repartitioning 53
2.27 Routing and Range Query in Ideal Routing Environment 56
2.28 Range Query in Error-Prone Routing Environment (I) 59
2.29 Range Query in Error-Prone Routing Environment (II) 63
2.30 Performance of Join and Leave Operations 65
2.31 Range Query at Different Order of D2H Repartitioning 67
2.32 Routing at Different Order of D2H Repartitioning 68

3.1 Dynamic Data Generated by the Movement of Objects (I) 102
3.2 Sate-transition Probabilities at Single Object Level 110
3.3 Performance Evaluation of Matching Model 118
3.4 Comparison of Storage Allocation Rates 120
3.5 Inside Storage Allocation Rate of Grid-based Schemes 122
3.6 Performance Evaluation of Switching Model 123
3.7 Performance Comparisons 125

Index

2DC: Send to DChild, 28
2DP: Send to DParent, 28
2HC: Send to HChild, 28
2HP: Send to HParent, 28
2SDC: Send to SPeer DChild, 46
2SDP: Send to SPeer DParent, 46

AGB: Adaptive Grid-based, 12
AMGT: Adaptive Message Grouping Transmission, 117
API: Application Programming Interface, 1
APR: Average Publishing Rate, 93
APSR: Average Publication Switching Rate, 111
ARAC: Adaptive Resource Allocation Control, 12
ASR: Average Subscribing Rate, 93
ASSR: Average Subscription Switching Rate, 111
ATC: Adaptive Transmission Control, 12

CAN: Content Addressable Network, 3

D table: Distributed table, 23
D2D: Send to DSibling via DParent, 29
D2H: Send to HSibling via DParent or HChild, 29
DChild: Distributed Child, 25
DChildren: Distributed Children, 24
DCode: Distributed Code, 26
DDM: Data Distribution Management, 1
DHT: Distributed Hash Table, 6
DParent: Distributed Parent, 23
DPeer: Distributed Peer, 25
DROCR: Distributed Routing Oriented Combined Routing algorithm, 33
DSiblings: Siblings in D table, 23

FMGT: Fixed Message Grouping Transmission, 117
FPP: Federate Publishing Probability, 91
FSP: Federate Subscribing Probability, 91

H table: Hierarchical table, 23
H2D: Send to DSibling via HParent or DChild, 29
H2H: Send to HSibling via HParent, 29
HChild: Hierarchical Child, 25

HChildren: Hierarchical Children, 24

HCode: Hierarchical Code, 26

HD table: Combination of H table and D table, 23

HD Tree: Hierarchically Distributed Tree, 9

HLA: High Level Architecture, 1, 3

HParent: Hierarchical Parent, 23

HPeer: Hierarchical Peer, 25

HSiblings: Siblings in H table, 23

MGRP: Multi-cast Group, 78

OPP: Object Publishing Probability, 91

OSP: Object Subscribing Probability, 91

P2P: Peer-to-Peer, 1

Pub: Publishing, 87

RTI: Run-Time Infrastructure, 1, 3

SDC: SPeer DChild, 46

SDP: SPeer DParent, 46

SMT: Single Message Transmission, 108, 117

SPeer: Self-mapping DPeer, 26

Sub: Subscribing, 87

UnPub: Un-publishing, 102

UnSub: Un-subscribing, 102

X2X: Combination of D2H, H2D, D2D, and H2H, 32

Chapter 1

Introduction

In general, Data Distribution Management (DDM) deals with two basic problems: how to distribute data generated at the application layer among underlying nodes in the distributed system and how to retrieve data back whenever it is necessary. Alternatively, these two problems can also be described as (1) the partitioning of the application data space and (2) the mapping from the application data space into the system identifier space. The concept of DDM is also defined in High Level Architecture/Run-Time Infrastructure (HLA/RTI) under IEEE Standard 1516. It is one of seven service APIs provided by RTI. HLA is a general purpose architecture for real-time distributed interactive simulation systems. It is used to support the portability and interoperability of large numbers of distributed simulations across various computing platforms. RTI is the implementation of the HLA interface specification. It manages the interaction between simulations and coordinates operations inside the simulation. The goal of DDM defined in HLA/RTI is to establish simulation inter-connectivity to send relevant data on request with minimal excess or irrelevant data.

In this thesis, DDM is addressed in two different network environments: P2P overlay networks and cluster-based network environments. Although DDM provides the same data indexing service in both distributed network environments, it serves upper layer applications with very different purposes, and it is supported by underlying networks with

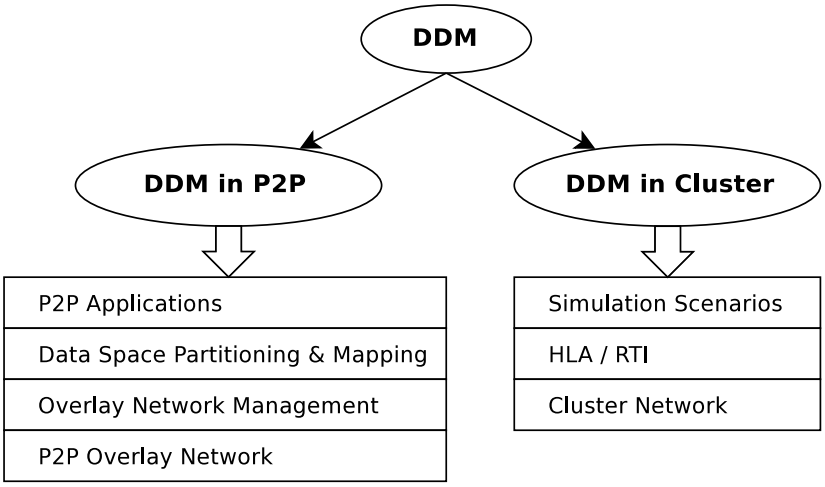


Figure 1.1: DDM in P2P Overlay Networks and Cluster-based Network Environments

distinctive infrastructures. In P2P overlay networks, peers share part of their resources. Both computing and data load are distributed to peers across the entire overlay network. The network size is not limited and may even expand to the Internet scale. However, peers might have different capacity for load sharing, some might join or leave the system, or even fail due to unstable connectivity. DDM in P2P overlay networks is more feasible for massive data sharing systems and parallel computing. On the other hand, all peers in a cluster-based network are dedicated resources in the distributed environment. The cluster-based network might be very limited in size, but the status of a peer is highly reliable and the connectivity is always stable. DDM in such network environments has been commonly used to support more intensive interactive simulations.

Because of the fundamental differences at both the application layer and the underlying network layer, the research work of DDM has to follow very different methodologies, which in turn diverts our interests and objectives in each area significantly. In the first section of this chapter, we discuss and compare the main features of DDM in the two different aforementioned network environments; in Section 1.2 and 1.3, we introduce the background information, research interests, objectives, and main contributions of DDM in both network environments respectively; and in Section 1.4, we describe the organization of this thesis.

1.1 Addressing DDM in Different Network Environments

Figure 1.1 shows two parts of the work that has been done in this thesis: DDM in P2P overlay networks and DDM in cluster-based network environments. The only common feature shared by both is that they both provide data indexing service in the distributed environment. However, DDM in P2P overlay networks is considered more of a complete concept of building and maintaining P2P overlay architecture (CAN[1,2] and Chord[1,3] for examples) than a simple data fetching technique, and is an open area that has been receiving a great deal of interest in the literature since 1960s. DDM in P2P overlay networks is closely related to associative searching or queries, which includes key queries, semantic key queries, range queries, multi-attribute queries, joined queries, aggregation queries, etc. On the other hand, DDM in the cluster-based network environment is one of the important services provided by the simulation middle-ware to support large-scale distributed interactive simulations. For this part of the thesis work, we use HLA/RTI as the simulation middle-ware. HLA/RTI is a publish/subscribe system. In HLA/RTI, all components are packed and provided as services to the higher layer simulation scenario. The details of the DDM implementation is implementation-dependent and is not publicly known.

Table 1.1 is a list of important features of DDM in both environments. In a P2P overlay network, the study object of DDM is the application data in the data space. All peers in the system share part of their resources to participate in system-wide activities, and they are organized in either a tightly controlled topology (structured P2P networks) or a random graph (unstructured P2P networks). Therefore, the main challenge of DDM in the P2P overlay network is to build a robust and reliable P2P search network that is not only efficient in retrieving data from multiple peers, but also inherently scalable, self-organizing, fault tolerant, and load balancing. Nevertheless, in P2P overlay networks,

	DDM in P2P	**DDM in Cluster**
Context	P2P overlay architecture - structured P2P - unstructured P2P	Simulation middle-ware (Publish/Subscribe system) - HLA / RTI
Applications	- Distributed directory system - P2P domain name services - Network file systems - Massive-scale fault-tolerant system, etc.	Real-time distributed interactive simulations
Study object	Application data in Data Space	Simulation data in Routing Space
Resources	Shared resources	Dedicated resources
Service provided	Data indexing	Data indexing
Underlying topology	- Structured P2P tightly controlled structure - Unstructured P2P random graph	Complete connection
Partitioning and mapping	- Structured P2P DHT Space filling curves - Unstructured P2P duplicating popular contents	Implementation-dependent (not publicly known)
Retrieving data	Request-based	Subscription-based
Main challenges	Building distributed data structure	Matching detection
Main trade-off	De-centralization vs. Global awareness	Communication overhead vs. Computation complexity
Research focuses	Scalability / Self-organizing / Fault tolerance / Load balancing	DDM performance

Table 1.1: Comparison of DDM in P2P and Cluster-based Network Environments

data is distributed and retrieved back by following the same partitioning and mapping strategy. Data retrieval is basically a request-based transmission involving a series of routing operations. The data producer distributes data to the data owner, which will be responsible for storing data and responding to queries, and the data consumer retrieves data from the data owner. The destination node (data owner) is computed progressively at each intermediate node on the route. The main trade-off of DDM in the P2P overlay network is the de-centralization of the application data and computation load versus the global awareness of system-wide operations. The work done in this part of the research

aims to design and develop a novel and complete P2P architecture in order to support queries for much more general purposes.

Similarly, DDM in the cluster-based network environment studies the simulation data (simulation objects) in *routing space*. However, all underlying nodes in the cluster are fully connected to each other and are dedicated resources to the higher layer simulation scenario. Retrieving data is a subscription-based process involving a considerable amount of computation load from the data owner. The *publisher* (Subsection 3.1.4) and *subscriber* (Subsection 3.1.4) register their interests, instead of distributing data, to the data owner. The data owner is responsible for detecting the *matching* (Subsection 3.1.4), which is to compute the intersection region in *routing space* that has been both published and subscribed by some objects. Once a *matching* is detected, the data owner notifies corresponding *publisher*s and *subscriber*s, and the relevant data is sent from *publisher*s to *subscriber*s directly. Therefore, the main challenge of DDM in the cluster-based network environment is to detect *matching*. Because detecting *matching* can be done either exactly in a exhaustive manner or with certain degree of tolerance by the data owner, the main trade-off of DDM in the cluster-based network environment is the communication overhead versus the computation complexity. In this part of the thesis, we are interested in improving the performance of DDM service. We propose a new adaptive DDM protocol, which tries to optimize the performance of DDM service by evaluating the size of a simulation based on two probability models.

1.2 DDM in P2P Overlay Networks

1.2.1 Background

P2P is a potentially disruptive technology with numerous applications such as distributed directory systems, P2P domain name services, network file systems, massive parallel systems, distributed e-mail systems, and massive-scale fault-tolerant systems. However,

these applications will not be realized without a robust and reliable P2P search network.

The most popular P2P overlay networks that have been well studied or deployed so far can be categorized into two classes: structured P2P overlay networks[2–7] and unstructured P2P overlay networks[8–20]. Structured P2P overlay networks are also called DHT-based systems, in which the data space is partitioned and mapped into the identifier space by using a randomization function - the Distributed Hash Table (DHT[29–31]). The DHT-based system normally has to maintain a tightly controlled topology. Some of the well known DHT-based systems include CAN[1, 2], Chord[1, 3], Pastry[1, 4], Tapestry[1, 5], Kademlia[1, 6], and Viceroy[1, 7].

Unlike DHT-based P2P systems, unstructured P2P overlay networks, like Gnutella[1, 8, 9], Freenet[1, 10], FastTrack/KaZaA[1, 11–14], BitTorrent[1, 15, 16], and eDonkey[1, 17–20] are often organized in a flat, random, or simple hierarchical manner. The unstructured P2P network replicates the frequently visited data objects among the system, and the queries are often made by using flooding or random walks. Unstructured P2P networks appear to be more effective at finding highly popular content but because of its ad hoc nature and flooding-based queries, the correctness and performance of the routing, the scalability of the system, and the consumption of network bandwidth are all uncertain.

The main challenge is how to locate data requested from the application data space in the P2P identifier space. This problem was solved by using DHT[29–31]. However, it is only applicable for exact key queries. Unstructured P2P networks can be used for more complex queries, but they do not guarantee to locate data. This thesis work has been focusing on multi-dimensional range queries, since it is believed to be a a more generalized form of many other types of queries.

The multi-dimensional range query provides efficient data indexing services in P2P environments. It is the primitive service over which all other complex services can be built. It has many applications in large-scale distributed environments, such as grid computing, publish/subscribe systems, multi-player games, group communication and naming, P2P data sharing, global storage, etc. Many of these applications collect, pro-

duce, distribute, and retrieve information from and into distributed data space. The scale of such data space can be extremely large and its dimensionality varies. The distributing and managing of such data space, even the index part, appears to be a great challenge to the distributed environment. For DHT-based systems, the very features that make them advantageous over others, like load balancing and random hashing, work against the range query. Therefore, we have to consider how data space is partitioned and mapped into the P2P identifier space, and ask if existing P2P overlay structures are good enough to support multi-dimensional range queries properly and efficiently

In addition to the basic efficiency concern of the data query, there are more fundamental issues which have to be addressed properly in a distributed environment, such as scalability, self-organizing, fault tolerance, and load balancing. In order to better resolve these problems, there are two interconnected parts of a process that have to be done cooperatively. The first part of this process is related to how data space is partitioned and mapped into the identifier space in the distributed environment, or in other words, how to distribute data into the P2P system so that specific data objects can be retrieved back with a reasonable computation and communication cost. This is called *application constraint*. The second part of the process is concerned with how to organize peers in the distributed system, or how to design and maintain a certain topology of the P2P system. This is to ensure that properties of the data object (like the semantic meaning of data) and the relationship among data objects (like the data locality), in the original data space preserved in the first part of the work, are manageable and maintained easily in the identifier space. This is known as *system constraint*. The *application constraint* determines what kind of space partitioning strategy should be used, and hence what kind of pattern for the information query can be supported by the system. The *system constraint* determines the performance of the basic routing and maintenance operations, the computation complexity, and if the system is scalable, error-resilient, self-organizing, and load-balancing.

1.2.2 Problem Statement

The *application constraint* determines that both structured and unstructured P2P overlay systems are well suited for the exact key query, and might be adapted to even more complex queries. Neither is able to provide efficient support for the multi-dimensional range query, because the multi-dimensional range query (associative searching) adds additional requirements into both the *application constraint* and the *system constraint* over the P2P overlay layer, which can be briefly stated as follows:

- how to preserve the data locality in the process of data space partitioning, and
- how to maintain the data locality at the P2P overlay layer.

Quad-tree[21, 22], K-d tree[22, 23], Z-order[24, 25], and Hilbert curve[26] are four well recognized works made to meet the *application constraint*. A common feature of these works is that they all employ recursive decomposition in the process of multi-dimensional data space partitioning, which can be best represented by a hierarchical data structure. Unfortunately, there is not very much work that has been done in order to provide comparable support from the P2P overlay layer. Experimental results from existing P2P systems[32–44] show very limited selectivity, and we are not able to obtain a complete view of the system states when conducting range queries in high dimensional data space.

1.2.3 Research Objective

Our research objective is to design and develop an efficient, scalable, and reliable P2P search network that has the inherent nature to accommodate and maintain data localities preserved using recursive decomposition at the higher layer, and is able to draw a complete view of the system state when processing multi-dimensional range queries at different levels of selectivity and in various error-prone routing environments.

1.2.4 Main Contributions

In order to accomplish this research objective, we propose a novel distributed data structure, Hierarchically Distributed Tree (*HD Tree*). We believe that *HD Tree* is an optimal solution for supporting multi-dimensional range queries in the distributed environment among all existing P2P overlay structures, because it has the natural connection to the recursive decomposition employed at the higher layer. This inherent nature makes all important features of a P2P overlay network, like partitioning and mapping, scalability, self-organizing, error-resilience, and dynamic loading balancing, to become a natural part of this novel distributed data structure.

HD Tree is the first distributed data structure proposed in this area that is able to not only adapt the hierarchical data structure into the distributed environment, but also to give a complete view of system states when processing multi-dimensional range queries at different levels of selectivity and in various error-prone routing environments.

The main contributions in this part of work includes the design, development, and implementation of:

- *HD Tree* data structure.
- A set of basic hierarchical routing algorithms in *HD Tree* (see Subsection 2.2.3.1).
- A set of basic distributed routing algorithms in *HD Tree* (see Subsection 2.2.3.1).
- A set of basic *SPeer* routing algorithms in *HD Tree* (see Subsection 2.2.5.3).
- The combined routing algorithm in *HD Tree* (see Subsection 2.2.3.2).
- A set of error-resilient routing algorithms in *HD Tree* (see Subsection 2.2.5).
- Node's Join and Leave algorithms in *HD Tree* (see Subsection 2.2.6).
- *H2H* and *H2D* local adaptation algorithms for a temporary load balancing solution in *HD Tree* (see Subsection 2.2.7.1).

- *D2H* repartitioning algorithm for a global and dynamic load balancing solution in *HD Tree* (see Subsection 2.2.7.2).

1.3 DDM in Cluster-Based Network Environments

1.3.1 Background

In cluster-based network environments, DDM is addressed by using large-scale, real-time, distributed, and interactive simulation systems[46–48]. The concept of DDM can also be found in HLA/RTI under IEEE Standard 1516. It is an advanced data relevance filtering technique used to send data upon request and send exact data required. Many HLA/DDM approaches have been devised in recent years to address this problem[45, 49–51]. In general, these designs can be classified into five categories: (i) the Region-based DDM approach[45, 50]; (ii) the Grid-based DDM approaches, examples of which include the Fixed Grid-based DDM approach[61, 62] and the Dynamic Grid-based DDM approach[45]; (iii) the Agent-based DDM approach[51]; (iv) the Hybrid DDM approach[49]; and (v) the Sort-based DDM approach[63], an example of which is the Grid-filtered Region-based DDM approach[50].

In practice, these implementations demonstrate different focuses regarding the DDM performance issue. In the Region-based DDM approach, a *region* in *routing space* is represented by a set of extents. Exact *matching* (sending the exact data required and only upon request) can be detected by computing these sets of extents of regions published or subscribed exhaustively. However, computing *matching* has to be performed by a central coordinator. In the Grid-based DDM approach, *routing space* is divided into a certain number of grid cells, known as a grid. The ownership of grid cells is distributed to each *federate* (Subsection 3.1.2), and an additional registration process has to be performed because the *publisher/subscriber* may not be the owner of the cell. In the Grid-based system, computing *matching* is performed by the owner of cells, and *matching* is detected

based on each unit of the cell. Therefore, the central coordinator is not necessary. However, exact *matching* can not be achieved.

If we compare these two types of DDM solutions, the Region-based approach achieves exact *matching* with considerable computation load and communication overhead at the central coordinator, while the Grid-based DDM approach sacrifices accuracy and distributes load among simulation nodes in order to reduce processing time and communication overhead at the central coordinator. In order to compensate for the loss of *matching accuracy* introduced by the Grid-based system, the Agent-based and Grid-filtered Region-based approaches have been proposed. In these two approaches, a data filtering mechanism has been used to filter out unnecessary data transmitted at the sender side and a higher *matching accuracy* can be obtained when compared to the grid-based DDM system, with a lower computation load and communication overhead than the Region-based DDM system[50, 51, 58, 59].

When we examine all existing DDM implementations proposed up until now, a common characteristic that has never drawn too much attention is that they have all been devised to reduce communication costs (*DDM messages*[45]) and improve processing performance (*DDM time*[45] and Multi-cast Group allocated[45, 60]) without any knowledge of how much data could be generated and transmitted by a DDM approach. In other words, they have been designed to achieve efficiency blindly.

1.3.2 Problem Statement

All existing DDM approaches were designed in order to accommodate the allocation, control, and management of data generated in each *timestep* (3.1.4.1) of a simulation properly. Although they achieved very good performance at some point, they can never reach the optimum. This is because existing DDM approaches manipulate data generated in each *timestep* without evaluating the size of a distributed simulation. We believe that an appropriate DDM solution must have clear knowledge of how much data should

be managed locally at each *federate* (Subsection 3.1.2) and how much data should be distributed among the *federation* (Subsection 3.1.2).

1.3.3 Research Objective

Throughout this part of thesis, we propose a brand new DDM approach - the Adaptive Grid-based (AGB) DDM approach. This novel DDM approach is built over the existing Grid-based DDM system, and it is designed to optimize the performance of DDM service based on the evaluation of the simulation size. The objective of this part of the work is to predict the average amount of data that can be generated for each specific input of a simulation and to conduct the simulation running in a way that best meets each specific simulation requirement.

1.3.4 Main Contributions

The novel AGB DDM approach consists of two adaptive control schemes: the Adaptive Resource Allocation Control (ARAC) scheme and the Adaptive Transmission Control (ATC) scheme. The first control scheme is derived from a static probability model. We call this model *matching model* and claim that it is static because it features the steady-state probability of a grid cell in each *timestep*[45] of the simulation and can be used to evaluate the average amount of local and distributed storage that have to be allocated for a running simulation. The second control scheme is derived from a dynamic probability model: the *switching model*. This model is a dynamic model because it denotes the state-transition probability of a grid cell between each two consecutive *timesteps* of the simulation and is used to evaluate the average amount of DDM messages that have to be transmitted to each other *federate* in each *timestep*.

The main contributions in this part of the work includes the design, development, and implementation of the:

- AGB DDM approach.
- *Matching model* (see Subsection 3.4.3).
- *Switch model* (see Subsection 3.5.3).
- ARAC scheme (see Subsection 3.4.3.5).
- ATC scheme (see Subsection 3.5.3).

1.4 Thesis Organization

In this chapter, we introduced background information about DDM in both P2P overlay networks and cluster-based network environments, and discussed the main challenging problems that have been found in each area. Our research objectives and main contributions to DDM in each network environment are stated independently. The remainder of this thesis is structured as follows: Chapter 2 is dedicated for DDM in P2P overlay networks. We begin with an overview of all related works in this area in Section 2.1. Then, in Section 2.2, our new approach of DDM in P2P overlay networks follows. We present our experimental results and analysis in the last section (Section 2.3) of this chapter. Similarly, Chapter 3 is solely used for the DDM in cluster-based network environments. We introduce basic concepts of DDM in HLA/RTI in Section 3.1 and all related works in Section 3.2. In Section 3.3, we discuss some fundamental problems in grid-based DDM systems. Our new AGB DDM approach in cluster-based network environments is described in Section 3.4 and 3.5. In Section 3.4, we build a *matching model* for the ARAC scheme, and in Section 3.5, we build a *switching model* for the ATC scheme. The experimental results and analysis are presented in Section 3.6. At last, we summarize our work in both network environments and present conclusions and future works in Chapter 4.

Chapter 2

DDM in P2P Overlay Networks

2.1 Related Works

In P2P overlay networks, our recent interest is in multi-dimensional range queries. We believe it is a more generalized form of many other types of queries. Some P2P applications inclue grid computing, publish/subscribe systems, multi-player games, group communication and naming, P2P data sharing, global storage, etc. DHT-based systems, the very features that make them good, like random hashing and load balancing, work against range queries. We have to consider how data space is partitioned and mapped into identifier space and examine if our existing P2P overlay structures are good enough to support multi-dimensional range queries properly and efficiently.

2.1.1 Multi-dimensional Data Space Partitioning and Mapping

The main concern in multi-dimensional space partitioning is all about how to preserve data localities. There are many works that have been done so far. Among all of them, Quad-tree[21, 22, 28], K-d tree[22, 23], Z-order[24, 25], and Hilbert curve[26] are four

P2P Systems	Space Partitioning	Underlying Architecture	Direct Mapping
DHT-based	DHT	CAN / Chord / ...	No
SkipNet	Naming sub-tree	Skip graphs	No
P-Grid	Virtual distributed search tree	Flat graphs	No
Mercury	Random sampling	Ring	No
SCARP	Z-order / Hilbert SFC	Skip graphs	No
MURK	K-d tree	d-torus (CAN)	No
ZNet	Z-order / Quad tree	Skip graphs	No
Skipindex	K-d tree	Skip graphs	No
Squid	Hilbert SFC	Ring (Chord)	No
SONAR	K-d tree	d-torus (CAN)	No
MAAN	Locality-preserving hashing	Ring (Chord)	No
BATON	Binary search tree	Balanced binary tree	Yes
VBI-Tree	SFC	Balanced binary tree	Yes

Table 2.1: Comparison of P2P Overlay Systems I

well-known proposals. Although these works provide different solutions, they all share a common feature: using recursive decomposition for partitioning, which is a tree structure in nature.

Quad-tree and K-d tree provided solutions using hierarchical data structures, while Z-order and Hilbert curve provided solutions by linearizing points in multi-dimensional space using the space filling curves. The performance of these decomposition schemes are all bound by $O(lg(n))$. It has been found that the processing of Z-order and Hilbert curve has a natural connection with Quad-tree.

In addition to these standard decomposition schemes, locality-preserving hashing[42] was proposed in order to retain the range query in the DHT-based systems; however, random sampling[41] is another effort that does not scale well. No matter which decomposition scheme is used, two direct observations about partitioning in the multi-dimensional data space can be observed:

Fact 2.1.1. *Data localities expand exponentially as the dimensionality of data space increases.*

Because the total number of neighboring range data in p dimensional data space at the first order of decomposition is $k = 2^p$.

Fact 2.1.2. *Data localities extend exponentially as the order of recursive decomposition increases.*

Because the total number of neighboring range data in p dimensional data space at the $d - th$ order of decomposition is k^d.

2.1.2 Corresponding Overlay Network Support

How to preserve data localities in data space partitioning is the *application constraint* enforced by the range query. Solutions include using recursive decomposition and the locality-preserving hashing. Correspondingly, the *system constraint* requires that the underlying P2P network layer has to possess a certain inherent nature in order to accommodate and maintain data localities with exponentially expanding and extending rates. Existing P2P overlay networks, either DHT-based or unstructured, are originally designed in order to meet a specific demand, such as, the exact key query. The very features that make them good, like random hashing and load balancing, or replicating the popular contents, work against the range query. Our solution is to use a tree like structure without any doubt; because a tree structure has the natural connection to the recursive decomposition scheme employed at a higher layer, and it supports direct mapping from data space into P2P identifier space.

Although the work in multi-dimensional space partitioning can be traced back to the 1960s, and the results and performance have been widely accepted ($O(lg(n))$), little effort has been made in the study of the corresponding P2P overlay network to provide comparable support for the multi-dimensional range query. Table 2.1 shows a comparison of current P2P systems supporting the multi-dimensional range query.

Underlying Architecture	P2P Systems	Routing Table Size	Routing	Join/Leave	Load Balancing	Fault Tolerance
Ring	Chord Squid MAAN Mercury	$O(\log(n))$ [3]	$O(\log(n))$ [3]	$O(\log^2(n))$ [3]	Static[3]	Background stabilization routine [3] Problem remains open [38]
d-torus	CAN MURK SONAR	$O(d)$ [2]	$O(dn^{1/d})$ [2]	$2d$ [2]	Static [2]	Background stabilization routine [2] Problem remains open [38]
Skip Graphs	SkipNet ZNet SCARP Skipindex	$O(n\log(n))$ [38]	$O(\log(n))$ [37]	$O(\log(n))$ [37]	Static [37, 38]	Inherently support [37] Problem remains open [38]
Flat Graph	P-Grid	N/A	$O(\log(n))$ [40]	N/A	Dynamic [40] Static [38]	Replication[40]
Balanced Binary Tree	BATON VBI-Tree	3 [43, 44]	$O(\log(n))$ [43, 44]	$O(\log(n))$ [43, 44]	Dynamic [43, 44]	Redundant links [43, 44]

n: total number of peers; d: d-torus; k: k-ary tree

Table 2.2: Comparison of P2P Overlay Systems II

In this table, as we have discussed in the previous section, we do not consider DHT-based systems. SkipNet[39] and P-Grid[40] were designed and have been claimed to support the range query, but have no experimental results that can be found so far. Mercury[41] uses random sampling for data distribution, and its experimental results show a maximum of 0.1 percent selectivity in a three dimensional data space (10% at each dimension). MAAN[42] employs locality-preserving hashing, and the maximum selectivity is about 1 percent in a two dimensional data space (10% at each dimension). Other systems, like SCARP[32], MURK[32], ZNet[33], Skipindex[34], Squid[35], and SONAR [36], use recursive decomposition for partitioning, have underlying architectures that are either adapted structures from DHT-based systems, like Chord, CAN, or Skip Graphs[37]: the only structure proposed for the range query. No system has direct mapping. They all have to manage a tree like structure internally. Their experimental results show very limited selectivity. We are unable to have a complete view of how the P2P system behaves in high dimensional range queries with a wide range of selectivity. However, BATON[43] and VBI-tree[44] are the only two systems that have been proposed with a tree like structure at the overlay layer. High dimensional range queries (up to 20) with the same selectivity at each dimension are evaluated in these two systems, but without any notion regarding the size of selectivity.

Nevertheless, one drawback that has been commonly shared in experimental results from all these systems is worth mentioning: when we increase the dimensionality of data space without varying selectivity at each dimension, the overall selectivity has been exponentially decreased, and the comparison of range queries presented in this way is not very convincing.

On the other hand, comparisons of basic distributed operations in existing P2P overlay networks are summarized in Table 2.2. These comparisons are made by grouping P2P systems that have the same underlying architecture. This table shows that the routing table size in tree based structures does not depend on the network size, and their basic distributed operations are all bound by $O(log(n))$. Moreover, it is also very

straightforward to conduct dynamic load balancing operations in a tree based structure. In comparison, P2P systems built over a Ring, d-torus, or Skip Graphs assume each peer has the same capacity to participate in the system wide operations; although P-Grid proposed a dynamic load balancing scheme in [40], it is considered to be static in [38]. However, most of these P2P overlay structures were not built in favor of supporting fault tolerance: P2P systems built over a Ring or d-torus depend on a background stabilization routine to maintain the correctness of the routing table, but the challenge is when and with what frequency the stabilization routine needs to run[1]. In the flat graph structure, which has been commonly adopted by unstructured P2P networks, replication is so far the best solution to deal with node's failures; and tree based structures need to build redundant links in order to survive error-prone environments. Perhaps Skip Graph is the only structure that claims to be able to support fault tolerance inherently[37], but it is considered as an open problem according to [38].

We conclude that multi-dimensional range queries are sensitive to underlying topology. Although it is feasible to adapt existing P2P overlay structures that were originally designed for the exact key query, their inherent nature shows no consistency in accommodating data localities that have been well preserved using decomposition schemes employed at a higher layer. This inconsistency not only involves a considerable routing and maintenance cost, but also, at the architectural level, makes a simple, scalable, and reliable P2P solution for multi-dimensional range queries very difficult, or even impossible.

2.1.3 Why HD Tree?

A simple hierarchical data structure has many difficulties surviving in a distributed environment. However, because of its natural connection to the data space decomposition, my thesis work is intended to adapt the hierarchical data structure to the distributed

environment. In order to cope with these difficulties, we propose a new data structure, called Hierarchically Distributed Tree (*HD Tree*). The add-on distributed structure in the *HD Tree* is built over a complete tree by making a limited number of extra connections, so that, each node in the system can be reached by any other nodes at the same depth via not only the hierarchical structure, but also the distributed structure, and with comparable routing performance. In *HD Tree*, there are multiple routes between any two nodes; each node is able to behave hierarchically or distributively depending on which situation is necessary; the routing table size is determined by the maximum number of children (k) at each node; the performance of distributed routing is bound by $O(lg(n))$, where n is the total number of nodes; and the basic maintenance operations in *HD Tree* (Join/Leave) are also bound by $O(lg(n))$. We believe *HD Tree* is more capable of handling multi-dimensional range queries than BATON and VBI-tree, because *HD Tree* can be built over any $k-ary$ tree ($k = 2^p$) structure instead of a binary tree alone, and it is built to support fault tolerance inherently.

2.2 Proposal of New Approach

2.2.1 HD Tree

2.2.1.1 Notations in Tree

All of the traditional notations for a tree can be used for *HD Tree*, and we also introduce a 5-tuple of letters (k, h, d, n, n_d) to denote the system parameters of the tree structure. In this set, k refers to $k-ary$, h refers to the height of a tree, d refers to the depth in a tree, n is the total number of nodes, and n_d is the total number of nodes at d.

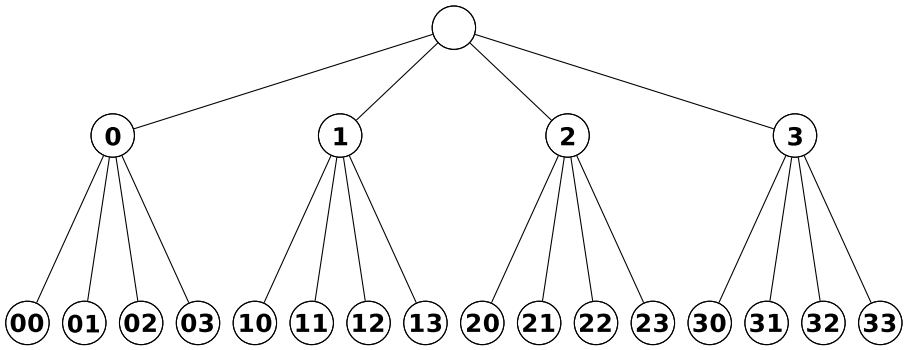


Figure 2.1: Coding of a Complete $4-ary$ Tree

2.2.1.2 Definition of HD Tree

HD Tree is a cyclic graph built over a complete $k-ary$ tree. It consists of two distinctive structures: the hierarchical structure and the distributed structure. The hierarchical structure is the inherent feature carried by its basic tree structure, it covers all nodes in *HD Tree*; and provides the complete support for all hierarchical operations over the entire *HD Tree*. While the distributed structure spans each two adjacent sets of nodes, the nodes at depth $i-1$ $(1 < i \leq h)$ and the nodes at depth i, the basic distributed operations are related only to these two sets of nodes.

We give a formal definition of *HD Tree* as follows:

Definition 2.2.1. *(HD Tree) 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 in each of k sub-trees of the root at depth i if that node exists; such that, each node at depth $i-1$ is able to reach any other nodes at the same depth in at most $2(i-1)$ hops time via nodes at depth of $i-2$ and $i-1$ only.*

An *HD Tree* is a full *HD Tree* if its complete tree is fully filled, and a full *HD Tree* of height less than h can always be found in an *HD Tree* of height h.

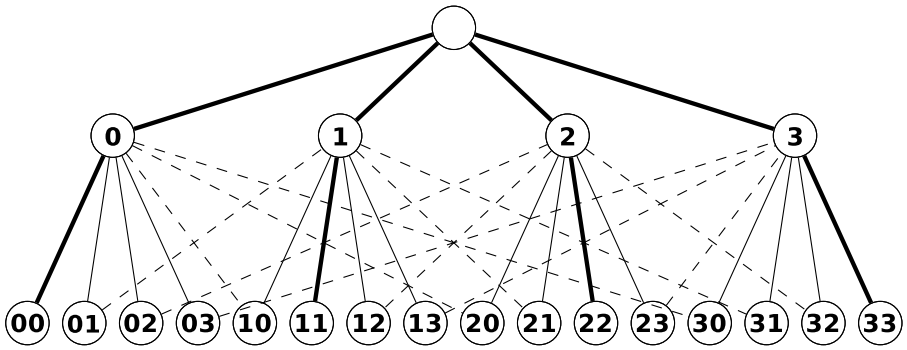


Figure 2.2: Constructing a $4-ary$ HD Tree

One way of constructing such an *HD Tree* is based on the coding of an ordered complete tree. Figure 2.1 is an instance of such tree, and it is connected by following the prefix coding strategy:

- *Root has null code.*
- *The code of a parent node is always the prefix code of its children.*
- *The prefix coded children are ordered by the right most digit from* 0 *to* $k-1$.

accordingly, conducting the connection of nodes via extra links is based on the suffix coding strategy, and it can be described as follows:

- *Root has null code.*
- *The code of a parent node is always the suffix code of its extra children*
- *The suffix coded children are ordered by the left most digit from* 0 *to* $k-1$.

Figure 2.2 and 2.3 demonstrate this constructing scheme. In these figures, solid links and dashed links represent connections built by following prefix coding and suffix coding respectively, while bold links indicate the intersecting part of these two coding strategies. It can be observed that this constructing scheme can be carried out at any depth of a complete tree structure.

The equivalent tree of an *HD Tree* is the complete tree (as shown in Figure 2.1) over which this *HD Tree* is constructed, and it is the base tree of *HD Tree*.

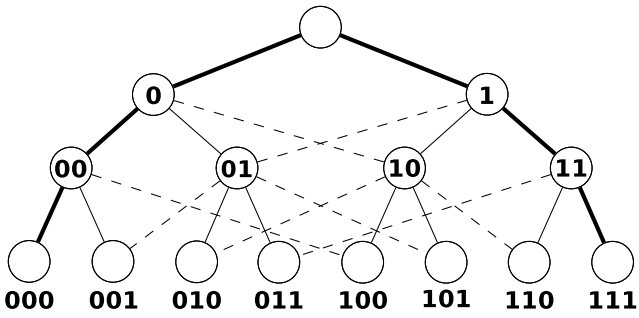


Figure 2.3: Constructing a $2-ary$ HD Tree

2.2.1.3 HD Table

Figure 2.2 is only able to depict the structure of an *HD Tree* with very limited height. In fact, it is very difficult to give a clear picture of the distributed structure of *HD Tree* in this way. In order to better understand this new data structure and to ease our study, we use an *HD tables* to describe the distributed structure. We start from introducing the coding table of an ordered tree.

Figure 2.4 (a) shows a coding table which sorts prefix coded nodes at depth 2 within the same column. We call this coding table *H table*, because it describes the original hierarchical structure of a $4-ary$ *HD Tree* (Figure 2.2). Accordingly, the code table shown in Figure 2.4 (b) sorts suffix coded nodes at depth 2 within the same column. It depicts the distributed structure built over the complete tree, and it is called *D Table*. In both tables, the same column nodes have the same parent. In *H table*, it is the parent from the hierarchical structure; while it is the parent from the distributed structure in the other.

HD table consists of an *H table* and a *D table*: the *H table* is placed over the *D table* (as shown in Figure 2.4 (c)). The *HD table* is used to represent the relationship of nodes at the same depth. All siblings are aligned in the same column. The prefix coded siblings (*HSiblings*) are in the *H Table*, while the suffix coded siblings (*DSiblings*) are in the *D Table*. The parent of each column is *HParent* of *HSiblings* in that column, and *DParent* of *DSiblings* in the same column. Correspondingly, these *HSiblings* and

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

(a) H table

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

(b) D table

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

(c) HD table

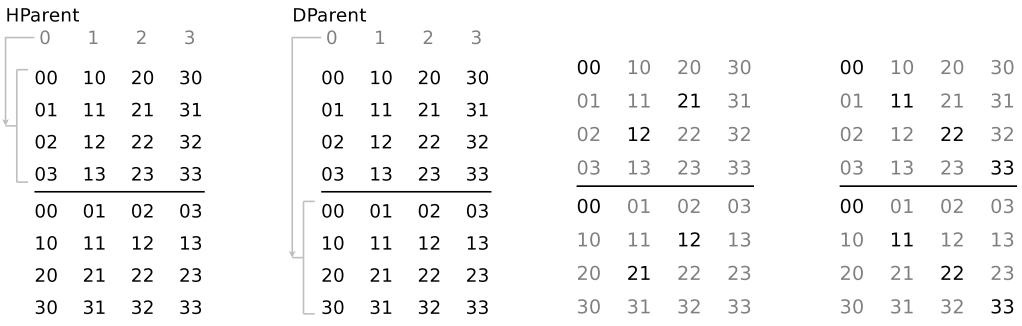
Figure 2.4: H table, D table, and HD table

DSiblings are *HChildren* and *DChildren* of the parent of that column. The sequence number of *HSiblings* is the right most digit of the code, and the sequence number of *HSiblings* is the left most digit of the code.

We believe there are multiple techniques to build different kinds of *HD Tree*, the *HD Tree* shown in Figure 2.2 is only one of them. By the direct observation from *HD table*, the technique used to build this type of *HD Tree* is nothing but splitting the same column elements in the *H table* into different columns but within the same row in *D table*, and we call it *HD Tree* generated by the *splitting rule*. Because this type of *HD Tree* appears to have our most favorite performance advantages and the global awareness in distributed computing. Therefore, in this part of thesis, the notation of *HD Tree* is used to represent an *HD Tree* generated by the *splitting rule* by default.

2.2.2 Properties of HD Tree

By introducing extra links into a complete tree structure, an internal node has not only prefix coded *HParent* and *HChildren* (as it does in an equivalent tree structure), but also suffix coded *DParent* and *DChildren*. It is necessary to define more relationships for a node in *HD Tree*.



(a) HParent (b) DParent (c) Mapping Entry (d) SPeer

Figure 2.5: HParent, DParent, Mapping Entry, and SPeer

2.2.2.1 Notations in HD Tree

According to Definition 2.2.1(*HD Tree*), although building an *HD Tree* over a complete tree does not change the system parameter set (k, h, d, n, n_i) of a tree, traditional notations for a tree need further clarification in order to adapt to the new structure. The *HD table* defines not only the hierarchical relationship but also the distributed relationship among nodes in an *HD Tree*. Nodes directly given in an *HD table* are the children set nodes at depth $d = i$. They are *DPeers* (Distributed Peers) of each other. Each *DPeer* has one and only one entry in both of the *H* and *D tables*. On the other hand, all nodes at depth $d = i - 1$ are *HPeer* (Hierarchical Peers) of the nodes at depth $d = i$. They compose the parent set of *DPeers*, and corresponding to each column in an *HD table*.

An *HPeer* could be either the hierarchical parent (*HParent*) or the distributed parent (*DParent*). It is the *HParent* of its column elements in *H table*, and the *DParent* of its column elements in *D table* (Figure 2.5 (a) and (b)). Moreover, each element in an *HD table* has two entries, the one in the *H table* is called the hierarchical location, and the other in the *D table* is called the distributed location. An element is called *HChild* if it is in the *H table*. It has *HSiblings* in the *H table* and *DSiblings* in the *D table*, and both of these siblings are in the same column of the *HD table*; on the other side, an element is called *DChild* if it is in the *D table*, and similarly, it has *HSiblings* in the *H table* and *DSiblings* in the *D table*. A *DPeer* could be a *HChild*, a *DChild*, a *HSibling*,

or a *DSibling*, depending on which context is applied.

2.2.2.2 HCode and DCode

HCode (Hierarchical Code) is the code used in both *HD Tree* and *HD table*. *HCode* is very helpful in locating a node in its hierarchical structure. However, it appears not so straightforward if we need to know the distributed location of a node in *HD Tree*. By direct observation from *HD table* (Figure 2.5 (c)), each element in *D table* has an unique *mapping entry* in its *HSiblings*. If this element is the $i-th$ *DSibling* in the column, then its *mapping entry* is the $i-th$ *HSibling* in the same column.

Definition 2.2.2. (*DCode*) *DCode is HCode of the mapping entry of a node at its distributed location.*

DCode is an alternative identifier of a node. It is used to locate a node in its distributed structure hierarchically.

2.2.2.3 SPeers

An *HD Tree* generated by the *splitting rule* has some special nodes at each depth which may not be in favor of the distributed environment.

Definition 2.2.3. (*SPeer*) *A SPeer is a self-mapping DPeer whose mapping entry is itself.*

A *SPeer* has the same value in each digit of the code. Because of this reason, it has the same *DParent* as its *HParent*, and a simple fact can be obtained from the direct observation:

Fact 2.2.4. *(SPeer) There are k SPeers at each depth from 1 to h in a full HD Tree.*

If a *SPeer* is the leaf node in an *HD Tree*, in case of the single parent's failure, it will be isolated from the system. Therefore, a *SPeer* is considered to be relatively weaker compared to other non-*SPeers*. Figure 2.5 (d) shows 4 *SPeer*s at depth 2 in a $4-ary$ *HD Tree*.

2.2.2.4 System Parameters

HD Tree is built over a complete $k-ary$ tree. Although extra links have been introduced during the constructing process to support the distributed features at each depth, system parameters (k, h, d, n, n_i) from the hierarchical tree structure remains the same. However, according to Definition 2.2.1(*HD Tree*), by introducing extra links to all internal nodes, each of them has k extra children (*DChildren*) from each k sub-trees of the root in addition to the existing k children (*HChildren*). However, the *SPeer* parent node has the same *SPeer* child in both *HChildren* and *DChildren* set, and the root has the same *DChildren* set as its *HChildren* because *SPeer* is a self-mapping *DPeer*. Therefore,

Fact 2.2.5. *In a full HD Tree, the root has k children, each internal node has $2k$ children if it has no SPeer child, and $2k-1$ children otherwise.*

and

Fact 2.2.6. *The total number of links in a full HD Tree is $2(n-1)-kh$.*

2.2.3 Basic Routing Strategies

2.2.3.1 Hierarchical and Distributed Routing Mechanism

In accordance with hierarchical and distributed structures in *HD Tree*, there are two different types of routing. Hierarchical routing continuously forwards the routing request towards root or leaf nodes, while distributed routing forward the routing request to

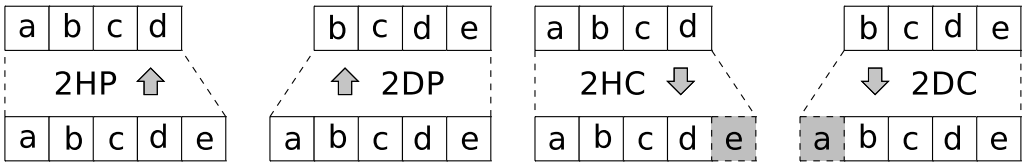


Figure 2.6: Basic Hierarchical Operations

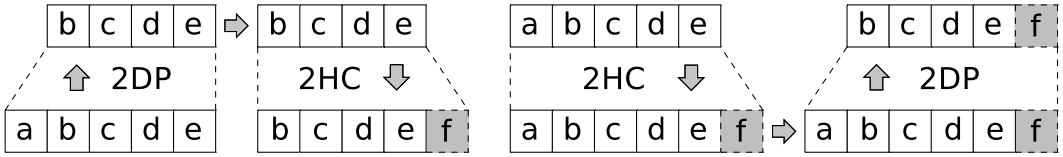


Figure 2.7: Basic Distributed Operation - D2H

a parent node and a child node alternately. Both of these two routing mechanisms are distributive operations in *HD Tree*. One reaches the destination by crossing many different depths, and the other does so by traversing only two adjacent depths in *HD Tree*.

Basic hierarchical operations The basic hierarchical operation is a single hop time operation shown in Figure 2.6. In the ideal routing environment, the hierarchical operation is mainly used to forward routing requests towards the destination depth. There are four basic hierarchical operations. *2HP* (send to *HParent*) or *2DP* (send to *DParent*) passes the routing request to a parent node. *2HP* truncates a digit in the current *HCode* at the right most side, while *2DP* truncates a digit in the current *HCode* at the left most side. On the contrary, *2HC* (send to *HChild*) or *2DC* (send to *DChild*) passes the routing request to a child node. Normally, *2HC(i)* and *2DC(i)* are used to denote ¨send to the *i-th HChild* or *DChild*¨. *2HC(i)* extends one more digit i to the current *HCode* at the right most side, while *2DC(i)* extends one more digit i to the current *HCode* at the left most side.

Hierarchical routing is simply a series of repeated operations of all or some of these four. The details regarding hierarchical routing algorithms pertain to a simple adapted version from the existing tree routing algorithm.

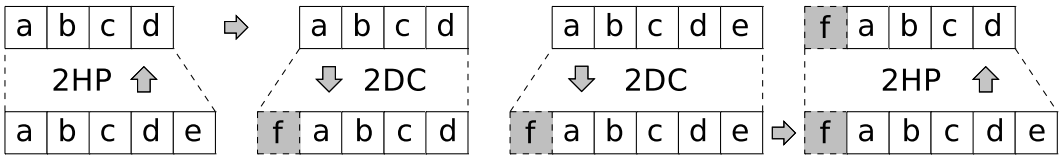


Figure 2.8: Basic Distributed Operation - H2D

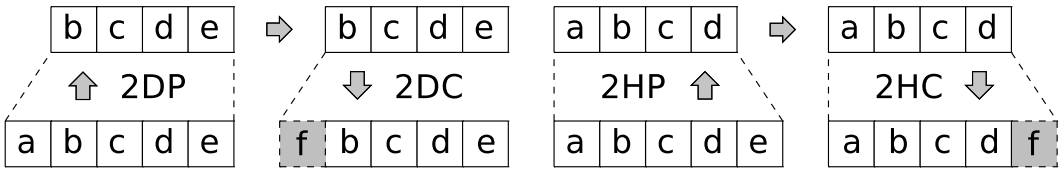


Figure 2.9: Basic Distributed Operations - D2D & H2H

Basic distributed operations Unlike the hierarchical operation, the basic distributed operation is a 2-hop time operation. It consists of two basic hierarchical operations which always involve a parent node and a child node. In the ideal routing environment, the distributed operation is mainly used to forward a routing request towards the destination node at the same depth.

There are four basic distributed operations. *D2H* (*2DP* and *2HC*) and *H2D* (*2HP* and *2DC*) are digit-shift operations (as shown in Figure 2.7 and 2.8). *D2H* forwards the routing request to an *HSibling* via *DParent* or *HChild*. It results in a one digit-shift in the current *HCode* towards the right, while *H2D* forwards the routing request to a *DSibling* via *HParent* or *DChild*; it results in a one digit-shift in the current *HCode* towards the left. On the other hand, *D2D* (*2DP* and *2DC*) and *H2H* (*2HP* and *2HC*) are digit-replacement operations (as shown in Figure 2.9). *D2D* forwards the routing request to a *DSibling* via *DParent*, and it replaces a digit in the current *HCode* at the left most side; while *H2H* forwards the routing request to an *HSibling* via *HParent*, and it replaces a digit in the current *HCode* at the right most side.

Distributed routing is essentially a series of such digit operations that are continuously applied towards one direction. The *D2H* or *H2D* routing is a series of *D2H* or *H2D* operations, and the *D2D* or *H2H* routing is a series of *D2H* or *H2D* operations followed

by a *D2D* or *H2H* operation at last. During each operation of distributed routing, if new digit is chosen properly by following every digit value in the destination code sequentially, the destination is guaranteed to be reached in, at most, d steps of such 2-hop time operations. Therefore, in the routing operations towards the destination node, we often use *2HC[i]* and *2DC[i]* to represent ¨send to an *HChild or DChild* whose sequence number equals to the *i-th* digit of the destination code¨.

Performance analysis In distributed routing operations, all siblings of the destination are 2 hops away, all siblings of the siblings of the destination are 4 hops aways, and so on. These $2i$ hops away sets of siblings might have some sub-codes in common with the destination. By choosing the maximum sub-code properly, 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 2.2.7. *The average case time complexity of distributed routing* $\mathcal{T}_{avg}(DR) \leq 2d - \frac{2}{k-1} + \frac{2}{k^d(k-1)}$.

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 *D2H* routing 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* only,

$\Rightarrow$ the probability of $D \in J_0$ 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 $|J_1 - J_0| = k - 1$.

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

Figure 2.10: D2H Routing When Source is a SPeer

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

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

⇒ the probability of $D \in J_2 - J_1$ 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 $|J_i - J_{i-1}| = k^{i-1}(k-1)$.

⇒ the probability of $D \in J_i - J_{i-1}$ 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 $|J_d - J_{d-1}| = k^{d-1}(k-1)$.

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

⇒ $\mathcal{T}_{avg}(D2H_{SPeer}) = \frac{2(k-1)}{k^{d+1}} \sum_{i=1}^{d} ik^i$

⇒ $\mathcal{T}_{avg}(D2H_{SPeer}) = 2d - \frac{2}{k-1} + \frac{2}{k^d(k-1)}$

Part 2:

If a *SPeer* is the source node, it has the maximum average time complexity in *D2H* routing because $J_{i-1} \subset J_i$ is always true for $0 < i < d$ and it leads to the worst average case time complexity compared to any other non-*SPeer* source nodes at the same depth. Figure 2.10 and 2.11 demonstrate the difference between the *SPeer* and non-*SPeer* in

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 = {120, 121, 122, 123}; J_2 = {200, 201, 202, 203}∪ {210, 211, 212, 213}
∪ {220, 221, 222, 223} ∪ {230, 231, 232, 233}

Figure 2.11: D2H Routing When Source is a Non-SPeer

D2H distributed routing.

The analysis for *H2D*, *D2D*, and *H2H routing* holds the same result as what we did for *D2H.* Therefore, by Part 1 and Part 2

$\mathcal{T}_{avg}(DR) \leq 2d - \frac{2}{k-1} + \frac{2}{k^d(k-1)}$ □

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

Fact 2.2.8. *Distributed routing achieves better routing performance than its equivalent tree*[1] *routing.*

However, each of these four distributed routing mechanisms alone does not have very much performance advantage over the equivalent tree routing. The significance of distributed routing lies in the combination of four distributed routing algorithms (*X2X* routing), and the combination of distributed routing and hierarchical routing. With the combination of all these routing mechanisms, the longest common sub-code between the source and destination pair can always be used to optimize the routing process, and more

[1]The equivalent tree of an *HD Tree* is the complete tree over which this *HD Tree* is constructed.

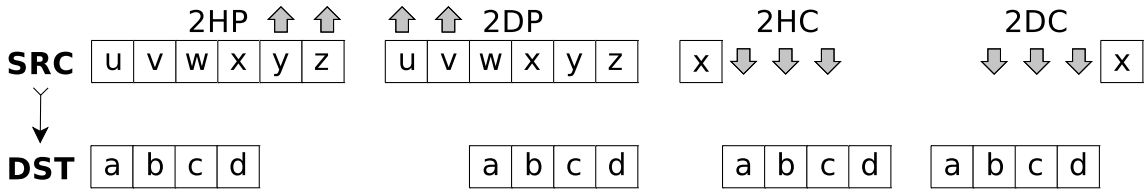


Figure 2.12: Hierarchical Routing without Match

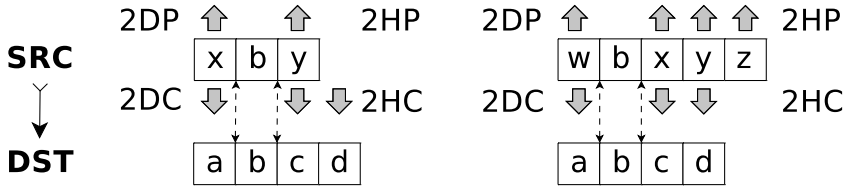


Figure 2.13: Hierarchical Routing with Match

routing options can be considered in order to build a robust and error-resilient routing mechanism. In addition, traffic in *HD Tree* can be easily monitored and manipulated in order to avoid certain heavily loaded nodes.

2.2.3.2 Combined Routing Mechanism

DROCR algorithm Routing in *HD Tree* is nothing but a digit operation which occurs only at the left most or right most side in the current *HCode*. By properly arranging these digit operations towards the destination code, the destination node can always be reached within $2h$ hops time at worst. However, better performance may be obtained if there exists a *match*. A *match* is a common sub-code between the current *HCode* and the destination code, and we let i and j $(i < j)$ be location index of the current *match* in the destination code. We propose a *DROCR* algorithm (Distributed Routing Oriented Combined Routing) which tries to use the longest existing *match* and distributed routing operations as much as possible. Hierarchical routing operations are applied only when the current node and destination are not at the same depth or distributed routing operations can not be applied.

In the ideal routing environment, hierarchical operations are used to forward the

routing request towards the destination's depth. If there is no *match* between the current node and the destination, there are two options to truncate the current *HCode* towards the destination depth, *2HP* and *2DP*, and two options to extend the current *HCode* towards the destination code, *2HC* and *2DC* (as shown in Figure 2.12). *2HC* or *2DC* has to be applied in order to generate a *match* starting from either the right most side or the left most side of the current *HCode*, and this can be done by choosing the child node whose sequence number equals to the digit at the left most side or the right most side of the destination code. In procedure "*HR_none_match()*", only one of two options at each routing direction is applied.

1. `if` $d_{SRC} >= d_{DST}$`,` *2HP*`;`
2. `if` $d_{SRC} < d_{DST}$`,` *2DC[0]*[2]`;`
3. `return success;`

On the other hand, if a *match* can be found between the current node and the destination (as shown in Figure 2.13), procedure "*HR_match()*" is called:

1. `if` $d_{SRC} >= d_{DST}$
   (a) `if` *2HP* `steps > 0`[3]`,` *2HP*`; else`
   (b) `if` *2DP* `steps > 0,` *2DP*`;`
2. `if` $d_{SRC} < d_{DST}$
   (a) `if` *2HP* `steps == 0 &&` *2HC* `steps > 0`[4]`,` *2HC[j+1]*[5]`; else`
   (b) `if` *2DP* `steps == 0 &&` *2DC* `steps > 0,` *2DC[i-1]*`;`
3. `return success;`

In this procedure, *2HP* or *2DP* is applied to truncate the current *HCode* towards the *match*, and *2HC* or *2DC* is applied to extend the existing *match* towards the destination code.

[2] *send to a DChild whose sequence number equals to the 0-th digit of the destination code.*
[3] *2HP operation will not truncate the current match (as shown in Figure2.13).*
[4] *2HC operation will not break the current match (as shown in Figure2.13).*
[5] *send to an HChild whose sequence number equals to the (j+1)-th digit of the destination code.*

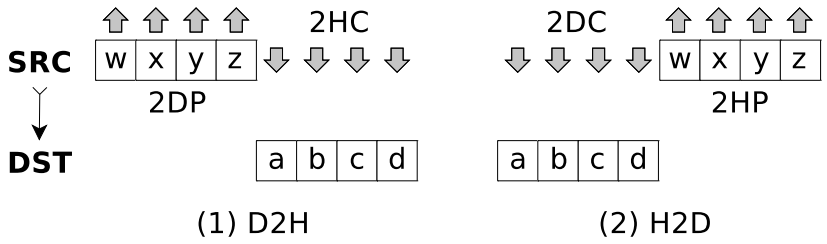


Figure 2.14: Distributed Routing without Match

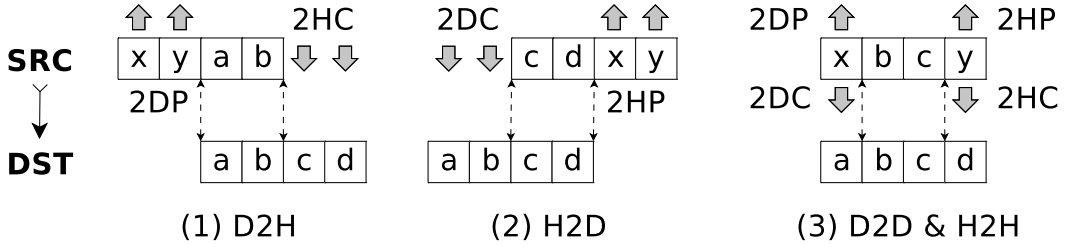


Figure 2.15: Distributed Routing with Match

When a node at the destination's depth receives a routing request, distributed operations may be applied to extend an existing *match* (shown in Figure 2.15) towards the destination node. In this case, we use *X2X* routing to choose one from four distributed routing procedures: *D2H(*$i = 0$*, j), D2D(*$i = 1$*, j), H2D(i,* $j = 0$*),* and *H2H(i,* $j = d_{DST} - 2$*)*, depending on location index i and j of the longest *match* in the destination code.

If no distributed operations can be applied to the existing *match*, the hierarchical operation can always be used instead, and procedure "*DR_other_match()*" will be called:

1. `if` *2HP* `steps > 0,` *2HP*`; else`
2. `if` *2DP* `steps > 0,` *2DP*`; else`
3. `if` *2HP* `steps == 0 &&` *2HC* `steps > 0,` *2HC[j+1]*`; else`
4. `if` *2DP* `steps == 0 &&` *2DC* `steps > 0,` *2DC[i-1]*`;`
5. `return success;`

In this case, the number of choices for the routing operation varies depending on the location of index of the longest *match* in both the current node and the destination

node.

If no *match* exists, there are at least two routing options: a *D2H(*$i = 0$, $j = 0$*)* operation starting from the left most side of the destination code and an *H2D(*$i = d_{DST} - 1$, $j = d_{DST} - 1$*)* operation starting from the right most side of the destination code (as shown in Figure 2.14). In fact, it is always possible to generate a *match* by calling other distributed routing procedure like *D2D(*$i = 1$, $j = 1$*)*, or *H2H(*$i = d_{DST} - 2$, $j = d_{DST} - 2$*)*.

Performance analysis According to the *DROCR* algorithm, performance of the combined routing is determined by the longest *match*, and the depth of source and destination pair. If no *match* can be found, the worst case time complexity is applied. Otherwise, the maximum saving is guaranteed to be achieved by applying the *DROCR* algorithm.

Theorem 2.2.9. *Let* $S = s_0...s_{d_{SRC}-1}$ *and* $D = c_0...c_{d_{DST}-1}$ *be the source and destination pair in HD Tree, and* m *be the length of the longest match between* S *and* D*, then,* $\mathcal{T}(DROCR) = d_{SRC} + d_{DSR} - 2m$.

Proof. we prove this claim in the three cases below:

(1) No *match* can be found ($m = 0$).

If $d_{SRC} = d_{DSR} = d$, then $\mathcal{T}_1 = d_{SRC} + d_{DSR}$.

If $d_{SRC} < d_{DST}$, by *DROCR* algorithm, it takes $d_{DST} - d_{SRC}$ hops for *2HC or 2DC* operations, and $2d_{SRC}$ hops for *D2H* operations, as shown below:

$s_0...s_{d_{SRC}-1} \xrightarrow{2HC} s_0...s_{d_{SRC}-1}c_0...c_{d_{DST}-d_{SRC}-1} \xrightarrow{D2H} c_0...c_{d_{DST}-1}$

$\Rightarrow \mathcal{T}_1 = d_{DST} - d_{SRC} + 2d_{SRC} = d_{SRC} + d_{DST}$

If $d_{SRC} > d_{DST}$, by *DROCR* algorithm, it takes $d_{SRC} - d_{DST}$ hops for *2HP* or *2DP* operations, and $2d_{DST}$ hops for *D2H* operations, as shown below:

$s_0...s_{d_{SRC}-1} \underline{2HP} s_0...s_{d_{DST}-1} \underline{D2H} c_0...c_{d_{DST}-1}$

$\Rightarrow \mathcal{T}_1 = d_{SRC} - d_{DST} + 2d_{DST} = d_{SRC} + d_{DST}.$

(2) *D2H* routing can be applied to the *match* ($m > 0$).

If $d_{SRC} = d_{DST}$, then $\mathcal{T}_2 = 2(d_{SRC} - m) = d_{SRC} + d_{DST} - 2m.$

If $d_{SRC} < d_{DST}$, by *DROCR* algorithm, it takes $d_{DST} - d_{SRC}$ hops for *2HC* operations, and $2(d_{SRC} - m)$ hops for *D2H* operations, as shown below:

$s_0...s_{d_{SRC}-m-1}c_0...c_{m-1} \underline{2HC} s_0...s_{d_{SRC}-m-1}c_0...c_{m-1}...c_{d_{DST}-d_{SRC}+m-1}$

$\underline{D2H} c_0...c_{d_{DST}-1}$

$\Rightarrow \mathcal{T}_2 = d_{DST} - d_{SRC} + 2(d_{SRC} - m) = d_{SRC} + d_{DST} - 2m$

If $d_{SRC} > d_{'DST}$, by *DROCR* algorithm, it takes $d_{SRC} - d_{DST}$ hops for *2DP* operations, and $2(d_{DST} - m)$ hops for *D2H* operations, as shown below:

$s_0...s_{d_{SRC}-m-1}c_0...c_{m-1} \underline{2DP} s_{d_{SRC}-d_{DST}}...s_{d_{SRC}-m-1}c_0...c_{m-1}$

$\underline{D2H} c_0...c_{d_{DST}-1}$

$\Rightarrow \mathcal{T}_2 = d_{SRC} - d_{DST} + 2(d_{DST} - m) = d_{SRC} + d_{DST} - 2m$

According *DROCR* algorithm, *H2D*, *D2D*, and *H2H* routing comply with the same result.

(3) Distributed routing can not be applied to the *match* ($m > 0$)

By *DROCR* algorithm, it takes $d_{SRC} - m$ hops for *2DP* or *2HP* operations if any, and $d_{DST} - m$ hops for *2DC* or *2HC* operations if any.

$\Rightarrow \mathcal{T}_3 = d_{SRC} + d_{DST} - 2m$

Combining the results from part (1), part (2), and part (3),

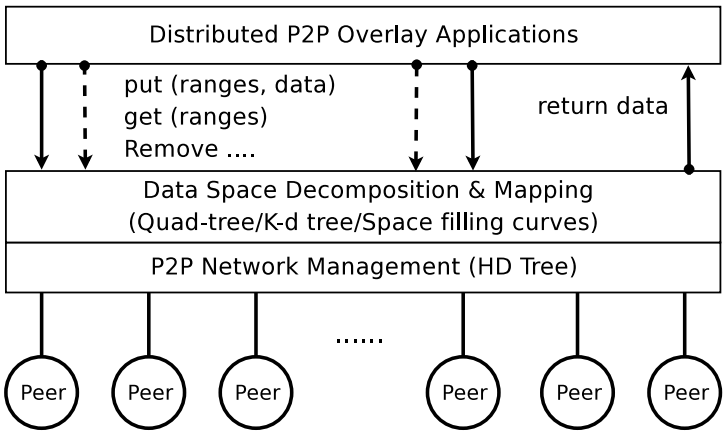


Figure 2.16: System Overview

$$\mathcal{T}(DROCR) = d_{SRC} + d_{DST} - 2m \qquad \square$$

According to this theorem, we have direct observations about the routing performance in *HD Tree*:

Fact 2.2.10. *HD Tree achieves better routing performance than its equivalent tree.*

This is true because any two nodes in *HD Tree* can have a shorter path if a *match* exists. But in its equivalent tree structure, a shorter path is only available if the *match* starts from the left most side of both.

2.2.4 Supporting Multi-dimensional Range Queries

The purpose of the range query is to find the minimum complete set of data nodes[6] by which the data held covers the range data requested. The details about how range query can be carried out in an implementation is determined by the recursive decomposition scheme employed at the higher layer of *HD Tree*, the multi-cast scheme used to forward the query to all relevant data nodes, and the routing mechanism used to forward the range data back to the initiator of the query.

[6]Leaf nodes

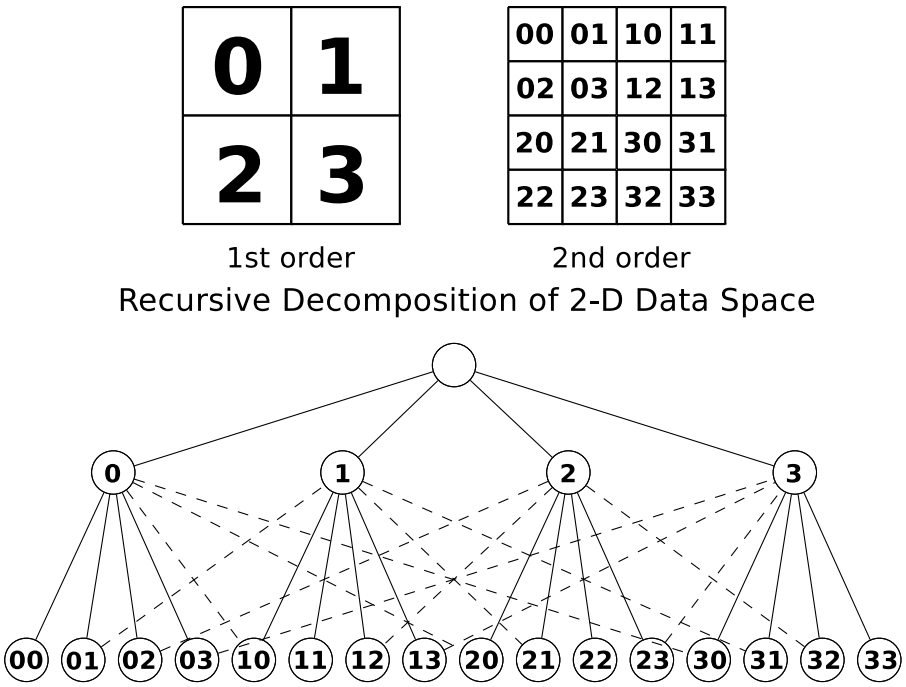
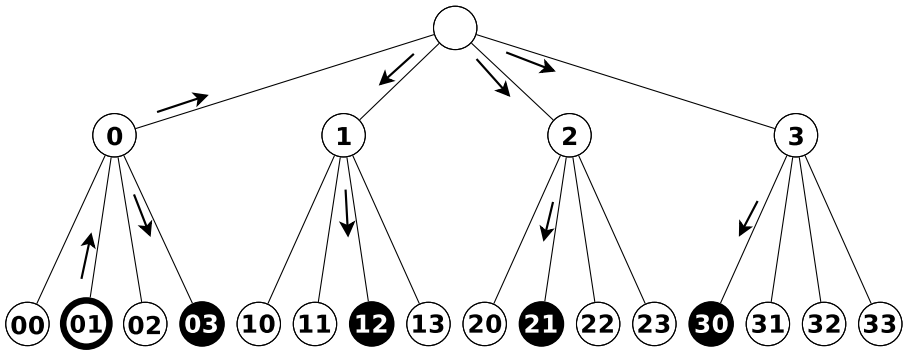


Figure 2.17: Direct Mapping from Data Space to Identifier Space

2.2.4.1 Data Space Partitioning and Mapping

Figure 2.16 depicts the abstract view of the *HD Tree* system. At the data space decomposition and mapping layer, we may choose different recursive decomposition schemes, like Quad-tree, K-d tree, or some other space filling curves like Z-order and Hilbert curve; and at the P2P network management layer, we use *HD Tree* as the basic structure to manage underlying nodes in the system. Mapping from data space to P2P identifier space is direct mapping because *HD Tree* is built over a complete tree structure. Nodes at each depth of a tree correspond to recursive decomposition of the data space at each order, and the locality is properly maintained via parent nodes. In order to accommodate the dimensionality changes, each node at certain depth can manage some number of virtual nodes. Figure 2.17 depicts an example showing the partitioning of a 2-dimensional data space using Z-order space filling curve. The first two orders of this process have been directly mapped into a $4-ary$ *HD Tree* of height 2. The code of each node in this *HD Tree* corresponds to the data range at certain order of the partitioning process.



Initiator: 01; Range data requested: 03, 12, 21, 30

Figure 2.18: Range Query in Tree

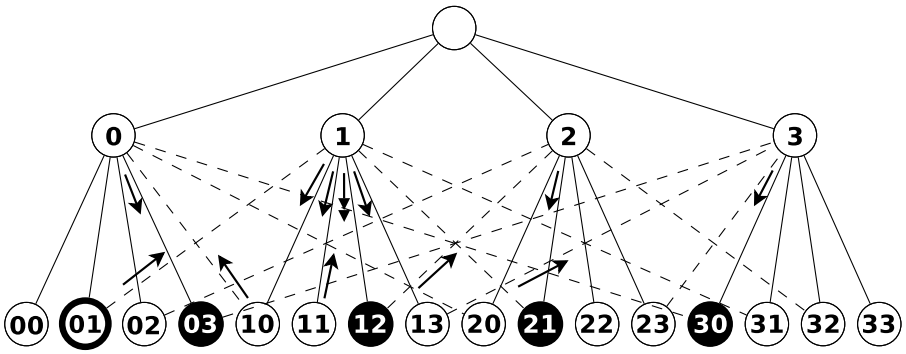
2.2.4.2 Multi-dimensional Range Queries

For the comparison purpose, we use Figure 2.18, 2.19, and 2.20 to show a complete process of the range query in both *HD Tree* and its equivalent tree structure. The range data in query is 03, 12, 21, and 30 (as shown in Figure 2.17). The initiator of this query is node 01. In a tree structure, in order to cover all range data requested, the query has to be forwarded all the way up to the root node, and responses to this query have to follow the same incoming path but in the opposite direction (Figure 2.18).

The range data requested in this example is 03, 12, 21, and 30 (as shown in Figure 2.17). The initiator of this query is node 01.

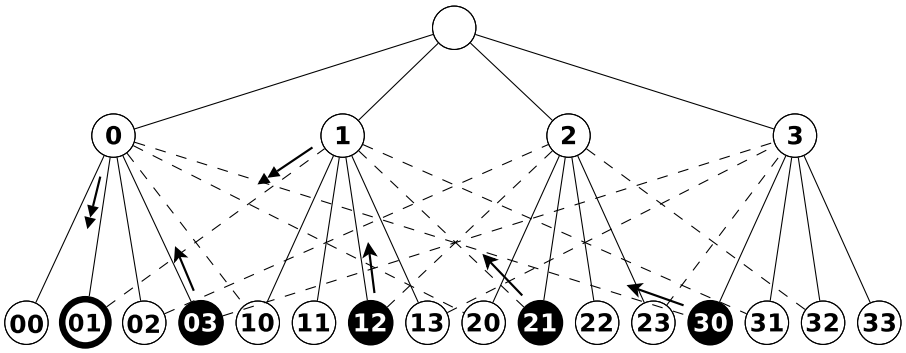
In an *HD Tree*, the query request is forwarded in a distributed manner. It is not necessary to pass through the root node. One of such standard query processes is called *D2H* query. It always chooses some *HSiblings* via *DParent* to forward the query. The process is described as follows:

- the initiator (node 01) initiates the query process and sends the request to *DParent;*
- *DParent* (node 1) forwards the request to *HChildren* whose last digit represents the first order of the current data range in the query (0, 1, 2, and 3 in this case);
- *HChildren* (nodes 10, 11, 12, and 13) pass requests to each of their *DParent*;



Initiator: 01; Range data requested: 03, 12, 21, 30

Figure 2.19: D2H Query in HD Tree



Senders: 03, 12, 21, 30; Receiver: 01;

Figure 2.20: Response to D2H Query

- *DParents* (node 0, 1, 2, and 3) continue forwarding requests to *HChildren* whose last two digits represent the second order of the current data range in the query (03, 12, 21, and 30 in this case); and
- the range data is found at each of the *HChildren* (nodes 03, 12, 21, and 30).

The response to this query is a point-2-point routing process. In Figure 2.20, the requested range data is returned to the initiator node from each data node. This response process demonstrated in this figure assumes that the shortest path can always be found.

The comparison of these two simple examples indicate that the multi-dimensional range query in *HD Tree* may have comparable message complexity as, but shorter time

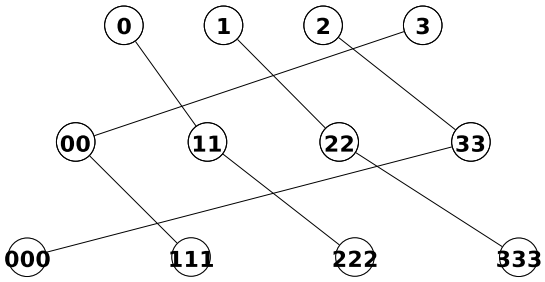


Figure 2.21: Connecting SPeers

complexity ($O(lg(n))$) than what it has in the equivalent tree structure. In addition to this, the traffic load in *HD Tree* are distributed to large number of nodes at higher depth instead of frustrating a limited number of nodes at the lower depth.

2.2.5 Error Resilient Routing

HD Tree doubles the number of neighbors in its equivalent tree structure at the cost of doubling the total number of links of a tree. As a consequence, routing in *HD Tree* could be designed to be highly error-resilient. In the case of a neighboring node's failure, a very simple solution is to redirect the routing request to a functioning neighbor, and to start the routing process over again, which requires another $2h$ hops time at worst. However, by comparing the *match* between functioning neighbors and the destination node and by rearranging the sequence of hierarchical operations and distributed operations, better performance might be achieved.

2.2.5.1 Connecting SPeers

In a $k - ary$ *HD Tree*, the root has k neighbors, all internal nodes except *SPeers* have $2k + 2$ neighbors, and *SPeers* have $2k$ neighbors. All leaf nodes have 2 parent nodes except *SPeers.* which are isolated from the system in the case of a single parent failure. However, this problem can be resolved by introducing extra links at each depth to connect the *SPeer* parent to a *SPeer* child of another *SPeer* at the same depth. There are many

ways to conduct these extra connections. One tactic we employed for error resilient routing is to connect the *i-th SPeer* to the *SPeer* child of *(i+1)-th SPeer* at the same depth (as shown in Figure 2.21). This improvement increases total links in a $k-ary$ *HD Tree* from $2(n-1)-kh$ to $2(n-1)-k$.

HD Tree is built over a complete tree structure. By introducing extra connections from the parent node to some other children nodes at each depth, the advantages we obtained are far beyond the basic properties of a simple tree structure. These advantages make fault tolerance and load balancing problems to be handled and resolved naturally within the *HD Tree* structure itself and at the minimum cost.

2.2.5.2 Error Resilient Routing Mechanism

In the error-prone routing environment, whenever a normal routing operation fails, *DROCR* algorithm switches the normal routing mode into the by-passing routing mode. In the by-passing routing mode, all possible routing operations are tried in order to get around neighboring nodes that have failed.

If there is no *match* between the current node and the destination, "*bp_none_match()*" is called:

1. `if` $d_{DST} \leq d_{SRC}$
   (a) `if` *2HP* `succeed, return success;`
   (b) `if` *2DP* `succeed, return success;`
2. `set` $i = 0$`;`
3. `if` *2HC[i]* `succeed, return success;`
4. `set` $j = d_{DST} - i - 1$`;`
5. `if` *2DC[j]* `succeed, return success;`
6. `if` $i < d_{DST}$
   (a) *i++;*
   (b) `go back to 2;`

7. return *bp_any()*;

In this procedure, if *2HP* and *2DP* can not proceed, "*bp_none_match()*" tries each digit sequentially from the left most side and the right most side of a destination code in order to generate a *match* using either *2HC* or *2DC* operations.

If there exists at least one *match* between the current node and the destination, procedure "*bp_match()*" is called. Let i and j ($i < j$) be the location index of the current *match* in the destination code:

1. if *2HP* steps > 0[7]
   (a) if *2HP* succeed, return success;
2. if *2DP* steps > 0
   (a) if *2DP* succeed, return success;
3. if *2HP* steps == 0 && *2HC* steps > 0[8]
   (a) if *2HC[j+1]* succeed, return success;
4. if *2DP* steps == 0 && *2DC* steps > 0
   (a) if *2DC[i-1]* succeed, return success;
5. if next *match* exists
   (a) set *i, j* to next *match*;
   (b) go back to 1;
6. return *bp_none_match()*;

In the procedure above, two parent nodes are tested in order to "by-pass the failure towards the current *match*. If *2HP* and *2DP* can not proceed, *2HC* and *2DC* are tried in order to extend the current *match* towards the destination code. If by-passing

[7] *2HP operation will not truncate the current match (as shown in Figure2.13).*
[8] *2HC operation will not break the current match (as shown in Figure2.15).*

operations fail in the current *match*, the next *match* is tried. If by-passing operations fail in all *matches* that can be found, "*bp_none_match()*" is tried at last.

"*bp_none_match()*" and "*bp_match()*" are intended to generate a new *match*, and maintain or even extend an existing *match* even if the normal routing operation has been interrupted by failures. Although, when compared to a random by-passing scheme, better performance might be achieved; they can not guarantee the success of by-passing all failures. This is because the next node is determined by a digit in the destination node instead of trying all functioning neighbors. Therefore, the procedure "*bp_any()*" is called when both "*bp_none_match()*" and "*bp_match()*" have failed.

1. `if` *HParent* `not visited`
   (a) `if` *2HP* `succeed, return success;`
2. `if` *DParent* `not visited`
   (a) `if` *2DP* `succeed, return success;`
3. `set` $i = 0$`;`
4. `if` *i-th HChild* `not visited`
   (a) `if` *2HC(i)*[9] `succeed, return success;`
5. `if` *i-th DChild* `not visited`
   (a) `if` *2DC(i)*[10] `succeed, return success;`
6. `if` $i < k$
   (a) *i++*`;`
   (b) `go back to 3;`
7. `return failure;`

HD Tree is a cyclic graph, "*bp_any()*" avoids looping by selecting the next node that has not been visited yet. "*bp_any()*" fails if by-passing operations can not proceed after all previously non-visited neighbors have been tried.

[9] *send to the i-th HChild.*
[10] *send to the i-th DChild.*

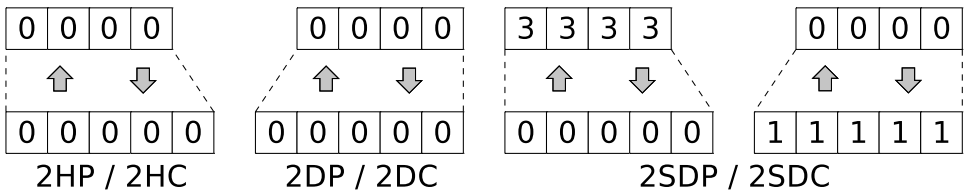


Figure 2.22: Comparison of SPeer's Routing Operations

2.2.5.3 2SDP and 2SDC

System failure occurs in *HD Tree* if there exists a functioning node that has been totally isolated from the system. If there is no *system failure* in an *HD Tree*, the by-passing mechanism is supposed to go around failures and get to the destination eventually. However, "*bp_any()*" guarantees a successful return from the by-passing routing mode only if the current node is not a *SPeer*.

A *SPeer* node has the same *SPeer DParent* as its *SPeer HParent*; and, it also has the same *SPeer DChild* as its *SPeer HChild*. In routing operations we have discussed in previous subsections, *2HP* and *2DP* pass the routing request to the same parent node if the current node is *SPeer*; so do *2HC* and *2DC* if *SPeer* child node is chosen as the next node.

In the by-passing routing mode, when the current node is a *SPeer* leaf node, by-passing operations can not proceed if *HParent* fails. On the other hand, when a *SPeer* leaf node is the destination node, no routing operations can be applied to by-pass the *HParent's* failure.

2SDP (Send to *SPeer DParent*) and *2SDC* (send to *SPeer DChild*) are two additional routing operations used to deal with this situation. *2SDP* forwards the routing request to another *SPeer* parent, called *SDP* (*SPeer DParent*), via an extra link shown in Figure 2.21. While *2SDC* forwards the routing request to another *SPeer* child, called *SDC* (*SPeer DChild*), via an extra link also shown in Figure 2.21. Figure 2.22 is a comparison of these routing operations at a *SPeer* node.

We describe a complete by-passing mechanism "*by_pass()*" as follows:

1. `set SDC_DEST to Null;`
2. `set DEST to destination;`
3. `if DEST is` *`SPeer`* `leaf`
   (a) `if` *`HParent`* `of DEST fails`
      i. `set SDC_DEST to DEST;`
      ii. `set DEST to` *`SDP`* `of DEST;`
4. `if SDC_DEST not null`
   (a) `if current node == DEST`
      i. `set DEST to SDC_DEST;`
      ii. `set SDC_DEST to null;`
      iii. *`2SDC`*`, return success;`
5. `if` *`match`* `not found`
   (a) `if` *`bp_none_match()`* `== success`
   `return success;`
6. `if` *`match`* `found`
   (a) `if` *`bp_match()`* `== success`
   `return success;`
7. `if sender exists`
   (a) `send back to sender, return success;`
8. `if` *`2SDC`* `succeed, return success;`
9. *`2SDP`*`;`
10. `return success;`

In the simulation, we only considered the routing nodes'[11] failures. We have tested this simple by-passing mechanism in a $2-ary$, $4-ary$, and $8-ary$ *HD Tree*. The maximum nodes' failures that can be handled so far is about 10 percent of routing nodes failures.

[11] Internal nodes.

Node 23	0	1	2	3
HChild ...	1	1	1	1
1-D2H via HChild ...	0	0	0	0
2-D2H via HChild ...	0	0	0	0

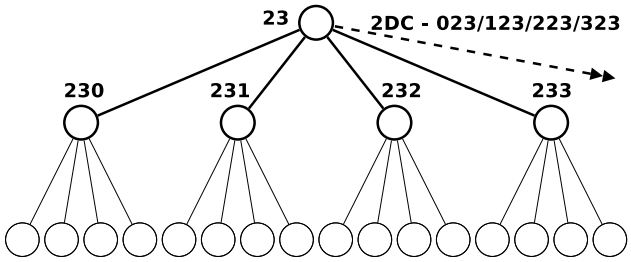


Figure 2.23: Entry Table at Node 23 in $4-ary$ HD Tree

2.2.6 Maintenance Operations

HD Tree needs to maintain a complete tree structure at all times. A new node has to join the system as a leaf node of the tree, and an existing node needs to find a leaf node as the replacement before it disconnects from the system. In *HD Tree*, a internal node uses *Entry Table* to direct the Join or Leave request towards the nearest leaf entry. *Entry Table* is a binary table maintained at each internal node to track the status of leaf entries at each different hops time. It is updated in a progressive manner after each Join or Leave operation.

2.2.6.1 Entry Table

An *Entry Table* at depth d has $(d+1) \times k$ binary entries (as shown in Figure 2.23). Entry (i, j) represents the state of all nodes at depth $d+1$ that are $2i+1$ hops away via j-th *HChild.* In a full *HD Tree* of height h, *Entry Table* for the Join operation is maintained by each node at depth $h-1$, and the Join request is directed to the node at depth h. For instance, in the *Entry Table* at node 23 (Figure 2.23), the first row indicates the status of all *HChildren.* In this case, no more empty entries can be used for the next Join request. The second row represents the status of all nodes in one *D2H* operation time, starting from each of k *HChildren* (3 hops in total). This figure shows that empty leaf entries exist in one *D2H* operation time via any *HChild* of the current node.

Moreover, if all entries in the i-th row have become 1, notifications have to be sent to all relevant nodes - *HParent* of all *DChildren.* The relevant node updates its own

Node 02	0	1	2	3
HChild ...	0	0	1	0
1-D2H via HChild ...	0	0	0	1
2-D2H via HChild ...	0	0	0	0

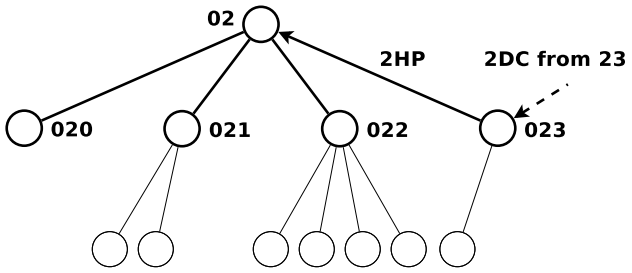


Figure 2.24: Entry Table at Node 02 in $4-ary$ HD Tree

Entry Table by finding the entry in the $(i+1)$-th row, and at the column of the *HChild* via which the notification is received, and setting it to 1. In the example depicted by Figure 2.23, these relevant nodes are 02, 12, 22, and 32, because *HChildren* of node 23 are nodes who are one *D2H* operation time away via the third *HChild* of each relevant node, and all empty leaf entries at *HChildren* have been used. Therefore, the second row of Entry Table at all relevant nodes (node 02, 12, 22, and 32) need to be updated accordingly (as shown in Figure 2.24). When all entries in the second row of *Entry Table* at node 23 become 1, the same notification process continues at the third row of *Entry Table* at all relevant nodes. By this way, all *Entry Tables* at depth $h-1$ are maintained simultaneously in a progressive manner. Whenever all entries in the bottom row of an *Entry Table* become 1, all entries in all *Entry Tables* have become 1.

2.2.6.2 Join and Leave Operations

In a full *HD Tree* of height h, *Entry Table* for the Join operation is maintained by each node at depth $h-1$; it is initialized to all $0s$. In a Join operation, entry 0 represents the existence of some empty leaf entries for the new node to join, while entry 1 stands for the opposite meaning.

A Join request is first directed towards nodes at depth $h-1$. Whenever a node at depth $h-1$ receives a Join request, it checks the first row of *Entry Table*. If there exists any 0 entry, the request is passed to the corresponding *HChild*. Otherwise, the next row is checked, and so on. When a node at depth h is fully filled with leaves, it

notifies *HParent* to update its entry in *Entry Table*. When a row in *Entry Table* has become all 1s, it notifies all relevant nodes to update their entries in the next row of *Entry Table* accordingly; and, notifications are sent to all *HParents* of *DChildren*. As the Join operation moves on, all entries in *Entry Table* may be filled with all 1s. At this point, the height of the full *HD Tree* is extended to $h+1$, and new *Entry Tables* for Join operation are initialized to all 0s and maintained at each node at depth h. The next Join operation will be directed to nodes at depth h.

On the other side, *Entry Tables* at depth $h-1$ were left all $1s$ because of previous Join operations at this depth; and, they become *Entry Tables* used for the next Leave operation. In a Leave operation, entry 1 indicates that there exist some leaf nodes which can be used as replacements, while entry 0 means that no leaf node can be used as the replacement any more.

In a full *HD Tree* of height $h+1$, a Leave request is directed towards nodes at depth $h-1$. Whenever a node at depth $h-1$ receives a Leave request, it checks the first row of *Entry Table*. If there exists any 1 entry, the request is passed to the corresponding *HChild*. Otherwise, the next row is checked, and so on. Right after the first Leave operation is completed, the height of the full *HD Tree* is reduced to h. When all children entries of a node at depth h are empty, it notifies *HParent* to update its entry in *Entry Table*. When a row in *Entry Table* has become all 0s, it notifies all relevant nodes to update their entries in the next row of *Entry Table* accordingly. It is the same notification process as the Join operation, but with opposite operations. As the Leave operation continues, all entries in *Entry Table* may become all 0s. At this time, *Entry Tables* at depth $h-2$ will be used for the next Leave operation, and *Entry Tables* at depth $h-1$ were left all $0s$ because of previous Leave operations at this depth; and, they become *Entry Tables* used for the next Join operation. In the implementation, *Entry Tables* used for Join and Leave operations have to be updated cooperatively.

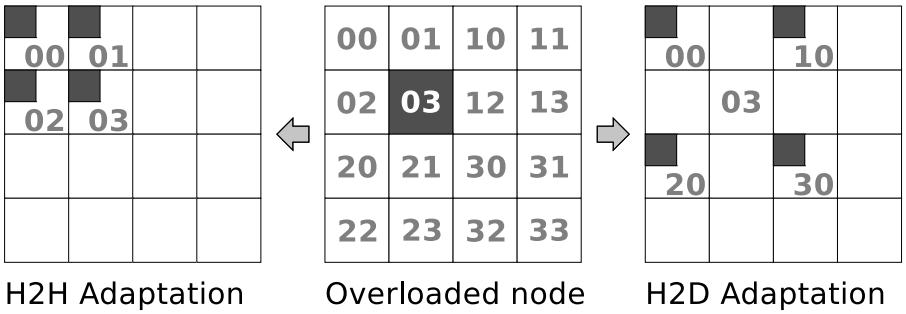


Figure 2.25: H2H and H2D Local Adaptation

2.2.6.3 Performance Analysis

In a full *HD Tree* of height h, directing the Join or Leave request from any node to a node at depth $h-1$ is $h-1$ hops time operation at worst; and, finding the right entry for Join or Leave operation is a $2h$ hops time operation at worst. Therefore, the Join or Leave operation in *HD Tree* based on maintaining a binary entry table costs $3h-1$ hops time at worst, and it is in $O(lg(n))$.

Fact 2.2.11. *In a full HD Tree of height h, the worst cast time complexity of Join or Leave operation is* $\mathcal{T}_{worst}(Join/Leave) < 3h$.

2.2.7 Dynamic Load Balancing

Figure 2.17 demonstrates direct mapping from two dimensional data space to *HD Tree* identifier space using Z-order space filling curve. Each order of recursive decomposition corresponds to nodes at the depth of that order. The data distribution load at each data node is uniquely identified directly by the node ID. This figure also indicates that data space is evenly partitioned and uniformly distributed into identifier space. Recursive decomposition partitions data space and maps it into identifier space. It stops when the recursive order reaches the height of the full *HD Tree*. Each data node manages the data distribution load at this order, and it is called *basic load*.

However, in a real P2P environment, some nodes might have a lower capacity for participating in system wide operations. On the other hand, some data might be accessed more frequently and become hot spot areas in data space. All these factors may lead to a bottle neck or some heavily loaded nodes in *HD Tree*.

2.2.7.1 H2H and H2D Local Adaptation

H2H and *H2D local adaptation* are local load balancing schemes affecting only sibling nodes and *HParent*. The overloaded node re-partitions current load, and redistributes it to *HSiblings* or *DSiblings* via *HParent* (as shown in Figure 2.25). This load adaptation scheme is a 2-hop time operation ($H2H$ or $H2D$); and, it is a local load balancing scheme because it disseminates load to nodes close to the hot spot area.

H2H and *H2D local adaptation* do not change the direct mapping feature in *HD Tree*. However, they alternate the mapping from data space to identifier space locally and temporally. The *basic load* at the heavily loaded node is partitioned into k smaller parts of load. Each part loses the lowest digit on the right, but obtains one digit from 0 to $k-1$ on the right most (*H2H*) or left most (*H2D*) side. This transition process is also depicted in Figure 2.25. This local load adaptation scheme might fail if other siblings are also heavily loaded. However, it does not generate any extra routing cost to *D2H* range queries, because all *data nodes* have to be accessed via *HParent* in *D2H* queries.

2.2.7.2 D2H Repartitioning

The local adaptation scheme provides a temporary but simple solution for the load balancing. However, it is not able to manage the situation when a big hot spot area spreads across many nodes in *HD Tree*. We propose another system wide solution, *D2H repartitioning*, which is based on the recursive decomposition scheme employed at higher layer. *D2H repartitioning* is a global load balancing scheme. It is the continuation of the recursive decomposition process, and it re-partitions and redistributes the *basic load* at

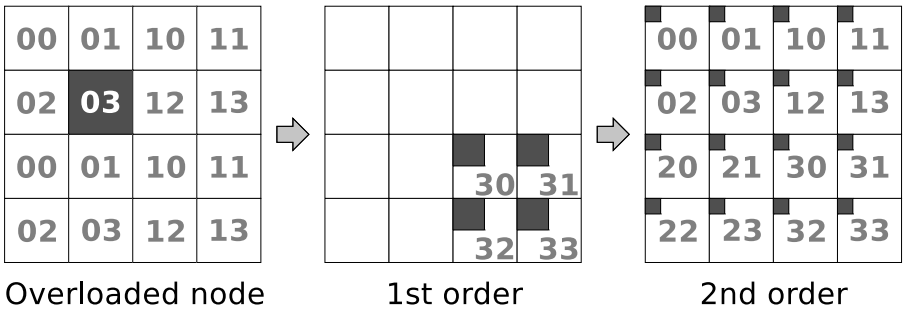


Figure 2.26: D2H Load Repartitioning

each data node. *D2H repartitioning* is a dynamic load balancing scheme, it balances the load at each data node system wide in a dynamic manner. In a full *HD Tree* of height h, the order of *D2H repartitioning* can be adjusted between 0 and h depending on the node´s capacity and hot spot areas in data space.

In order 0, the *basic load* is managed at each data node. The overloaded node initiates the next order *D2H repartitioning*. This load balancing process re-partitions the load in the current order and redistributes it to *HSiblings* via *DParent*. All other nodes start the next order *D2H repartitioning* process once they receive the redistributed load in the next order from *HParent*. *D2H repartitioning* will stabilize at the next order within $2h$ hops time, after which all data nodes have had their load in the current order re-partitioned and redistributed to *HSiblings* which are 2 hops away via *DParent*. This global load balancing process stops at this order if no overloaded node can be found anymore. Otherwise, the same process will be initiated again, but it will continue at another higher order.

Through this way, the load at each data node is re-partitioned and redistributed. This, in turn, recombines the load in the next order system wide at each data node. This process can be repeated at an even finer grade as the order of *D2H repartitioning* goes higher. Therefore, the hot spot areas are most likely to be dissolved among all data nodes in the system. Figure 2.26 depicts this *D2H repartitioning* process at the overloaded node 03.

D2H repartitioning alternates the mapping from data space to identifier space system wide. In each order of *D2H repartitioning* process, the load identified by the node ID in the previous order is partitioned into k smaller parts of load. Each part loses the highest digit on the left (*2DP*), but obtains one lowest digit from 0 to $k-1$ on the right (*2HC*). This transition process is also shown in Figure 2.26.

D2H repartitioning does not change the direct mapping feature in *HD Tree*, because the new order of repartitioning becomes global knowledge after $2h$ hops stabilization time. However, the involvement of data nodes caused by a range query expands exponentially: the same range query involving only 1 data node at order 0 will become a range query involving k^j data nodes at the order j; and, it will become the range query involving the entire system at the order h. This is the main trade-off between load balancing and the performance of range queries.

2.3 Experimental Results

We built an *HD Tree* P2P overlay structure in ns2. The reason why we choose ns2 as the implementation platform was to facilitate our future study of different scenarios that might be applied to the lower layer network infrastructure, the higher layer data space partitioning, and the P2P overlay application layer.

2.3.1 Simulation Environment

Currently, the *HD Tree* overlay is built over the traditional wired network infrastructure. At the P2P overlay application layer, we simplify the multi-dimensional data space in real application scenarios to an abstract data space with $[0, 1]$ range at each dimension. At the data space decomposition and mapping layer, we employed Z-order space filling curve. An *HChild* at depth d decomposed 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 Z-order space filling curve. Therefore, the *HCode* of a node indicates which part of the abstract data it is currently managing (as shown in Figure 2.17). In order to support multi-dimensional range queries in the error-prone routing environment, the error resilient routing has been built inside the *DROCR* algorithm.

We have tested *HD Tree* structure in both ideal and error-prone distributed routing environments. In an ideal distributed routing environment, the maximum network size we tested for basic routing operations includes a $2-ary$ *HD Tree* of height 12 (8191 nodes), a $4-ary$ *HD Tree* of height 6 (5461 nodes), and a $8-ary$ *HD Tree* of height 4 (4681 nodes). However, in an error-prone distributed routing environments, the maximum network size we are able to use for the range query is reduced to a $2-ary$ *HD Tree* of height 9 (1023 nodes), a $4-ary$ *HD Tree* of height 5 (1365 nodes), and a $8-ary$ *HD Tree* of height 3 (585 nodes). The maximum dimensionality we tested in all simulation runs is up to 6. Each node in *HD Tree* manages a certain number of virtual nodes in order to accommodate the dimensionality changes.

We have to say that *HD Tree* itself is highly scalable. These limitations are set according to the capability of our hardware resources, because ns2 is a single process simulator.

2.3.2 Average Routing Performance

Simulations for validating the average routing performance are conducted in an ideal routing environment. It includes the average routing performance of basic distributed routing and the combined routing. The average routing performance is evaluated among all combinations of the source and destination pair at the same depth from 1 to the tree height; and, it is measured by the fraction of the average number of hops used from the source to the destination node over two times their depth.

$$Avg.\,Routing = \frac{average\,\#\,of\,hops\,used}{2 \times d}$$

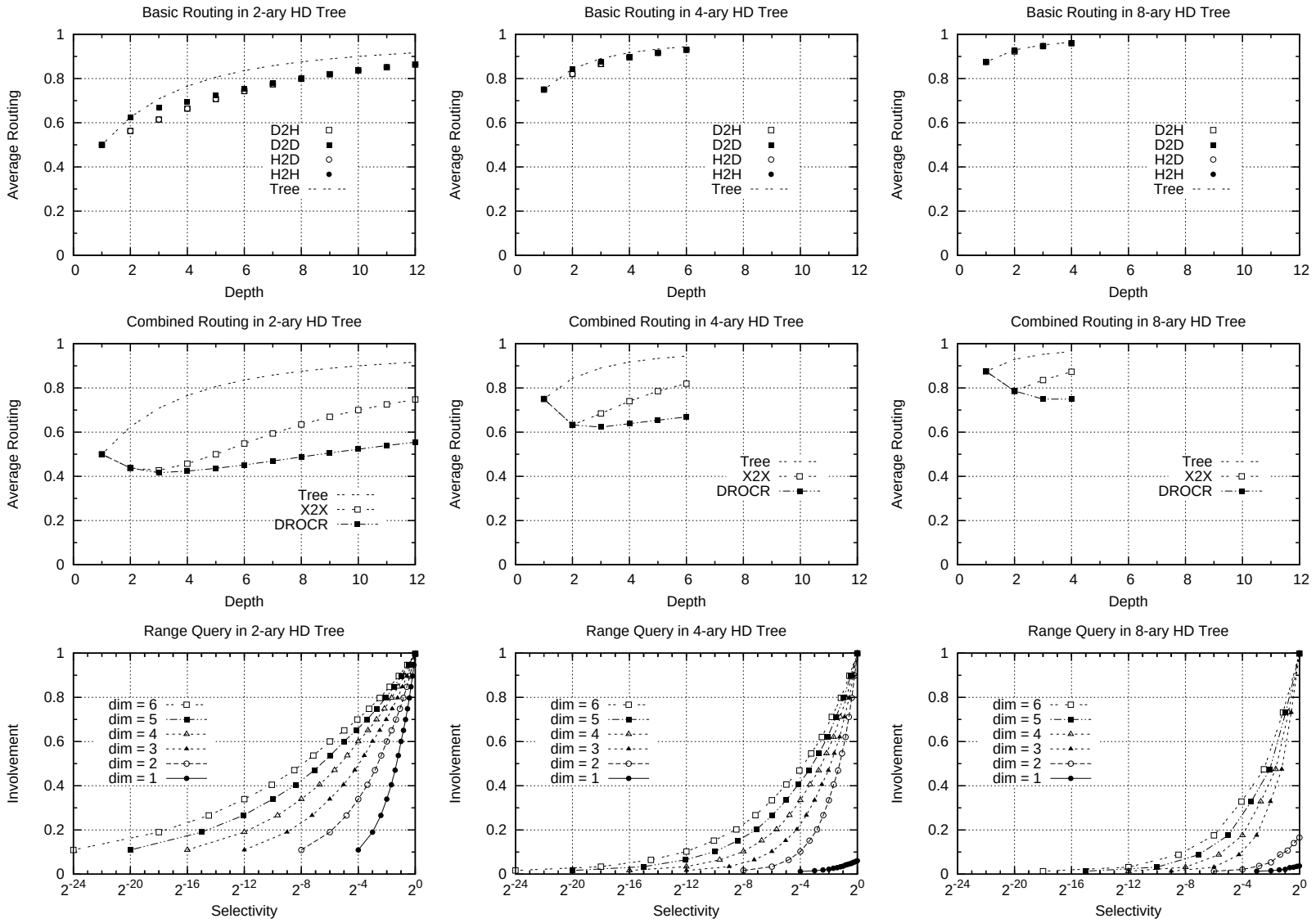


Figure 2.27: Routing and Range Query in Ideal Routing Environment

In this part of the experiment, we first simulated each of the four distributed routing operations (*D2H*, *D2D*, *H2D*, and *H2H*) separately; and, we also simulated the combination of these four distributed routing operations (*X2X* routing) and the combination of both hierarchical and distributed routing (*DROCR* routing). The results are shown by using different points or line-points in the first and second row of Figure 2.27; the comparisons are also made by drawing performance lines for each of the equivalent tree routing.

In subsection 2.2.3.1, we conclude that the average performance of basic distributed routing has an upper bound. This upper bound is the average routing performance of a *SPeer* at the same depth, and it is the same as the performance of the routing in the equivalent tree structure. They are all in $O(log_k n)$, but outperform $2d$ hops time: the worst case time complexity of distributed routing in *HD Tree*. Our simulation results in the first row of Figure 2.27, comply with this analysis. It also indicates that, each of the four basic distributed routing alone achieves slightly better performance than what is possible in an equivalent tree structure. In addition, *D2H* and *H2D* have the same routing performance as well as *D2D* and *H2H*, because they are the reversed distributed operations of each other.

We have claimed that the average routing performance in *HD Tree* is comparable to all existing structured P2P overlay systems. Although it is true that they all appear to be bound by $O(lgN)$, the significance of this part of the experiment does not lie in this. In *HD Tree*, distributed routing has more routing options by which to reach the same destination than its equivalent tree routing; and, they all appear to have comparable routing performance ($O(log_k n)$). Therefore, by combining four distributed routing algorithms together (*X2X* routing), we are able to achieve more performance gain over the equivalent tree routing. This is because the shortest path is more likely to be found by comparing the *match* found in *D2H*, *H2D*, *D2D*, and *H2H* routing all together. Moreover, the *DROCR* algorithm, which is the complete combination of both distributed routing and hierarchical routing, achieves the most performance gain. This is because as long as

there is a *match* between the source and destination pair, the shortest path can always be found. The second row of Figure 2.27 shows that the performance gain of *DROCR* over tree almost doubles that of *X2X* over tree at a higher depth.

Simulation results shown in the first 2 rows of Figure 2.27 also indicate that the lower ary *HD Tree* achieves better routing performance compared to the higher ary *HD Tree,* but less performance gain over the equivalent tree routing. The reason is that the lower ary *HD Tree* has s relatively larger number of routing nodes but less routing options when compared to the higher ary *HD Tree*. However, they all rise up slowly and get close to the performance of the equivalent tree routing as the depth increases step by step. Due to the increase of depth, routing between source and destination pairs without any *match* becomes the dominant element in computing the average routing performance. Therefore, there might exist a certain performance trade-off between the data distribution load and the routing performance. As we have known, the data distribution load at each leaf node decreases exponentially with the increase of height. Meanwhile, the average routing performance goes up much more slowly.

2.3.3 Performance of Range Queries

Range queries in *HD Tree* are conducted in both ideal and error-prone routing environments. The performance of range queries is evaluated by the involvement against the selectivity (as shown in the third row of Figure 2.27 and Figure 2.28), the average routing performance against the involvement (as shown in the first column of Figure 2.29), and the by-passing rate against the involvement (as shown in the second column of Figure 2.29). The involvement is measured by a fraction of the total number of nodes involved in the entire query process over the total number of functioning nodes in the system:

$$Involvement = \frac{total\,nodes_{involved}}{total\,nodes_{functioning}}$$

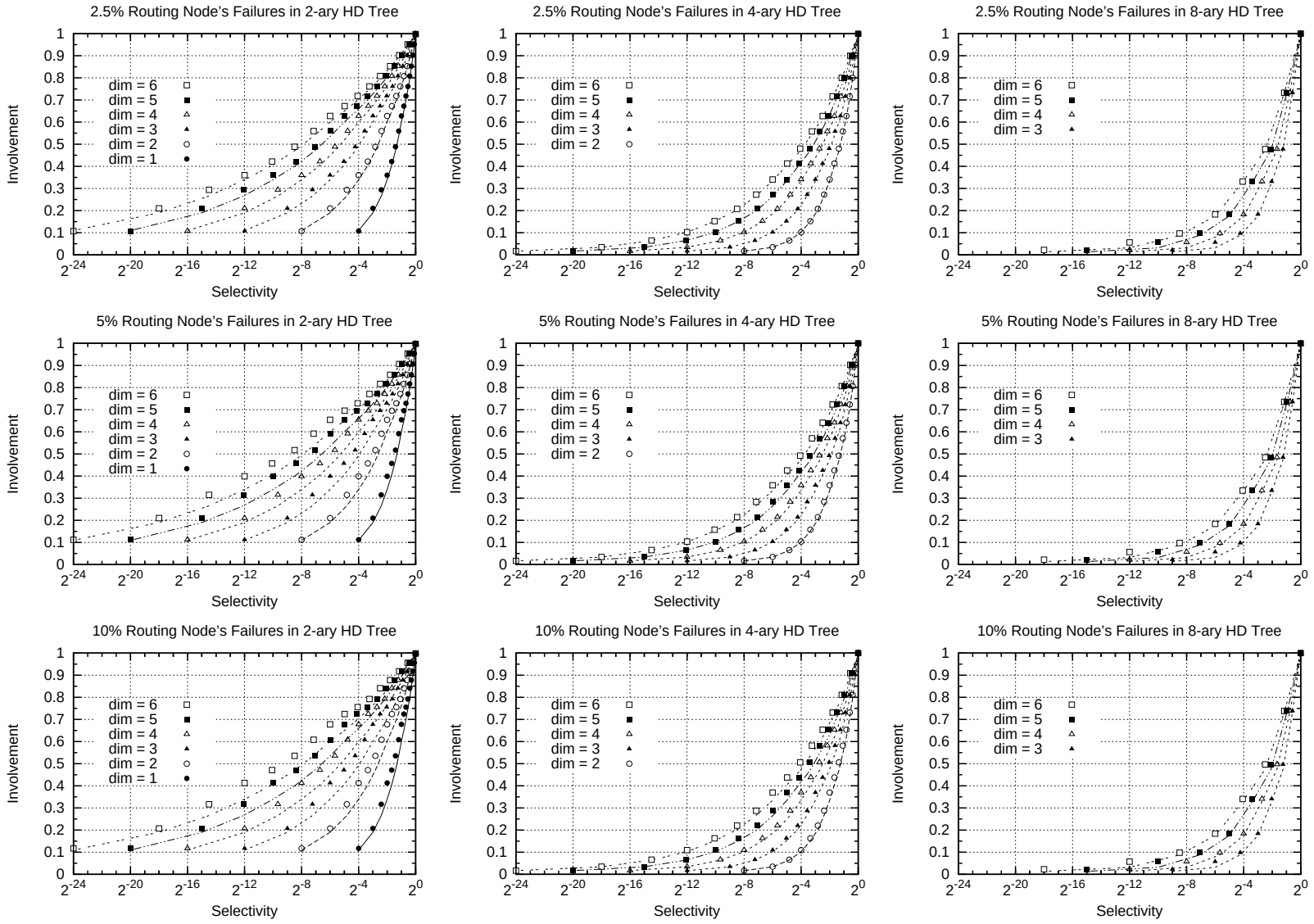


Figure 2.28: Range Query in Error-Prone Routing Environment (I)

The selectivity is measured by another fraction of the size of the range data in the query over the entire data space:

$$Selectivity = \frac{range\,query\,size}{entire\,data\,space}$$

And, the by-passing rate indicates the total number of redirected routing caused by failures over the total number of routing conducted for the same range query:

$$By-passing\,rate = \frac{total\,by-passing\,routing}{total\,routing}$$

In this part of the simulations, we vary the range data in the query from a very small range size to the entire data space. There are a total of 16 steps in the $2-ary$ and $4-ary$ *HD Tree*, and 8 steps in the $8-ary$ *HD Tree*; and, the range data size rises from $\frac{1}{16}$ to 1 and from $\frac{1}{8}$ to 1 at each dimension correspondingly.

In each step of the simulation, we collected the performance of the same range query initiated by different functioning nodes. Each node has to run as the initiator of the same query once. The initiator starts the query and stops until all range data requested has been received. The average performance of the same range query is evaluated among all functioning nodes in the system.

In order to set up an error-prone routing environment in the simulation, we randomly chose some routing nodes at a depth from depth 0 to $h-1$ and set their states to the failure mode. As long as these failures do not cause any *system failure*, any functioning node is supposed to survive the error-prone environment, and to fulfill the routing task that it carries. The maximum failures we have tested so far is up to 10 percent of routing nodes failures.

The performance is evaluated in four different cases as follows:

1. The variation of the involvement in a fixed multi-dimensional data space by varying

the selectivity of range data from a very small range size to the entire data space, which is shown by each performance lines and points in the third row of Figure 2.27 and Figure 2.28.

2. The comparison of the involvement in different multi-dimensional data space by varying the dimensionality from 1 to 6, which is shown by different performance lines and points in the third row of Figure 2.27 and Figure 2.28.

3. The comparison of the involvement among a $2-ary$ *HD Tree* of height 9 (the first column at the third row of Figure 2.27 and the first column in Figure 2.28), a $4-ary$ *HD Tree* of height 5 (the second column at the third row of Figure 2.27 and the second column in Figure 2.28), and a $8-ary$ *HD Tree* of height 3 (the third column at the third row of Figure 2.27 and the third column in Figure 2.28).

4. The comparison of the involvement in the ideal routing environment (performance lines in Figure 2.28) and the error-prone routing environment (performance points in Figure 2.28).

5. The variation of the average routing and by-passing rate of range queries at a fixed failure rate by varying the involvement of nodes at different selectivity, which is shown by each performance line and point in Figure 2.29.

6. The comparison of the average routing and by-passing rate of range queries at different failure rates by varying the total number of routing nodes´ failures, which is shown by different performance lines and points in Figure 2.29.

In both the ideal and error-prone routing, the experimental results show that the involvement of range queries varies proportionally according to the selectivity within the same data space, but deviates considerably with the change of dimensionality (as shown in the third row of Figure 2.27 and Figure 2.28). However, the total number of nodes involved in the range query has been effectively minimized. Because *HD Tree* has

a natural consistency with the recursive decomposition scheme used for the data space partitioning, it does not need to maintain an extra data structure to accommodate the change of data localities with exponentially expanding and extending rates.

In the error-prone routing environment, extended simulations for the range query in *HD Tree* with three levels of routing nodes' failures are conducted. The experimental results (Figure 2.28) show that even 10 percent of routing nodes' failures does not cause any significant variations in the involvement. In fact, it is not even so noticeable in $4-ary$ and $8-ary$ *HD Tree* cases (as shown in the second and third column of Figure 2.28), because $4-ary$ *HD Tree* doubles routing options in $2-ary$ *HD Tree*, and $8-ary$ *HD Tree* doubles routing options in $4-ary$.

However, the lower selectivity cases in $8-ary$ *HD Tree* (as shown in the third column of Figure 2.28) indicate that a finer design of the error-resilient *DROCR* algorithm has to be considered if a higher percentage of routing nodes' failures needs to be tested. The current routing algorithm is able to make use of at most one *match* in each of the four basic distributed operations in the by-passing routing mode. The performance may be further optimized if all *matches* between the source and destination pair can be found and properly used in the by-passing routing mode. In addition, routing operations via extra links connecting *SPeers* (as shown in Figure 2.21 and 2.22) should be further explored and well merged into the *DROCR* algorithm.

The average routing and by-passing rate of range queries depicted in Figure 2.29 confirm our analysis in the previous and current subsection, which can be summarized as follows:

- *The lower ary HD Tree has better average routing performance.*
- *The higher ary HD Tree has better fault tolerant capacity.*
- *The error-resilient routing in higher ary HD Tree will be further improved if a finer design of by-passing routing can be explored.*

A simple fact that is necessary to mention is: a $2-ary$ *HD Tree* of height 9 has 512 data

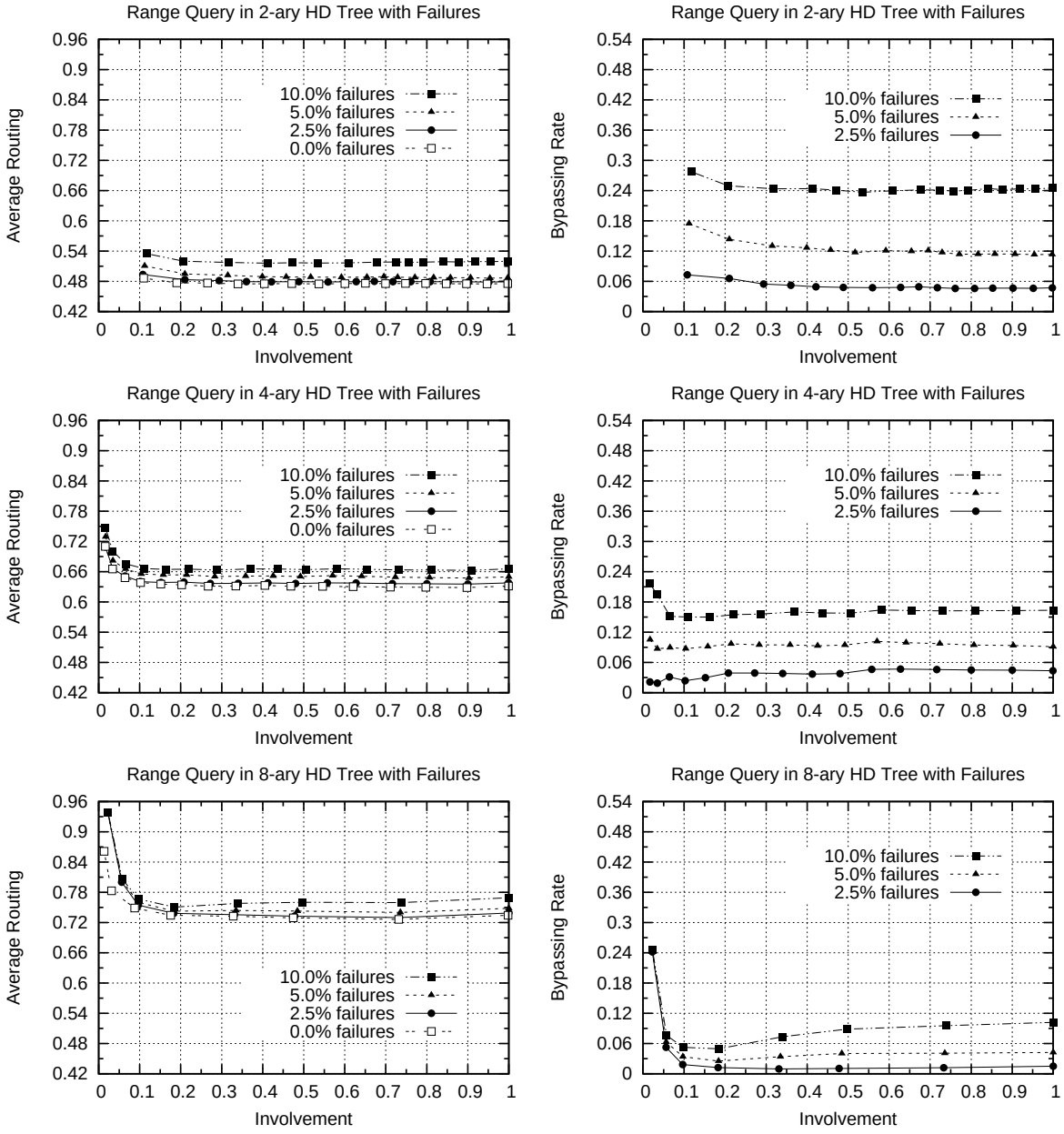


Figure 2.29: Range Query in Error-Prone Routing Environment (II)

nodes, they are indexed by 511 routing nodes, and the average data distribution load at each data node is $\frac{1}{512}$. As a comparison, a $4-ary\ HD\ Tree$ of height 5 has 1024 data nodes. They are indexed by 341 routing nodes, and the average data distribution load at each data node is $\frac{1}{1024}$; and, a $8-ary\ HD\ Tree$ of height 3 has 512 data nodes. They are indexed by 73 routing nodes, and this $8-ary\ HD\ Tree$ has the same data distribution load as the $2-ary\ HD\ Tree$ of height 9.

2.3.4 Performance of Join and Leave

Simulations for nodes' Join and Leave operations are conducted in an ideal routing environment. Join and Leave operations run in a full *HD Tree* of height h. We set up a full $2-ary\ HD\ Tree$ of height 10, a full $4-ary\ HD\ Tree$ of height 5, and a full $8-ary\ HD\ Tree$ of height 3 for this maintenance operation.

Before each step of Join or Leave operations, we initialize the simulation environment by randomly setting some number of leaf nodes at $h+1$ as existing leaves. Join operation joins *HD Tree* as a new leaf node at $h+1$ until all leaf entries at $h+1$ are fully filled, while Leave operation takes an existing leaf node at $h+1$ as the replacement until all leaf entries at $h+1$ are empty. The performance of Join and Leave operation is evaluated against the number of existing leaf nodes at $h+1$ and the number of empty leaf entries at $h+1$ respectively; and, it is measured by the fraction of the average number of hops used to find the leaf entry over $3h$:

$$Average\ Join/Leave = \frac{average\ \#\ of\ hops\ used}{3h}$$

In each step of Join or Leave operation, we collect and compute the performance data by initiating the same Join or Leave request from each node at a depth less than $h+1$. Figure 2.30 shows Join and Leave performance in both the average and the worst cases. The performance of Join operation goes up with the increase in the number of existing

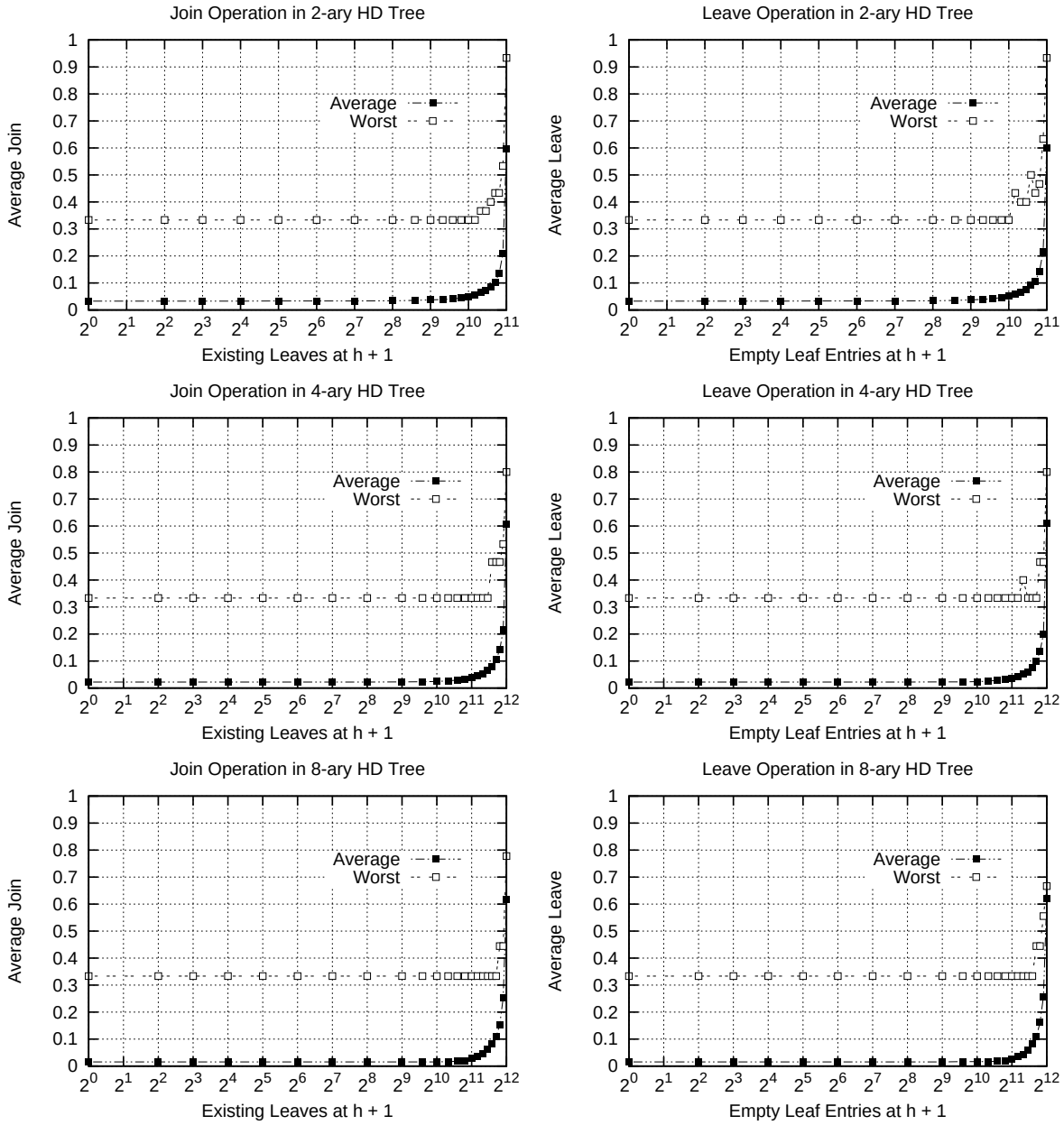


Figure 2.30: Performance of Join and Leave Operations

leaves at depth $h+1$; on the contrary, the performance of Leave operation goes up with the increase in the number of empty leaf entries at depth $h+1$. However, according to Subsection 2.2.6, the performance of both Join and Leave operations should be less than 1, and the experimental results shown in Figure 2.30 complies with this analysis very well.

2.3.5 Performance of D2H Repartitioning

The performance of *D2H repartitioning* is examined by both the involvement bar (the second row in Figure 2.31) and the average routing bar (the second row in Figure 2.32) of the same range query at each order of *D2H repartitioning*. In order to illustrate the connection between a single range query at different orders of *D2H repartitioning* and range queries at different levels of selectivity, we set up the same simulation environment in a 2-ary, 4-ary, and 8-ary *HD Tree* for *D2H repartitioning*. This was likewise done for different range queries in the ideal routing environment. The comparison is made by plotting the involvement (the first row in Figure 2.31) and the average routing performance (the first row in Figure 2.32) of range queries against the selectivity. Moreover, in order to show the impact of load balancing on each data node in *HD Tree*, the partitioning bar of *basic load* at the heavily loaded node is compared against the order of *D2H repartitioning* (the third row in both Figure 2.31 and 2.32).

In this part of the simulation, the range query that is used to collect the performance data at different orders of *D2H repartitioning* involves only 1 data node at order 0; and, we assume the *basic load* at this data node is a hot spot area in data space. The experimental results in the first two rows of Figure 2.31 show that *D2H repartitioning* at each order has equivalent range queries at a certain level of selectivity corresponding to it. This indicates that a range query at a higher order of load balancing involves significantly more nodes to participate in the same operation, and the involvement expands exponentially. However, the experimental results in the first two rows of Figure 2.32 also show that the

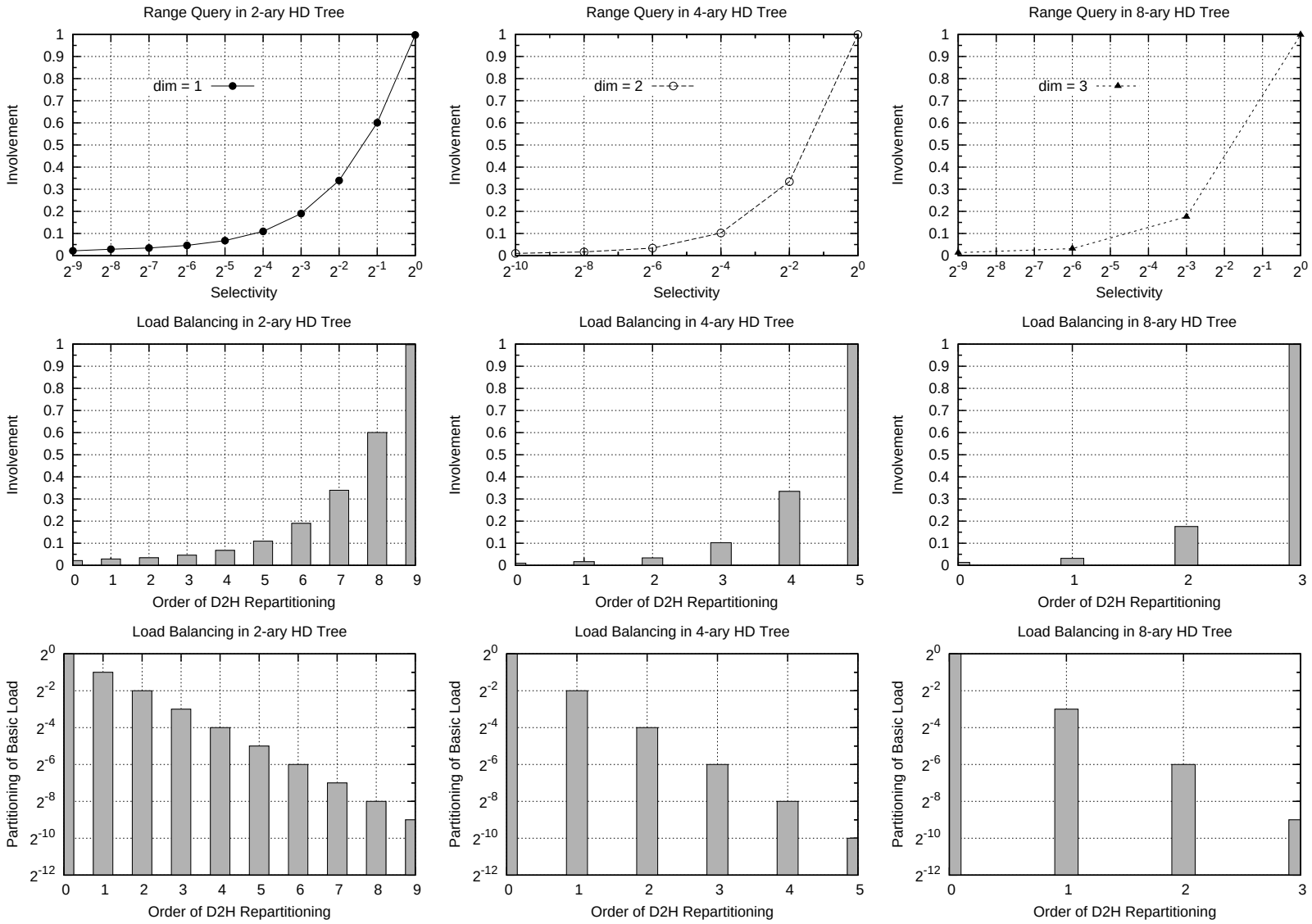


Figure 2.31: Range Query at Different Order of D2H Repartitioning

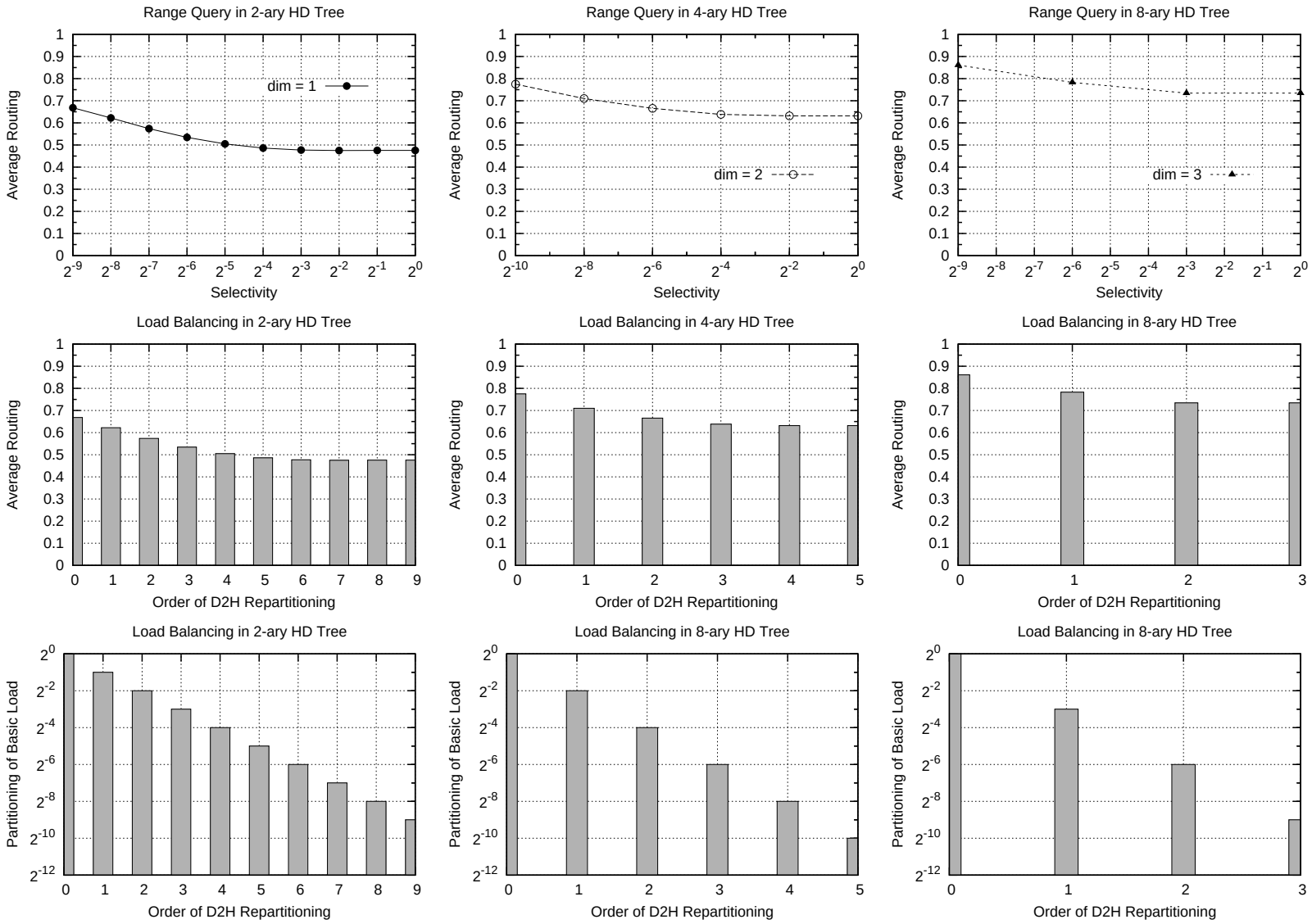


Figure 2.32: Routing at Different Order of D2H Repartitioning

average routing performance of a range query at a different order does not vary so much. Like range queries at different levels of selectivity, it begins to stabilize at a certain value as the involvement expands.

In both figures, the third row shows the partitioning bar of load at the heavily loaded node at each order. The left most bar at order 0 is the initial *basic load*. The exponential deduction of this bar is the consequence of an exponential increase in the involvement of data nodes at each order. This indicates that the hot spot area in data space is most likely to be resolved at a certain order of *D2H repartitioning*. All these experimental results comply with our analysis in Subsection 2.2.7, which addresses the main trade-off between load balancing and the involvement of the range query.

2.4 Summary

Data space partitioning using recursive decomposition makes localities expand and extend exponentially. *HD Tree* provides an optimal solution for supporting multi-dimensional range queries at the P2P overlay layer because of its natural connection to the recursive decomposition scheme employed at a higher layer. This inherent nature makes all important features of a P2P overlay network, like partitioning and mapping, scalability, self-organizing, error-resilience, and dynamic loading balancing, to become a natural part of this novel data structure. *HD Tree* is the first distributed data structure proposed in this area that is able to not only adapt the hierarchical data structure into the distributed environment, but also to give a complete view of system states when processing multi-dimensional range queries at different levels of selectivity and in various error-prone routing environments.

HD Tree is built over a complete tree structure. It doubles the number of neighbors at each node at a depth greater than 1 at the cost of doubling the total number of links of a equivalent tree. *HD Tree* is highly scalable because of its embedded tree structure. However, *HD Tree* has to maintain a complete tree structure at all times.

The basic maintenance operations like Join or Leave are directed by each node according to the *Entry Table* which is updated at all relevant nodes in a progressive manner. In a $k - ary$ *HD Tree*, the routing table size is determined by the system parameter k, and each node has maximum of $2(k + 1)$ neighbors. The main computation in routing operations is a series of digit-shift or digit-replacement operations at either the leftmost or the rightmost side of the code. *HD Tree* achieves better routing performance than its equivalent tree because routing may be conducted either hierarchically, distributively or through a combination of both. Moreover, *HD Tree* is highly error-resilient because multiple options are always available for each routing operation at any intermediate node between the source and destination pair.

The experimental results agree with our performance analysis. It shows that the performance of all basic operations, including routing, and Join and Leave, are bound by $O(lg(n))$ in an ideal routing environment. The experimental results also indicate that lower ary *HD Tree* has a better average routing performance, while higher ary *HD Tree* has a better fault tolerance capacity. The maximum random failures we have tested so far are about 10 percent of routing nodes' failures with very limited performance variations. However, the experimental results also indicate that if a finer design of routing in the by-passing mode can be further explored, the performance will be continuously improved, and *HD Tree* may survive even more harsh error-prone routing environments.

Because of direct mapping from data space to identifier space, the range query is actually a multi-cast routing operation that finds the minimum complete set of neighboring data nodes in which data held covers data requested. On the other hand, load balancing becomes a recursive decomposition process at another level in which the *basic load* at each data node is re-partitioned and again mapped back into identifier space. We claim that the range query in *HD Tree* optimizes the involvement of nodes because of the direct mapping feature and the progressive routing strategy employed. We also claim that load balancing is a global dynamic balancing scheme because it is a system wide decomposition process and it can be adjusted to a finer grade at a higher recursive

order.

HD Tree is not proposed to turn down the exponentially expanding and extending rates of data localities in data space partitioning and mapping. Instead, it is designed to minimize the impact of this effect introduced in the P2P overlay layer. In the simulation for multi-dimensional range queries, the experimental results show that the involvement of nodes varies proportionally with the selectivity of range query. However, it deviates considerably with the change of dimensionality, because *HD Tree* is the first P2P overlay structure that is able to give a complete view of the system state when processing multi-dimensional range queries in data space with various dimensionality and from a very small range size to the entire data space. We believe that our experimental results in this part of the thesis work is reasonable, but they only provide a preliminary system overview of range queries. Therefore, the performance of range queries in *HD Tree* may be continuously improved in the future.

D2H repartitioning scheme makes load balancing become a dynamic and global process in *HD Tree*. The experimental results show that a range query with the selectivity equivalent to the *basic load* at order 0 becomes a range query involving the entire system at order h. However, because only a small part of data space is frequently accessed by the range query generally, this phenomenon is considered more as the effective balancing of heavy load among data nodes rather than frustrating the *HD Tree* system. Moreover, the successful switching from a lower balancing order to a higher balancing order indicates that the *D2H repartitioning* is able to stabilize within $2h$ hops time at each order. Finally, because this balancing scheme is a reversible process, the order of load balancing can be adjusted back and forth according to the change of hotspot areas in data space.

This part of the thesis work provides a preliminary overview of a novel distributed data structure. Although our conclusions are supported by both theoretical analysis and experimental results, the experimental part is obtained from a simulation and by using an abstract data space. Nevertheless, *HD Tree* presents outstanding properties to support multi-dimensional range queries in the distributed environment. Our next step

of work will be to fully explore the fault tolerance capacity in higher ary *HD Tree*, and apply real application scenarios to the new data structure. In our future work, we will continue to adapt this distributed data structure into the wireless network, and explore the new potentials in both wired and wireless environments.

Chapter 3

DDM in Cluster-Based Network Environments

3.1 Basic Concepts

In this section, we introduce some basic concepts that are widely used in HLA/RTI based distributed simulations community[46–48, 66]. In order to understand these concepts more thoroughly, we incorporate their different representations as used by two major DDM systems: the Region-based DDM system and the Grid-based DDM system.

3.1.1 Routing Space

Routing space is a virtual area in which a simulation runs[45, 52, 53]. In the Region-based DDM system, *routing space* is represented using a multi-dimensional coordinate system, while, in the Grid-based DDM system, it is divided into a certain number of grid cells. Therefore, a target area in *routing space* can be denoted either by a set of extents

from the multi-dimensional coordinate system when the Region-based DDM system is employed, or by a set of cell IDs when the Grid-based DDM system is employed. In this part of thesis, we call the first representation of *routing space* the Region-based DDM environment and the second representation the Grid-based DDM environment. Because our interest lies in the Grid-based DDM environment, we consider a two-dimensional, grid-based, square *routing space* in order to simplify our analysis. However, our results can also be extended for more general situations. In this thesis, we use $C = c \times c$ to describe the size of the *routing space*, where $c \in N$ represents the total number of cells in each dimension.

3.1.2 Objects, Federates and Federation

In a distributed environment, a *federate* is considered to be a distributed physical entity or a node in the network. Each *federate* can manage multiple virtual entities, called *objects*. Indeed, each *federate* is the owner of the *object*s it manages. Finally, *federation* is used to represent all of the *federates* involved in a simulation. We thus have three levels: the *object* level, *federate* level, and *federation* level. In this part of thesis work, our main problem is resolved recursively by solving smaller problems at each lower level. When our focus is on the *object* level, we are interested in the behavior of each *object* owned by a *federate;* when our focus moves to the *federate* level, our interest is in the overall behavior of all *objects* owned by a single *federate*; and when our focus is on the *federation* level, we are interested in the overall behavior of all *federates*. Normally, we use f to denote the total number of *federates*, n to denote the total number of *object*s, and n_i to denote the total number of *object*s at a *federate* i.

3.1.3 Behavior of Entities

3.1.3.1 Distribution of Objects

Initially, the *object*s owned by each *federate* are distributed to certain locations inside the *routing space*. In our work, this is the first distribution pattern that needs to be defined explicitly.

3.1.3.2 Movement of Objects

After their initial distribution, the *object*s move in one out of four directions (East, West, South and North) in each *timestep*. An *object* stops at the border of *routing space* until a different direction is chosen. DDM messages are generated because the movement of *objects* leads to the state change of cells in the Grid-based DDM environment: thus, choosing the direction of movement for each *object* in each *timestep* is the second distribution pattern that needs to be defined explicitly.

3.1.3.3 Publishing and Subscribing

Publishing occurs when an *object* wants to provide information about itself to a certain area inside *routing space*, while subscribing occurs when an *object* needs to obtain information that is being published by other *object*s within a certain area inside *routing space*. A key fact to note about these two behaviors is that they are always bounded to a certain area. Any information published by an *object* is published into a specific area (publication region) in *routing space*, while any subscribing action is an effort made by an *object* to see if any information can be obtained from a specific area (subscription region). The movement of *object*s in *routing space* results in a considerable amount of publishing/subscribing requests being sent to the new target area, and in un-publishing/un-subscribing requests to the area to which the *objects* have previously published or subscribed.

3.1.4 Other Important Concepts

3.1.4.1 Timestep

There has been no exact definition of *timestep* thus far. According to the existing DDM works, in a discrete event simulation, we understand it as the distance of movement of an *object* within a constant period of time, after which a new direction of movement should be chosen. Our study shows that *timestep* is of great value in the Grid-based DDM system. We will give *timestep* a formal definition later in this part of thesis.

3.1.4.2 Publisher and Subscriber

At the *object* level, a *publisher* or *subscriber* to an area is an *object* that is currently publishing or subscribing to that area. At the *federate* level, a *publisher* or *subscriber* to an area is a *federate* that has at least one *object* publishing or subscribing to that area. When an *object* has both publishing and behaviors, it can be either the *publisher* of one area or the *subscriber* of another depending on which behavior we are studying. In our analysis, we assume that the *object*s have both the publishing and subscribing behaviors.

3.1.4.3 Publication and Subscription Region

A publication or subscription region is an area of limited size to which an *object* is able to publish or subscribe. As we have mentioned previously, in the Region-based system, a region is denoted by a set of extents from the multi-dimensional coordinate system; and in the grid-based system, it is denoted by a set of cell IDs. Because we are interested in the grid-based system, and because an *object* has the symmetric behavior of publishing and subscribing, we use $s \times s$ to represent the size of a publication region or subscription region, where s is the number of cells in each dimension. However, we must also keep in mind that $s \times s$ is the *object* publication region size ($s = s_p$) when we study the publishing behavior, and it becomes the *object* subscription region size ($s = s_s$) when we

study subscribing behavior. These two sizes are not necessarily identical.

3.1.4.4 Maximum Publication and Subscription Region

An important fact that we have to mention is that, in the Grid-based system, an *object* with $s \times s$ publication or subscription region size normally publishes or subscribes to $(s + 1) \times (s + 1)$ cells. The reason for this is that a published or subscribed region tends to fall outside the boundaries of grid cells, and in the grid-based system a region is counted in an integer number of cells. This effect is the main drawback introduced by the grid-based DDM system. In addition, when an *object* is in the border area, it might not be able to publish or subscribe to the desired area, because the publication or subscription region may fall outside the boundary of *routing space*. This is the main reason for publication and subscription loss when we try to scale a simulation.

3.1.4.5 Matching

We have mentioned that publishing and subscribing are two behaviors that are bounded to certain areas inside *routing space*. Thus, whenever there is an intersection between the publication region and the subscription region of different *object*s, the information published by any *object*s in the intersection area becomes visible to any other *object*s that are subscribing to that area, meaning that information bounded to the intersection area needs to be transmitted from the *publisher*s to the *subscriber*s. In this case, the state of the intersection area is called *matching*.

In our study, because *federate*s are the physical entities that are responsible for exchanging information in the network, *matching* is a concept at the federation level [45, 54, 56, 64, 65]. It describes a very important state of an area in *routing space*, specifically, that which occurs when different *federates* are publishing and subscribing to it during the same *timestep*. After *matching* is detected, user level information needs to be transmitted from the *publisher*s to the *subscriber*s.

Therefore, the main functionality of DDM can be summarized as; detecting *matching*, distributing *matching* information, and arranging the transmission of information in *matching* from *publisher*s to *subscriber*s. In DDM, this work is done by allocating a MGRP (Multi-cast Group) to the area in *matching*, and by sending triggering messages to all *publisher*s and *subscriber*s of that area and asking them to join that MGRP. Correspondingly, a reverse operation should be carried out once the *matching* condition is broken. All of this work is done by exchanging DDM messages within the *federation*.

3.1.5 Performance of DDM Implementations

As a simulation middle-ware, the performance of the HLA/DDM implementation has a direct effect on the overall simulation performance. Traditionally, we use DDM time, DDM messages and MGRP allocated as three main parameters to evaluate the performance of a DDM implementation.

DDM time is the total time used by the DDM system in a simulation. It includes the time spent on the registration of publishing and subscribing requests and its reverse process, *matching* detection, MGRP allocation and de-allocation, and triggering *publishers/subscribers* to join or leave a MGRP. DDM time is regarded as the total response time of a DDM implementation to its higher level applications.

DDM messages refers to the messages transmitted by the DDM system. DDM messages are different from the user messages transmitted by higher level applications in that they are used to fulfill the goal of the DDM of a simulation.

MGRP allocated refers to the multi-cast groups allocated by the DDM system[45, 56, 57]. These multi-cast groups are used by higher level applications for multi-casting user messages, and they depend on the *matching* performance of a simulation.

3.2 Related Works

Matching detection is the main computing problem in finding a pair of senders and receivers in the same area in a DDM system, while MGRP allocation is the main resource management problem. Existing DDM systems have addressed these problems very differently depending on their various performance concerns. In this section, we will give a quick review of these DDM systems based on a comparison of their solutions to these two problems.

3.2.1 Region-based DDM Approach

In the Region-based DDM approach, a region in *routing space* is represented by a set of extents using the multi-dimensional coordinate system, *matching* is computed exhaustively, and the exact *matching* region can be found. However, since no single *federate* has a complete knowledge of how many other *federate*s are publishing or subscribing to a target region, a central coordinator has to be chosen to perform the *matching* detection. Therefore, whenever there is a change to the publication or subscription regions in a *federate*, a notification has to be sent to the central coordinator. Because a simulation will normally have a large number of simulated *object*s, and the movement of each *object* can cause frequent changes in the publication and subscription regions, a considerable amount of communication load towards the central coordinator will be added to the network. In addition, an extremely large amount of computation work will be loaded onto the central coordinator.

One advantage of the Region-based DDM approach is that the exact *matching* region can be found. Therefore, the MGRP is allocated whenever it is necessary, and the *publisher* multi-casts the exact information that is required by the *subscribers*.

3.2.2 Grid-based DDM Approaches

In the grid-based DDM system, *routing space* is divided into a certain number of grid cells, and publication and subscription regions are represented by a set of cell IDs. In place of exhaustive computing, *matching* is found by checking a pair of *publisher*s and *subscriber*s within each cell. However, since a region that is represented using the multi-dimensional coordinate system tends to fall outside the boundaries of grid cells, a region represented using a set of grid cells normally appears larger than its region-based representation. Depending on what kind of MGRP allocation strategy is used, the Grid-based DDM approaches have two different implementations.

3.2.2.1 Fixed Grid-based DDM Approach

The Fixed Grid-based approach is a simple solution for DDM. In this approach, *matching* detection is not implemented, and MGRPs are pre-allocated and bounded to each grid cell in the initialization phase of the simulation. All *federate*s have a complete knowledge of the MGRP information of each cell. Once a *publisher* is interested in a cell, it joins its MGRP and starts publishing regardless of whether there is a *subscriber*; correspondingly, once a *subscriber* arrives at a cell, it joins the cell's MGRP and starts listening regardless of whether there is a *publisher*.

Obviously, the Fixed Grid-based approach is not as feasible in a real simulation because it generates too many aimless messages and consumes a large amount of resources. Right after the work of this fixed approach was completed, an improved dynamic approach with *a matching* detection mechanism was carried out. This approach is based on distributing the ownership of grid cells to each *federate* and implementing a triggering mechanism for joining the MGRP.

3.2.2.2 Dynamic Grid-based DDM Approach

Two new mechanisms are employed by the Dynamic Grid-based DDM approach:

Cell's ownership distribution mechanism: In the Dynamic Grid-based DDM approach, grid cells have their ownerships uniformly distributed to each *federate*. Any publishing or subscribing request to a target cell needs to be registered to the owner of the cell. Therefore, each *federate* has complete knowledge of all publishing and subscribing activities that occur in the cells it owns. Depending on the cell's ownership distribution pattern, *matching* computing can be distributed evenly or with some other pattern within the *federation*, thus eliminating the need for a central coordinator.

Triggering mechanism: Since each *federate* has only part of the *matching* information *federation*-wide, once the *matching* of a cell has been detected, the owner is responsible for allocating a MGRP for this cell. The triggering mechanism is used to notify all *publisher*s and *subscriber*s to join the MGRP and prepare to multi-cast user messages.

By employing the cell's ownership distribution mechanism, the Dynamic Grid-based DDM scheme distributes the work of *matching* computing to each *federate*. All publishing and subscribing activities and their reverse operations need not only to be processed locally at each *federate*, but also to be registered with the owner of the target cell. Thus, each *federate* performs part of the *matching* detection *federation*-wide. When the *matching* of a cell is detected by its owner, the MGRP is dynamically allocated and the triggering mechanism is used to trigger all *publisher*s and *subscriber*s to join this MGRP. In response to the triggering message, *publisher*s and *subscriber*s register themselves to the MGRP at the owner and begin multi casting user messages.

The advantages of the Dynamic Grid-based DDM approach are obvious: the resource allocation is more efficient because the multi-cast groups are dynamically allocated when necessary; the communication overhead is distributed within the *federation* because the cells' ownership has been distributed to each *federate* and the central coordinator has been removed; the computation overhead has been reduced because exhaustive *matching* computing is not necessary in the grid-based system; and the total workload is distributed to each *federate.*

The weakness of this approach is the reduced accuracy introduced by the grid-based DDM system. As we mentioned in the beginning of this section, the publication/subscription region normally appears larger when using grid-based representation than when using region-based representation; specifically, this means that the *publisher* may send more user messages than are actually required by the *subscriber*s. Therefore, the performance gain of the Dynamic Grid-based DDM approach is obtained with a trade-off in additional traffic at the user level.

3.2.3 Other DDM Schemes

As we have seen, the Region-based DDM approach achieves exact *matching* with considerable computation and communication overhead, while the Dynamic Grid-based DDM approach sacrifices accuracy to obtain a gain in DDM performance. However, several other schemes such as the Grid-filtered Region-based DDM approach and the Agent-based DDM approach have been devised to balance this trade-off. The main idea is to filter user messages at the sender side to avoid transmitting unnecessary data to the *subscriber*s. Since these schemes are far away from what we are considering in our thesis work, we do not discuss any of their details here.

All of existing DDM implementations that we have examined so far have been working on balancing the trade off between the communication overhead and the computation complexity. No effort has ever been made to evaluate the size of a distributed simulation. In other words, existing DDM solutions manipulate data in a simulation without predicting how much data can be generated. Due to the lack of awareness about the size of a simulation, the efficiency of these implementations has never been tested fairly. Starting in the next subsection, we will introduce the resource allocation and transmission control problems found in our work, and show how to model the distributed environment to solve these problems. Implementations derived from these analysis have been carried out, and

the experimental results demonstrate outstanding performance compared to all existing DDM approaches.

3.3 Foundations of Grid-based DDM

In order to evaluate the average amount of data generated in each simulation step, this section will clarify a few problems and define some important concepts that have not drawn much attention in existing grid-based DDM approaches.

3.3.1 Simulation Scenario

In our work, we use the "tanks' dogfight" simulation scenario implemented in the RTI-Kit[45, 52, 67] that was originally developed by Georgia Institute of Technology as our testing environment. Specifically, we consider a simulation running in a two dimensional, grid-based, square *routing space*. The cell's ownership distribution mechanism is applied to the grid cells, and each *federate* manages a certain number of *object*s. At the beginning of the simulation, all *federate*s distribute their *object*s across *routing space*, and each *object* then moves in one out of four directions (East, West, South, and North) in each *timestep* of the simulation. An *object* moves by subscribing to a number of cells centered at its location and publishing to a number of cells centered at another location. The direction of movement will be reconsidered after each *timestep* and an *object* will stop at the border of *routing space* until a different direction is chosen.

The reason for which we chose the grid-based DDM environment as our study environment is due to the fact that it is actually a simplified form of the region-based DDM environment, it does not require a central coordinator, and data can be managed in a purely distributed manner. The grid-based DDM is therefore more feasible for mathematical modeling.

In the scenario described above, we did not give any details about the distribution

patterns we will employ because they do not affect how we will carry out our analysis. However, they do affect the final results of the estimation. The distribution patterns in this scenario include the following: (1) the distribution pattern of the cells' ownership, (2) the distribution pattern of the *object*s' ownership, (3) the pattern of the initial distribution of *object*s across *routing space,* (4) the distribution pattern of the directions of movement, (5) the relationship between the locations of the publication region and the subscription region of the same *object*, and (6) the relationship between the behaviors of all *object*s in the simulation. The first two distribution patterns determine the computation and communication load at each *federate*, they are the distributed processing work assigned to them. The rest of the distribution patterns determine the overall behavior of *object*s in the simulation. It is certain that the final estimation results will be different if we manipulate these distribution patterns.

3.3.2 Matching Accuracy

We have mentioned that the grid-based systems sacrifice *matching accuracy* to reduce the communication overhead and computation load at each *federate*. However, it is still unclear how much *matching accuracy* can be achieved by a grid-based system. This problem is important to our current study because we need to scale the size of a simulation in the grid-based DDM environment. If we do not consider the effect of filtering mechanism, the accuracy problem in the grid-based system is easy to answer.

Proposition 3.3.1. ***(Minimum accuracy)*** *The minimum accuracy that can be reached by a grid-based DDM system is the size of one grid cell.*

Proof. Since one grid cell is the atomic unit for publishing or subscribing activities in a grid-based DDM system, the entire grid cell is considered to be published or subscribed to once there is a *publisher* or *subscriber* publishing or subscribing to any part of it. The

details regarding the actual size of this area inside the cell are ignored. Therefore, the minimum accuracy of a Grid-based DDM system is the size of one grid cell. □

Because of Proposition 3.3.1, we introduce a basic measurement in the grid-based environment: *unit distance*. The definition of *unit distance* is as follows:

Definition 3.3.2. (Unit distance) *Unit distance* is the distance across one grid cell.

In the rest part of thesis, when we talk about the distance or movement of *object*s, they will be measured in *unit distance.*

3.3.3 Timestep

RTI-based simulations are discrete event simulations. The concept of *timestep* has long been used to measure the logical time of an RTI-based simulation, but has not drawn too much attention before now, as it was formerly set randomly or at will. This lack of attention may be due to the different focuses of such studies. However, it is of great value to our current concern. In subsection 3.1.4.1, we informally introduced the concept of *timestep* based on existing DDM approaches [45, 52]. We now introduce the formal definition of *timestep* in the grid-based system as follows:

Definition 3.3.3. (Timestep) *Timestep is the basic time unit used by the slowest object to cross one grid cell in grid-based DDM systems.*

3.3.4 Uniform Distribution Patterns

According to the simulation scenario described previously, if we assume that (1) the ownership of grid cells is uniformly distributed among all *federate*s, (2) each *federate* owns an equal number of *object*s, (3) the *object*s at each *federate* are uniformly distributed

across *routing space,* (4) four directions of movement for each *object* are chosen evenly and likely, (5) the locations of the publication region and subscription region of an *object* in *routing space* are independent, and (6) the behaviors of all *object*s in a *federate* are independent as well as the behaviors of all *federate*s in the *federation*, then we have a set of uniform distribution patterns.. The uniform distribution patterns also imply that all *object*s in the simulation have the same speed of movement. In other words, all *object*s have the same *timestep* and it is set to the *unit distance* (d).

Throughout this part of the thesis, we apply the uniform distribution patterns to the simulation scenario in order to show our work. The generalization of this scenario will be carried out soon in our future work.

3.4 Matching Model

3.4.1 Static Data

In a distributed simulation, a *federate* plays two different roles in each *timestep*. As the owner of its *object*s, it is responsible for distributing these *object*s across *routing space*, then driving them to move inside *routing space*, and keeping track of the location of each *object*. It is also responsible for sending new publishing and subscribing requests to the owners of all target cells, as well as un-publishing and un-subscribing requests to the cells being left. As the owner of cells, it accepts all publishing and subscribing requests to the target cells it owns, as well as all un-publishing and un-subscribing requests sent to these cells. For this reason, each *federate* is responsible for keeping track of all publishing and subscribing activities federation-wide in every cell it owns, and it performs the *matching* detection in these cells. If a new *matching* state of a cell is detected, the owner *federate* allocates a MGRP for this cell, and sends triggering messages to all registered *publisher*s and *subscriber*s in this cell asking them to join that MGRP. If a cell that has been in *matching* since the last *timestep* has new arrival or departure activities, the owner *federate*

sends a notification about the new changes to all registered *publisher*s and *subscriber*s in this cell.

3.4.1.1 Definition of Static Data

In a simulation, the movement of *object*s leads to frequent changes in the publication and subscription regions. *Static data* is generated because (1) as the owner of some *object*s, the *federate* needs to record all publishing and subscribing activities of the *object*s it owns; and (2) as the owner of some grid cells, the *federate* needs to record all publishing and subscribing activities in these cells *federation-wide*, as well as the *matching* information. Therefore, *static data* is used to refer to the storage allocated at a *federate* for the local and distributed processing work.

According to the above discussions, the first fact is obtained from direct observation:

Fact 3.4.1. *Static data determines the local storage allocated at a federate.*

Actually, the name *static data* does not represent the property of the data we are studying. It is static because we will use a static probability model to evaluate the average amount of this data, and this static model models the steady-state of a grid cell in the grid-based DDM environment.

3.4.1.2 Inside Static Data

In a simulation, depending on the different roles that a *federate* may play, *static data* can be categorized into three parts. (1) As the *object*s' owner, the *local Pub/Sub data* is used to save the publishing/subscribing activities of the local *object*s it owns (*object* level). (2) As the cell's owner, the *distributed Pub/Sub data* is used to save the publishing/subscribing activities *federation-wide* (*federate* level). (3) Moreover, once the cell's owner detects that a cell is in *matching*, a MGRP is allocated for all registered *publishers* and *subscribers*. Because this *matching* information is necessary for multi-casting user

messages, it has to be sent to all registered entities by the cell's owner. Therefore, as a *publisher* or *subscriber* of a cell in *matching*, the *matching data* is used to save the *matching* information received (*federate* level).

Normally, the total amount of *static data* does not vary noticeably. The reason for this is easy to explain: during a given time period within a simulation, the number of total *federate*s involved is fixed, the number of total simulated *objects* is fixed, and the publication and subscription region sizes of each *object* are also fixed. Therefore, we obtain the second fact from direct observation:

Fact 3.4.2. *If the distribution pattern of a simulation is given, the total amount of static data is predictable.*

From Facts 3.4.1 and 3.4.2, we are able to conclude that the storage usage at each federate is scalable and can be computed according to the input size of a simulation.

3.4.2 Performance Analysis

3.4.2.1 Evaluating Static Data

According to the discussions in Subsection 3.4.1, *static data* is generated because some grid cells have publishing/subscribing activities. Therefore, the number of these grid cells is directly related to the amount of *static data*. In order to keep the implementation details out of our study, we use *unit space* to denote the atomic storage required by the publishing/subscribing activities relating to one grid cell. In this part, the *static data* is evaluated in terms of the number of *unit spaces*.

Let C be the total number of grid cells in *routing space*; f be the total number of *federate*s involved; and $STG_{loccal\,PubSub}$, $STG_{dist\,PubSub}$, and STG_{match} be the average storage allocated for the *local Pub/Sub data*, the *distributed Pub/Sub data,* and the

Matching data in a *federate i* respectively. Then,[1]

by the uniform cell's ownership distribution mechanism,

$$[\#of\ cells\ owned\ by\ Fed\ i] = C/f$$

and the average storage allocated for the *static data* is

$$STG_{local_{PubSub}} \approx C \times (Pr[a\ cell\ be\ Pub\ by\ Fed\ i] + Pr[a\ cell\ be\ Sub\ by\ Fed\ i]) \quad (3.1)$$

$$STG_{dist\,PubSub} \approx \frac{C}{f} \times (Pr[a\ cell\ be\ Pub\ by\ any\ Fed] + Pr[a\ cell\ be\ Sub\ by\ any\ Fed]) \quad (3.2)$$

$$STG_{match} \approx STG_{local\,PubSub} \times Pr[matching] \quad (3.3)$$

In the above formulas, three terms are unknown: informally, these are the publishing, subscribing, and *matching* probabilities. In order to evaluate the storage required for the *static data* generated in a simulation, we need to build a static probability model called the *matching model* and describe how to compute these probabilities based on the simulation input.

3.4.2.2 Evaluating DDM Performance

The objective of this section is to design an efficient and scalable resource allocation scheme for grid-based DDM systems that is adaptive to different simulation sizes and best approaches the optimum performance. Therefore, the storage allocation rate is used instead of the MGRPs allocated to evaluate the efficiency of the storage allocation of a DDM implementation. The storage allocation rate can be expressed as

$$Storage\,allocation\,Rate = r_{STG} = \frac{Storage\,allocated}{Storage\,required}$$

For a simulation with a specific input size, the storage allocated refers to the storage actually allocated at a node for the local processing work and the distributed processing

[1] In the expressions, Fed: federate; Pub: published; Sub: subscribed

work, while the storage required refers to the storage theoretically required by the same node for this specific input size. Because the MGRPs allocated are part of the storage allocated, we believe that the storage allocation rate is more meaningful in evaluating the efficiency of a DDM implementation. In theory, the optimum DDM implementation leads to an ideal storage allocation rate, which is

$$r'_{STG} = 1$$

In practice, our objective for this parameter is

$$r_{STG} \rightarrow 1 \; and \; \geq 1$$

3.4.3 Matching Model

The *matching model* consists of three basic probabilities: the publishing probability, the subscribing probability, and the *matching probability*. In the previous section, we mentioned that these probabilities are the measurement of the static performance of a simulation and are used to determine the amount of storage allocated for local and distributed processing work. The objective of this section is to formalize these probabilities. We start from the formal definitions of these basic concepts.

3.4.3.1 Definitions

In the DDM system, publishing/subscribing probabilities have three levels of meaning: the *object* level, the *federate* level, and the *federation* level respectively.

Definition 3.4.3. (Publishing/Subscribing probability)

- *At the object level, the publishing/subscribing probability is the probability of a cell being published or subscribed to by one object; correspondingly, this object is the*

publisher or subscriber of this cell. The probability at this level is called the Object Publishing/Subscribing Probability (OPP/OSP).

- *At the federate level, the publishing/subscribing probability is the probability of a cell being published or subscribed to by at least one object of a federate; correspondingly, this federate is the publisher or subscriber of this cell. The probability at this level is called the Federate Publishing/Subscribing Probability (FPP/FSP).*
- *At the federation level, the publishing/subscribing probability is the probability of a cell being published or subscribed to by at least one federate. Normally, it is known as the publishing/subscribing probability.*

However, the *matching probability* is a concept defined only at the *federation* level. It can be considered as the accumulated effect of the publishing and subscribing probabilities from the *object* level to the *federation* level.

Definition 3.4.4. (Matching probability) *The matching probability is the probability of a cell being published and subscribed to by different federates.*

In this section, we will use the uniform distribution patterns given in Section 3.3 as an example to carry out our analysis.

3.4.3.2 Publishing/Subscribing Probability

According to the three-level definition of the publishing/subscribing probability, there are three probability expressions that correspond to each level.[2]

$$
\begin{aligned}
&Pr[\textit{publishing/subscribing at the federation level}] \\
&= Pr[\textit{cell k is Pub/Sub by at least one Fed}] \\
&= 1 - Pr[\textit{cell k is not Pub/Sub by all Feds}] \\
&= 1 - \prod_{\text{all Fed}}^{i} Pr[\textit{cell k not be Pub/Sub by Fed i}]
\end{aligned} \tag{3.4}
$$

$$
\begin{aligned}
&Pr[\textit{publishing/subscribing at the federate level}] \\
&= Pr[\textit{cell k is Pub/Sub by at least one Obj in Fed i}] \\
&= 1 - Pr[\textit{cell k is not Pub/Sub by all Obj at Fed i}] \\
&= 1 - \prod_{\substack{\text{all Obj in} \\ \text{Fed i}}}^{j} Pr[\textit{cell k is not Pub/Sub by Obj j}]
\end{aligned} \tag{3.5}
$$

$$
\begin{aligned}
&Pr[\textit{publishing/subscribing at the object level}] \\
&= \frac{\textit{Average object publication/subscription region size}}{\textit{Total number of routing space cells}}
\end{aligned} \tag{3.6}
$$

In Expression 3.6, we use the term *average object publication/subscription region size* to compute the *OPP/OSP*. The reason for this is the *publication/subscription loss* at the border area of *routing space,* because some publication or subscription area might fall out of the range of *routing space*.

This phenomenon introduces another new factor in the study of the *matching model*. As before, we let $C = c \times c$ be the cells in *routing space*, and if we let $S = s \times s$ be the publication/subscription region size of an *object*, then the maximum publication/subscription

[2]Fed: federate; Pub: published; Sub: subscribed; Obj: object

region size is $S' = (s+1) \times (s+1)$. The *average object publication/subscription region size* $\mathbf{E}(S)$ can be computed as

$$E(S) = r \cdot S' = r \cdot (s+1)(s+1) \tag{3.7}$$

In Equation 3.7, the term r is called APR/ASR (Average Publishing/Subscribing Rate*)*. The derivation of r is simply a manipulation of *routing space* cells and cells that are being published or subscribed to. Normally, when c is large

$$r \approx (1 - \frac{s+1}{4c})^2 \quad where\, s \leq 4c - 1 \tag{3.8}$$

Let f be the total number of *federate*s, n_i be the total number of *object*s at *federate* i, and $S_p = s_p \times s_p$, $S_S = s_s \times s_s$ be the publication/subscription region size of an *object*. In addition, let α_F, β_F be the publishing/subscribing probability at the *federation* level, α_i, β_i be these probabilities at the *federate* level, and $p_{i,j}$, $q_{i,j}$ be the corresponding probabilities at the *object* level (the *j-th* object at *federate* i). According to expressions 3.4, 3.5, and 3.6, we have

$$\alpha_F = 1 - \prod_{i=1}^{f}(1 - \alpha_i) \quad \text{and} \quad \beta_F = 1 - \prod_{i=1}^{f}(1 - \beta_i) \tag{3.9}$$

$$\alpha_i = 1 - \prod_{j=1}^{n_i}(1 - p_{i,j}) \quad \text{and} \quad \beta_i = 1 - \prod_{j=1}^{n_i}(1 - q_{i,j}) \tag{3.10}$$

$$p_{i,j} = \frac{\mathbf{E}(S_{p,i,j})}{C} \quad \text{and} \quad q_{i,j} = \frac{\mathbf{E}(S_{S,i,j})}{C} \tag{3.11}$$

3.4.3.3 Matching Probability

According to the formal Definition 3.4.4, the *matching probability* is derived directly from the publishing/subscribing probability at both the *federation* and *federate* levels:

$$\begin{aligned} & Pr[Matching] \\ &= (Pr[a\ cell\ is\ Pub\ by\ at\ least\ one\ Fed] \times Pr[a\ cell\ is\ Sub\ by\ at\ least\ one\ Fed]) \\ &\quad - Pr[a\ cell\ is\ Pub\ and\ Sub\ by\ only\ one\ Fed] \end{aligned} \tag{3.12}$$

Let γ represent the *matching probability*. According to Equation 3.9, we obtain

$$\gamma = [1 - \prod_{i=1}^{f}(1-\alpha_i)] \cdot [1 - \prod_{i=1}^{f}(1-\beta_i)] - \sum_{i=1}^{f} \alpha_i\beta_i \cdot \prod_{j\mp i}^{f}(1-\alpha_j)(1-\beta_j) \tag{3.13}$$

When f is large, we have the approximation

$$\gamma \approx [1 - \prod_{i=1}^{f}(1-\alpha_i)] \cdot [1 - \prod_{i=1}^{f}(1-\beta_i)] \tag{3.14}$$

3.4.3.4 Lower Boundary Analysis

In the previous two subsections, we established performance equations 3.9, 3.10, 3.11, and 3.13 for computing the publishing/subscribing probability and the *matching probability*. If we look inside these equations, they are quite disordered because each *federate* can have a different number of *object*s and both the *publisher* and *subscriber* can behave very differently. In fact, these equations are expressed with respect to a real simulation situation and are not as efficient or meaningful for computing and analysis purposes. In order to render our computation and results more practical, it is necessary to introduce the *Lower Boundary Analysis* to the *matching model*.

We start from the publishing/subscribing probability at the *object* level. By applying

Equations 3.7 and 3.8, Equation 3.11 can be expressed as a function in s:

$$p_{i,j} = f(s_{p,i,j}) \text{ and } q_{i,j} = f(s_{s,i,j}) \tag{3.15}$$

When c is a constant, it is not difficult to show that $p = f(s_p)$ and $q = f(s_s)$ are real continuous functions over the interval $(0, 4c - 1]$. If we let $\bar{p_i}$ and $\bar{q_i}$ be the average publishing/subscribing probability of an *object* at *federate* i, we have

$$\bar{p_i} = \frac{1}{n_i} \times \sum_{j=1}^{n_i} f(s_{p,i,j}) \geq f(\frac{\sum s_{p,i,j}}{n_i}) = f(\bar{s_{p,i}}) \quad \text{and}$$

$$\bar{q_i} = \frac{1}{n_i} \times \sum_{j=1}^{n_i} f(s_{s,i,j}) \geq f(\frac{\sum s_{s,i,j}}{n_i}) = f(\bar{s_{s,i}}) \tag{3.16}$$

Moreover, by applying the General Means Inequality and Inequality 3.16 to Equation 3.10, we obtain the lower boundaries of the publishing probability and the subscribing probability of the *federate i:*

$$\begin{aligned}
\alpha_i &= 1 - \prod_{j=1}^{n_i}(1 - p_{i,j}) \\
&\geq 1 - [\frac{1}{n_i}\sum_{j=1}^{n_i}(1 - p_{i,j})]^{n_i} = 1 - (1 - \bar{p_i})^{n_i} \\
&\geq 1 - [1 - f(\bar{s_{p,i}})]^{n_i} = \alpha_i' \quad \text{and}
\end{aligned}$$

$$\begin{aligned}
\beta_i &= 1 - \prod_{j=1}^{n_i}(1 - q_{i,j}) \\
&\geq 1 - [\frac{1}{n_i}\sum_{j=1}^{n_i}(1 - q_{i,j})]^{n_i} = 1 - (1 - \bar{q_i})^{n_i} \\
&\geq 1 - [1 - f(\bar{s_{s,i}})]^{n_i} = \beta_i' \qquad (3.17)
\end{aligned}$$

Similarly, in order to obtain the lower boundary of the *matching probability*, we let

$\bar{\alpha} = \frac{1}{f}\sum_{i=1}^{f}\alpha_i'$, $\bar{\beta} = \frac{1}{f}\sum_{i=1}^{f}\beta_i'$, and apply the General Means Inequality and Inequality 3.17 to the approximation Equation 3.14. Then we have

$$\begin{aligned}
\gamma &\approx [1 - \prod_{i=1}^{f}(1-\alpha_i)] \cdot [1 - \prod_{i=1}^{f}(1-\beta_i)] \\
&\geq [1 - \prod_{i=1}^{f}(1-\alpha_i')] \cdot [1 - \prod_{i=1}^{f}(1-\beta_i')] \\
&\geq \{1 - [\frac{1}{f}\sum_{i=1}^{f}(1-\alpha_i')]^f\} \cdot \{1 - [\frac{1}{f}\sum_{i=1}^{f}(1-\beta_i')]^f\} \\
&\geq [1 - (1-\bar{\alpha})^f] \cdot [1 - (1-\bar{\beta})^f] \qquad (3.18)
\end{aligned}$$

If we let $\bar{\alpha_F} = 1 - (1-\bar{\alpha})^f$ and $\bar{\beta_F} = 1 - (1-\bar{\beta})^f$, then

$$\gamma \geq \bar{\alpha_F} \cdot \bar{\beta_F} \qquad (3.19)$$

In the implementation of our new Adaptive Grid-based DDM approach, if we let (C, f, n/f, $\bar{S_P}$, $\bar{S_S}$) be one simulation input case, in which $C = c \times c$ is the grid size of *routing space*, f is the number of total *federate*s, n is the number of total *object*s, $n_i = n/f$ is the number of *object*s in each *federate*, and $\bar{S_P} = \bar{s_p} \times \bar{s_p}$ and $\bar{S_S} = \bar{s_s} \times \bar{s_s}$ are the publication and subscription region sizes of each *object*, then, according to the lower boundary inequalities 3.17 and 3.18, we have the following fact

Fact 3.4.5. *By applying the Lower Boundary Analysis to the Adaptive Grid-based DDM approach, the performance*[3] *of one simulation input* (C, f, n/f, $\bar{S_P}$, $\bar{S_S}$) *represents the lower boundary result of a series of simulation input cases in which the total number of grid cells is C, the total number of federates is f, the average number of objects in each federate is n/f, and the average publication and subscription region sizes of each object*

[3]Performance is measured by using α , β, and γ in this thesis.

are $\bar{S_P}$ *and* $\bar{S_S}$.[4]

3.4.3.5 Adaptive Resource Allocation Control Scheme

In subsection 3.4.2.1, we discussed theoretically the storage required[5] at each *federate* for a specific simulation input. Now, we are able to summarize and rewrite those expressions.

$$STG_{local\,PubSub} \approx C \times (\bar{\alpha} + \bar{\beta}) \tag{3.20}$$

$$\begin{aligned} STG_{dist\,PubSub} &\approx \frac{C}{f} \times \{[1-(1-\bar{\alpha})^f] + [1-(1-\bar{\beta})^f]\} \\ &= \frac{C}{f} \times [\bar{\alpha_F} + \bar{\beta_F}] \end{aligned} \tag{3.21}$$

$$\begin{aligned} STG_{match} &\approx STG_{local\,PubSub} \times [1-(1-\bar{\alpha})^f] \times [1-(1-\bar{\beta})^f] \\ &= STG_{local\,PubSub} \times (\bar{\alpha_F} \cdot \bar{\beta_F}) \end{aligned} \tag{3.22}$$

Therefore, theoretically, the total storage required is simply the summation of the above three results:

$$STG_{req} = STG_{local\,PubSub} + STG_{dict\,PusSub} + STG_{match} \tag{3.23}$$

Since we are using *Lower Boundary Analysis* to evaluate the performance of a series of simulation inputs, the actual storage used may vary from the theoretical result. In addition, in practice, no matter how small the input size of a simulation is, it requires a minimum amount of storage for starting up. Therefore, these formulas need certain adjustments to adapt to the real simulation requirements. The adjustment normally depends on the details of the implementation. In our coding, we add one factor for the

[4] In order to avoid confusion, we use $E(S_{P,i,j})$ and $E(S_{S,i,j})$ to represent the average object publication/subscription region size because of the *publication/subscription loss* at the border area, while $\bar{S_P}$ and $\bar{S_S}$ represent the average publication/subscription region size computed based on all simulated *objects.*

[5] In terms of *unit space.*

extra storage required due to variations in storage usage as well as another factor for the minimum amount of storage required. Both value are set to 0.05 for now.

$$STG'_{local\ PubSub} = \begin{cases} 0.10C & 0 \leq \bar{\alpha} + \bar{\beta} < 0.05 \\ STG_{local\ PubSub} + 0.05C & 0.05 \leq \bar{\alpha} + \bar{\beta} < 0.95 \\ C & 0.95 \leq \bar{\alpha} + \bar{\beta} \leq 1 \end{cases} \tag{3.24}$$

$$STG'_{dist\ PubSub} = \begin{cases} 0.10\frac{C}{f} & 0 \leq \bar{\alpha_F} + \bar{\beta_F} < 0.05 \\ STG_{dist\ PubSub} + 0.05\frac{C}{f} & 0.05 \leq \bar{\alpha_F} + \bar{\beta_F} < 0.95 \\ \frac{C}{f} & 0.95 \leq \bar{\alpha}_F + \bar{\beta_F} \leq 1 \end{cases} \tag{3.25}$$

$$STG'_{match} = \begin{cases} 0.10STG'_{local\ PubSub} & 0 \leq \bar{\alpha_F}\bar{\beta_F} \leq 0.05 \\ STG_{match} + 0.05STG'_{local\ PubSub} & 0.05 \leq \bar{\alpha_F}\bar{\beta_F} \leq 0.95 \\ STG'_{local\ PubSub} & 0.95 \leq \bar{\alpha}_F\bar{\beta_F} \leq 1 \end{cases} \tag{3.26}$$

After the adjustment, the total storage allocated is

$$STG_{alloc} = STG'_{local\ PubSub} + STG'_{dist\ PubSub} + STG'_{match} \tag{3.27}$$

In the following section, we will use experimental data to prove that the optimum solution to the storage allocation problem has been best approached by the ARAC scheme.

3.5 Switching Model

3.5.1 Dynamic Data

In a distributed simulation, a *federate* plays two different roles in each *timestep*. As the owner of its *object*s, it is responsible for distributing *object*s across *routing space*, then driving these *object*s to move inside *routing space*, keeping track of the location of each *object*, and sending new publishing and subscribing requests to the owners of all target cells, as well as sending un-publishing and un-subscribing requests. As the owner of a cell, on the other hand, the *federate* accepts all publishing and subscribing requests to the target cells it owns, as well as all un-publishing and un-subscribing requests. Because of this role, each *federate* is responsible for keeping track of the publishing and subscribing

activities federation-wide in each cell it owns, and it performs the *matching* detection in these cells. If a new *matching* of a cell is detected, the owner *federate* allocates a MGRP for this cell, sends triggering messages to all registered *publisher*s and *subscriber*s in this cell, and asks them to join the MGRP. If a cell that has been in *matching* since the last *timestep* has new arrival or departure activities, the owner *federate* sends a notification about the new changes to all registered *publishers* and *subscriber*s in this cell.

3.5.1.1 Two Phases of Transmission

Because a *federate* plays two different roles in a distributed simulation, the transmission of DDM messages in each *timestep* is also divided into two different phases. In the first phase of transmission, a *federate* plays the role of the owner of its *objects* and behaves like the *publisher* or *subscriber* of grid cells. The senders of the transmissions in this phase are new *publisher*s or *subscriber*s of grid cells at the *federate* level, while the receivers are the owner *federate*s of these grid cells. The goal of the first phase of transmission is to register or un-register publishing and subscribing requests to the owner of the target cells.

In the second phase of transmission, a *federate* plays the role of the owner of the grid cells that have been distributed to it. Since it has obtained *federation*-wide knowledge of the new publishing and subscribing activities in the cells it owns through the first phase of transmission, it now performs the *matching* detection and all consequent tasks. The senders of the transmission in this phase are the owners of grid cells, while the receivers are the *publishers* or *subscribers* in the grid cells in the same *timestep* at the *federation* level. The goal of the second phase of transmission is to trigger the *publishers* and *subscribers* of a cell that is found in *matching* to join the MGRP, to notify the remaining *publishers* and *subscribers* in a cell that is still in *matching* about new arrivals or departures. or to trigger the remaining *publishers* or *subscribers* of a cell that is no longer in *matching* to leave the MGRP.

3.5.1.2 States of a Cell

The steady-state of a cell is the state of a cell within only one *timestep*, and it can't change in the same *timestep*.

The steady-state of a cell describes how many *publisher*s and *subscriber*s are publishing or subscribing to that cell. In this thesis, we use (n_{pub}, n_{sub}), [n_{pub}, n_{sub}] and $\{n_{pub}, n_{sub}, m\}$ to represent the steady-state of a cell at the *object* level, the *federate* level, and the *federation* level respectively.

At the *object* level, n_{pub}, $n_{sub} \in \{0, 1\}$, and the steady-state of a cell refers to whether the cell is being published or subscribed to by a single *object*; at the *federate* level, n_{pub}, $n_{sub} \in \{0, ..., n_i\}$[6], and the steady-state of a cell refers to the number of *object*s owned by a single *federate* that are publishing or subscribing to the cell; at the *federation* level, n_{pub}, $n_{sub} \in \{0, ..., f\}$[7] and $m \in \{0, 1\}$, and the steady-state of a cell refers to the number of *federates* that have at least one *object* publishing or subscribing to the cell. In addition, at the *federation* level, if $m = 1$, then the cell's state is in *matching*; otherwise, it is not in *matching*.

At the *federation* level, because of the requirement of the *matching* condition, some state representations may not be valid. For example, $\{u, 0, 1\}$ and $\{0, v, 1\}$ are not valid representations, along with $\{u + 1, v, 0 : u, v \geq 1\}$ and $\{u, v + 1, 0 : u, v \geq 1\}$. On the other hand, both $\{1, 1, 1\}$ and $\{1, 1, 0\}$ are valid representations.

The *dynamic state* of a cell is the state-transition of a cell between two consecutive *timesteps*. It is called "dynamic" because the state of a cell may be affected by the movement of *object*s in two consecutive *timesteps*.

Because the *dynamic state* describes the transition of the state of a cell in two consecutive *timestep*s, it can be represented using two steady-state representations at a certain level. In the this thesis, the state-transitions are only limited to transitions between two valid cell states. In addition, if the character # appears in the state's representation

[6] n_i: the total number of *object*s owned by the $i - th$ *federate*.
[7] f: the total number of *federates*.

instead of a number, such as $[\#, n_{sub}]$, it means that the number in that location can be any number in its range that makes the representation valid.

3.5.1.3 Definition of Dynamic Data

Dynamic data is generated at a *federate* because the movement of *object*s in each *timestep* leads to the state change of a cell to which they had previously published or subscribed. As the owner of the *objects*, a *federate* needs to register new publishing and subscribing requests to the owner of the target cells, as well as the un-publishing and un-subscribing requests; also, as the owner of the grid cells, a *federate* needs to process the registration requests received, perform *matching* detection in these cells, and send triggering messages to all *publishers* or *subscribers* in a cell if necessary. Therefore, *dynamic data* at a *federate* refers to the total DDM messages generated at that *federate*.

According to the above discussions, the first fact is obtained from direct observations is the following:

Fact 3.5.1. *Dynamic data determines the DDM messages generated in each timestep at a federate.*

It is important to note that the name *dynamic data* does not refer to the actual property of data we are studying. Instead, we use this term because we are going to use a dynamic probability model to evaluate the average amount of such data, and this dynamic model demonstrates the behavior of the state-transition of a grid cell in the grid-based DDM environment.

3.5.1.4 Inside Dynamic Data

Figure *3.1* demonstrates the movement of three *object*s (one *publisher* and two *subscriber*s) in three different *federate*s (1, 2, *and* 3) from *timestep* $k-1$ to $k+1$. In this

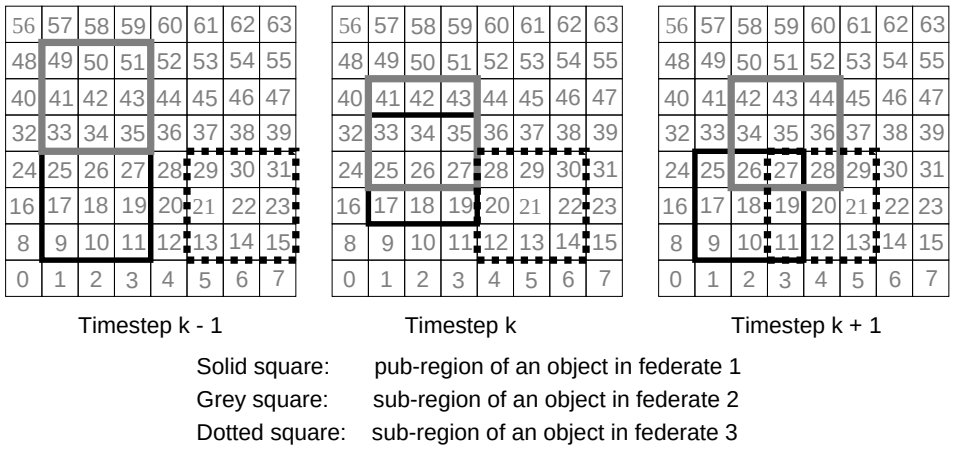


Figure 3.1: Dynamic Data Generated by the Movement of Objects (I)

figure, each grid cell is marked by its cell ID. We use this figure to show how *dynamic data* is generated by the movement of *object*s in a simulation.

New Pub/Sub data New Pub/Sub data is generated in the first phase of transmission. It is generated due to the transition of a cell's state from $[0, \#]$ to $[u, \# : u \geq 1]$, or from $[\#, 0]$ to $[\#, v : v \geq 1]$. Examples include cells $\{33, 34, 35\}$ of *federate* 1, cells $\{25, 26, 27\}$ of *federate* 2, and cells $\{28, 20, 12\}$ of *federate* 3 in *timestep* k of Figure *3.1.*

UnPub/UnSub data UnPub/UnSub data is generated in the first phase of transmission. It is generated due to the transition of a cell's state from $[u, \# : u \geq 1]$ to $[0, \#]$, or from $[\#, v : v \geq 1]$ to $[\#, 0]$. Examples include cells $\{9, 10, 11\}$ of *federate* 1, $\{49, 50, 51\}$ of *federate* 2, and $\{31, 23, 15\}$ of *federate* 3 in *timestep* k of the figure.

TellPubJoin and TellSubJoin data TellPubJoin and TellSubJoin data are generated in the second phase of transmission. There are two cases: the first occurs when the cell's state transits from $\{u, v, 0\}$ to $\{u', v', 1\}$ (cell $\{33, 34, 35\}$ and $\{25, 26, 27\}$ in *timestep* k of the figure), and the owner of this cell needs to allocate a MGRP for this cell and trigger all *publishers and subscribers* within it to join; the second occurs when a cell's state transits from $\{u, v, 1\}$ to $\{u', v', 1\}$ and has new arrival activities[8] (arrival of *federate* 3 in cell $\{27\}$ in *timestep* $k + 1$).

[8]Arrival of new *publishers* or *subscribers.*

In the subsequent process, once a *publisher* or *subscriber* receives a *TellPubJoin* or *TellSubJoin* message, it also receives an entire list of current *publishers* and *subscribers* in this cell. We call this list the Cached List (CList).

TellLeave data TellLeave data is generated in the second phase of transmission. It is generated when the cell's state transits from $\{u, v, 1\}$ to $\{u', v', 0\}$ (cell {33, 34, 35, 25} in *timestep* $k+1$). As the owner of this cell, the *federate* needs to de-allocate the MGRP of this cell, and then trigger all remaining *publishers* or *subscriber*s in this cell to leave this MGRP.

RefreshCList data RefreshCList data is generated in the second phase of transmission. It is generated when a cell's state transits from $\{u, v, 1\}$ to $\{u', v', 1\}$ and has new arrival or departure activities[9] (arrival of *federate* 3 in cell {27} in *timestep* $k+1$). In this case, the CList held by existing *publishers* and *subscribers* (*federate* 1 and 2 in cell {27} in *timestep* k) in this cell is out of date and needs to be updated.

According to the classification we have described so far, *dynamic data* represents the dynamic features of a distributed simulation in which the movement of simulated *object*s leads to a situation in which *objects* are continuously publishing or subscribing to new cells, and correspondingly un-publishing or un-subscribing from old cells. The consequence of this effect is that, in each *timestep*, the states of some cells transit to other states, and therefore the owners and all interested *publishers/subscribers* of these cells must be notified about new changes within the same *timestep*.

During a certain time period within a simulation, because the number of total *federate*s, the number of total simulated *object*s, and the size of the publication and subscription regions of each *object* are all fixed, we obtain the second fact from direct observation:

Fact 3.5.2. *If the distribution pattern of a simulation is given, the total amount of dynamic data is predictable.*

[9] Arrival of new *publishers* or *subscribers,* or departure of existing *publisher*s or *subscriber*s.

Table 3.1: State-transition at Object and Federate Level

State-trans.	Prev state	Curr. state
up	$(0, \#)$	$(1, \#)$
uu	$(0, \#)$	$(0, \#)$
pu	$(1, \#)$	$(0, \#)$
pp	$(1, \#)$	$(1, \#)$
us	$(\#, 0)$	$(\#, 1)$
uu'	$(\#, 0)$	$(\#, 0)$
su	$(\#, 1)$	$(\#, 0)$
ss	$(\#, 1)$	$(\#, 1)$

Object Level

State-trans.	Prev. state	Curr. state
$Pr[u2p]$ or $u2p$	$[0, \#]$	$[u, \# : u \geq 1]$
$Pr[u2u]$ or $u2u$	$[0, \#]$	$[0, \#]$
$Pr[p2u]$ or $p2u$	$[u, \# : u \geq 1]$	$[0, \#]$
$Pr[p2p]$ or $p2p$	$[u, \# : u \geq 1]$	$[u', \# : u' \geq 1]$
$Pr[u2s]$ or $u2s$	$[\#, 0]$	$[\#, v : v \geq 1]$
$Pr[u2u']$ or $u2u'$	$[\#, 0]$	$[\#, 0]$
$Pr[s2u]$ or $s2u$	$[\#, v : v \geq 1]$	$[\#, 0]$
$Pr[s2s]$ or $s2s$	$[\#, v : v \geq 1]$	$[\#, v' : v' \geq 1]$

Federate Level

3.5.2 Performance Analysis

Normally, the amount of *dynamic data* generated in each *timestep* at a *federate* is evaluated in terms of the number of DDM messages that need to be transmitted to other *federate*s. In the grid-based DDM system, a DDM message is produced once the state change of a grid cell results in a notification that must be sent to the owner of that cell or to the *publishers* or *subscriber*s in that cell. Therefore, the number of DDM messages that must be transmitted from one *federate* to every other *federate* in each *timestep* can be computed by evaluating the number of cells that have the same ownership or the same *publishers* or *subscribers,* as well as one of these state changes.

3.5.2.1 The State-transition of a Cell

In Tables 3.1 and 3.2, we specify notations used for the state-transition probabilities at three different levels respectively. According to the probability theory, we have their relations as follows.

$$up + uu + pu + pp = us + uu' + su + ss = 1$$

$$u2p + u2u + p2u + p2p = u2s + u2u' + s2u + s2s = 1$$

$$p_{u2p} + p_{u2u} + p_{p2u} + p_{p2p} = p_{u2s} + p'_{u2u} + p_{s2u} + p_{s2s} = 1$$

$$u2m + u\_u + m2u + m2m = 1$$

It is easy to understand that the state-transition probabilities of $u2u$, $u2u'$, p_{u2u}, p'_{u2u}, and u_u do not have the potential to generate any DDM messages. However, we should pay more attention to $p2p$, $s2s$, p_{p2p}, p_{s2s}, and $m2m$, because although these state-transition probabilities do not lead to a state change, they have the potential to generate DDM messages in cases when there are new arrival or departure activities of *objects* or *federates* in the current *timestep*. Therefore, in the discussion of this section, we add the capitalized notations for each state-transition probability listed in Table 3.1 and 3.2 to emphasize the fact that the cells with state-transitions are those cells with new arrival or departure activities. Normally, we have

$$U2P = u2p,\ P2U = p2u,\ U2S = u2s,\ S2U = s2u$$

$$P_{U2P} = p_{u2p},\ P_{P2U} = p_{p2u},\ P_{U2S} = p_{u2s},\ P_{S2U} = p_{s2u}$$

$$U2M = u2m,\ M2U = m2u$$

but

$$P2P \leq p2p,\ U2U \leq u2u,\ S2S \leq s2s,\ U2U' \leq u2u'$$

$$P_{P2P} \leq p_{p2p},\ P_{U2U} \leq p_{u2u},\ P_{S2S} \leq p_{s2s},\ P'_{U2U} \leq p'_{u2u}$$

$$M2M \leq m2m,\ U\_U \leq u\_u$$

3.5.2.2 Evaluating New Pub/Sub Data and UnPub/UnSub Data

According to the classification of *dynamic data* presented in subsection 3.5.1.4, let $N(i,j)_{Pub}$ represent the *new Pub data* at *federate* i, which is the number of cells that have the same ownership (*federate* j) and that have new publishing activities in the current *timestep* by *objects* in *federate* i; similarly, let $N(i,j)_{Sub}$, $N(i,j)_{unPub}$, and $N(i,j)_{unSub}$ represent the *new Sub, UnPub* and *UnSub data* at *federate* i respectively. In the transmission of these data, the sender side (*federate* i) plays the role of a *publisher* or *subscriber* to the target cell, while the receiver side (*federate* j) plays the role of the owner of the target cell.

According to the distribution pattern applied to the simulation scenario, the cells' ownership is uniformly distributed to a total f number of *federate*s. The total number of grid cells is $C = c^2$, and each *federate* owns $C_f \approx C/f$ cells.

The number of DDM messages generated at source *federate* i that must be sent to each destination *federate* j in each *timestep* is approximately

$$N(i,j)_{Pub} = N(i,j)_{unPub} \approx (Pr[u2p]_i \times C)/f \tag{3.28}$$

$$N(i,j)_{Sub} = N(i,j)_{unSub} \approx (Pr[u2s]_i \times C)/f \tag{3.29}$$

By transmitting these data, each cell's owner (*federate* j) obtains *federation*-wide knowledge of the publishing and subscribing activities in all of the cells it owns.

3.5.2.3 Evaluating TellPubJoin, TellSubJoin Data

Similarly, let $N(j,i)_{tellPub}$ represent the *TellPubJoin data* at *federate* j, which is the number of cells owned by *federate* j that have the same *publisher* (*federate* i), whose states transit from $\{u, v, 0\}$ to $\{u', v', 1\}$, or from $\{u, v, 1\}$ to $\{u', v', 1\}$ and that have new arrival activities (*federate* i). Similarly, let $N(j,i)_{tellSub}$, $N(j,i)_{tellLeave}$, and $N(j,i)_{refCList}$ represent *TellSubJoin, TellLeave*, and *RefreshCList data* at *federate* j respectively. In

Table 3.2: State-transition at Federation Level

State-trans.	Prev. state	Curr. state
p_{u2p}	$\{0, \#, \#\}$	$\{u, \#, \# : u \geq 1\}$
p_{u2u}	$\{0, \#, \#\}$	$\{0, \#, \#\}$
p_{p2u}	$\{u, \#, \# : u \geq 1\}$	$\{0, \#, \#\}$
p_{p2p}	$\{u, \#, \# : u \geq 1\}$	$\{u', \#, \# : u' \geq 1\}$
p_{u2s}	$\{\#, 0, \#\}$	$\{\#, v, \# : v \geq 1\}$
p'_{u2u}	$\{\#, 0, \#\}$	$\{\#, 0, \#\}$
p_{s2u}	$\{\#, v, \# : v \geq 1\}$	$\{\#, 0, \#\}$
p_{s2s}	$\{\#, v, \# : v \geq 1\}$	$\{\#, v', \# : v' \geq 1\}$

Federation Level I

State-trans.	Prev state	Curr state
$Pr[u2m]$ or $u2m$	$\{u, v, 0\}$	$\{u', v', 1\}$
$Pr[u\_u]$ or u_u	$\{u, v, 0\}$	$\{u', v', 0\}$
$Pr[m2u]$ or $m2u$	$\{u, v, 1\}$	$\{u', v', 0\}$
$Pr[m2m]$ or $m2m$	$\{u, v, 1\}$	$\{u', v', 1\}$

Federation Level II

the transmission of these data, the sender side (*federate j*) plays the role of the owner of a target cell, while the receiver side (*federate i*) plays the role of a *publisher* or *subscriber* to the target cell.

For $N(j,i)_{tellPub}$ and $N(j,i)_{tellSub}$, we have

$$N(j,i)_{tellPub} \approx Pr[Pub]_i \times Pr[u2m] \times C_f + Pr[u2p]_i \times Pr[M2M] \times C_f \tag{3.30}$$

$$N(j,i)_{tellSub} \approx Pr[Sub]_i \times Pr[u2m] \times C_f + Pr[u2s]_i \times Pr[M2M] \times C_f \tag{3.31}$$

(where $Pr[Pub]_i$ or $Pr[Sub]_i$ is the steady-state probability of a cell being published or subscribed to by *federate i* discussed in Subsection 3.4.3.)

By transmitting these data, all of the *publishers* and *subscribers* of a target cell are triggered to join the MGRP in the current *timestep*.

3.5.2.4 Evaluating TellLeave Data

$N(j,i)_{tellLeave}$ considers the number of cells owned by *federate j* whose states transit from $\{u, v, 1\}$ to $\{u', v', 0\}$, and who have the same *publisher* or *subscriber* (*federate i*) remaining in the current *timestep*. By transmitting these data, all remaining *publishers* or *subscribers* of a target cell in the current *timestep* are triggered to leave the MGRP of the target cell.

$$N(j,i)_{tellLeave} \approx (Pr[Pub]_i + Pr[Sub]_i) \times Pr[m2u] \times C_f \tag{3.32}$$

3.5.2.5 Evaluating RefreshCList Data

$N(j,i)_{refCList}$ considers the number of cells owned by *federate* j that have the same *publisher* or *subscriber* (*federate* i) remaining since the previous *timestep,* whose states transit from $\{u, v, 1\}$ to $\{u', v', 1\}$, and that have new arrival or departure activities. By transmitting these data, all of the remaining *publishers* and/or *subscribers* of a target cell are notified about the new activities occurring in the current *timestep.*

$$N(j,i)_{refCList} \approx (Pr[Pub]_i + Pr[Sub]_i) \times Pr[M2M] \times C_f \tag{3.33}$$

In this section, we have evaluated the average number of DDM messages that needs to be transmitted to a destination *federate* in each *timestep* at a source *federate.* In order to compute the result using these approximation formulas, we need to know the state-transition probabilities of a grid cell. In the next section, we will discuss how these probabilities can be solved by applying a *switching model* to the grid-based DDM system.

3.5.2.6 Evaluating Transmission Performance

The objective of this part of work is to design an efficient and scalable transmission control scheme for a grid-based DDM system that is adaptive to different simulation sizes and best approaches optimum performance.

Traditionally, DDM time and DDM messages are two out of three parameters used to evaluate the performance of a DDM approach. Since our existing DDM implementations use the SMT (Single Message Transmission) mode, the DDM time is dominated mainly by the total transmission time at each node. In addition, because the total number of

DDM messages increases tremendously with the input size of a simulation, the DDM time increases as well quickly with the input size of a simulation either. Therefore, in our proposed new adaptive transmission control scheme, the objective for DDM time is very simple:

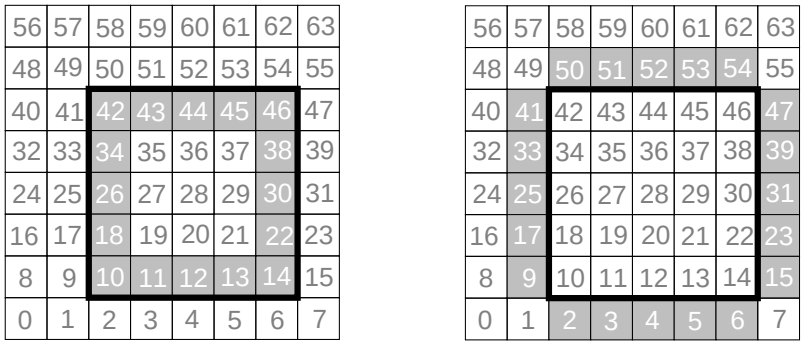
"DDM time should be controlled by the propagation delay instead of the transmission time of DDM messages."

If this objective is fulfilled in our proposed scheme, the DDM time will not rise considerably with the input size of a simulation, and will be the shortest time among all existing DDM implementations. According to our objective for the DDM time, DDM messages generated by a grid-based DDM approach will not be transmitted using the SMT mode; instead, DDM messages destined for the same node will be grouped in one buffer. In order to maximize the performance gain in terms of DDM time, our objective for the DDM messages is to

"Optimize the buffer size for DDM messages based on the input size of a simulation to obtain the best DDM time performance."

3.5.3 Switching Model

The *switching model* is established based on the simulation scenario described in Section 3.3. It consists of state-transition probabilities at the *object* level, the *federate* level, and the *federation* level. We have mentioned previously that these probabilities are the measurement of the dynamic performance of a DDM approach and are used to determine the average number of DDM messages generated in a simulation. In this section, we will evaluate these probabilities in the grid-based DDM environment with the uniform distribution patterns given in Section 3.3. Because of the symmetric behaviors of the publishing and subscribing activities of an *object*, we focus only on the state-transition probabilities of publishing.



(a) Cells that may be unpublished in next timestep

(b) Cells that may be published in next timestep

Figure 3.2: Sate-transition Probabilities at Single Object Level

3.5.3.1 State-transition Probabilities at Object Level

At the *object* level, we are interested in the state-transition probabilities of a single *object*. Figure 3.2 shows the region actually published by an *object* with 4×4 publication region size in the current *timestep*. Because the distance of movement in each *timestep* is set to the *unit distance*, only the grey cells shown in Figure 3.2 (a) may have the chance to switch from the state $(1, \#)$ to $(0, \#)$. In particular, cell 10, 14, 42, or 46 may have such a state change if two out of four directions of movement are chosen by the *object* in the next *timestep,* while other grey cells ($\{11, 12, 13\}$, $\{43, 44, 45\}$, $\{18, 26, 34\}$, or $\{22, 30, 38\}$) may have such state changes if only one out of four directions is chosen by the *object*. Therefore, given a cell being published by an *object*, if the *routing space* grid is large enough, the probability (p_{10}) of this cell's state transition from $(1, \#)$ to $(0, \#)$ is

$$p_{10} = [\frac{4}{(s+1)^2} \times \frac{2}{4} + \frac{4(s-1)}{(s+1)^2} \times \frac{1}{4}] = \frac{1}{s+1}$$

(where $s \times s$ is the publication region size of an *object* and $(s+1) \times (s+1)$ is the maximum publication region size published by this *object*.)

Correspondingly, only the grey cells shown in Figure 3.2 (b) may have the chance to switch from the state $(0, \#)$ to $(1, \#)$. Therefore, given a cell not being published by an

object, if the *routing space* grid is large enough, the probability (p_{01}) of this cell's state transition from $(0, \#)$ to $(1, \#)$ is

$$p_{01} = \frac{4(s+1)}{c^2} \times \frac{1}{4} = \frac{s+1}{c^2}$$

(where $c \times c$ is the *routing space* grid size.)

Let p be the steady-state probability of a cell being published by an *object*. According to Subsection3.4.3, it is given directly as

$$p = (\frac{s+1}{c})^2$$

therefore, we have the expressions for state-transition probabilities at the *object* level as follows

$$up = p_{01} \cdot (1 - p) \tag{3.34}$$

$$pu = p_{10} \cdot p \tag{3.35}$$

$$uu = (1 - p_{01}) \cdot (1 - p) \tag{3.36}$$

$$pp = (1 - p_{10}) \cdot p \tag{3.37}$$

Normally, *routing space* has a limited grid size; as a result, part of the region actually published or subscribed to by an *object* will sometimes fall outside the range of *routing space*. Thus, we lose part of the publication or subscription region. In addition, if we consider the grey cells shown in Figure 3.2 (a) and (b) to be the switching region, then we have the switching region loss of publication or subscription. Let r_p and r_s be the Average Publication Rate (APR) and the Average Subscription Rate (ASR) because of the publication or subscription region loss, and r_{ps} and r_{ss} be APSR (Average Publication Switching Rate) and ASSR (Average Subscription Switching Rate) because of the switch-

ing region loss of publication or subscription. We thus have the modified approximation expressions for the probability p and the conditional probabilities of p_{01} and p_{10}:

$$p \approx r_p \cdot (\frac{s+1}{c})^2 \tag{3.38}$$

$$p_{01} \approx r_{ps} \cdot \frac{s+1}{c^2} \tag{3.39}$$

$$p_{10} \approx \frac{r_{ps}}{r_p} \cdot \frac{1}{s+1} \tag{3.40}$$

In the expressions above,

$$r_p \approx (1 - \frac{s+1}{4c})^2 \tag{3.41}$$

$$r_{ps} \approx 1 - \frac{2(s+2)(3s-2)c - (s+2)^2(s-2)}{8sc^2} \tag{3.42}$$

where $s \in [2,\, c-3]$

The derivations of these two approximations are outside of our focus. We will use the simulation results to show that these approximations are very accurate.

3.5.3.2 State-transition Probabilities at Federate Level

At the *federate* level, we are interested in the state-transition probabilities of all *objects* (n_i) owned by a single *federate* i. When we evaluate the state-transition probability $(p2u)_i$ at the *federate* level, $(pp)_k$ and $(up)_k$ of any *object* k in the *federate* i are out, because these two state-transition probabilities at the *object* level violate the state-transition of a cell from $[u, \# : u \geq 1]$ to $[0, \#]$. Therefore, $(p2u)_i$ is all of the possible combinations of the state-transition probabilities of $[(pu)_k + (uu)_k]$ of all *objects* in the *federate* i except in the case that $(uu)_k$ is true for all.

$$(p2u)_i = \prod_{k=1}^{n_i}[(pu)_k + (uu)_k] - \prod_{k=1}^{n_i}(uu)_k \tag{3.43}$$

Similarly,

$$(u2p)_i = \prod_{k=1}^{n_i}[(up)_k + (uu)_k] - \prod_{k=1}^{n_i}(uu)_k \tag{3.44}$$

$(u2u)_i$ at the *federate* level is easy to express,

$$(u2u)_i = \prod_{k=1}^{n_i}(uu)_k \tag{3.45}$$

since the sum of total probabilities is 1, so

$$\begin{aligned}(p2p)_i &= 1 - (p2u)_i - (u2p)_i - (u2u)_i \\ &= 1 - \prod_{k=1}^{n_i}[(up)_k + (uu)_k] - \prod_{k=1}^{n_i}[(pu)_k + (uu)_k] + \prod_{k=1}^{n_i}(uu)_k\end{aligned} \tag{3.46}$$

Let $(p2p)'_i$ be the state-transition probability $(p2p)_i$ without any new arrival or departure activities. It refers to the probability of the state-transition from $[u, \# : u \geq 1]$ to $[u, \# : u \geq 1]$. For each *object* in the previous *timestep*, if it was publishing to the cell, it continues publishing to the cell in the current *timestep*; otherwise, it continues not publishing. Therefore, $(p2p)'_i$ is all of the possible combinations of the state-transition probabilities of $[(pp)_k + (uu)_k]$ of all *objects* in the *federate* i except in the case that $(uu)_k$ is true for all.

$$(p2p)'_i = \prod_{k=1}^{n_i}[(pp)_k + (uu)_k] - \prod_{k=1}^{n_i}(uu)_k$$

According to the discussion in subsection 3.5.2.1, $(P2P)_i$ represents part of the proba-

bility $(p2p)_i$ that has new arrival or departure activities in the current *timestep*

$$\begin{aligned}(P2P)_i &= (p2p)_i - (p2p)'_i \\ &= 1 - \prod_{k=1}^{n_i}[(up)_k + (uu)_k] - \prod_{k=1}^{n_i}[(pu)_k + (uu)_k] - \prod_{k=1}^{n_i}[(pp)_k + (uu)_k] + 2 \cdot \prod_{k=1}^{n_i}(uu)_k\end{aligned} \tag{3.47}$$

3.5.3.3 State-transition Probabilities at Federation Level

At the *federation* level, we are interested in the state-transition probabilities of all *federates* in the simulation. The probabilities listed in Table 3.2 can be evaluated by following exactly the same reasoning described in the previous subsection, although these are at the *federation* level. As before, let f be the total number of *federates*, then we have

$$p_{p2u} = \prod_{k=1}^{f}[(p2u)_k + (u2u)_k] - \prod_{k=1}^{f}(u2u)_k \tag{3.48}$$

$$p_{u2p} = \prod_{k=1}^{f}[(u2p)_k + (u2u)_k] - \prod_{k=1}^{f}(u2u)_k \tag{3.49}$$

$$p_{u2u} = \prod_{k=1}^{f}(u2u)_k \tag{3.50}$$

$$p_{p2p} = 1 - \prod_{k=1}^{f}[(u2p)_k + (u2u)_k] - \prod_{k=1}^{f}[(p2u)_k + (u2u)_k] + \prod_{k=1}^{f}(u2u)_k \tag{3.51}$$

$$P_{P2P} = 1 - \prod_{k=1}^{f}[(u2p)_k + (u2u)_k] - \prod_{k=1}^{f}[(p2u)_k + (u2u)_k] - \prod_{k=1}^{f}[(p2p)_k + (u2u)_k] + 2 \cdot \prod_{k=1}^{f}(u2u)_k \tag{3.52}$$

At the *federation* level, $m2u$ in Table 3.2 II describes the state-transition probability of a cell from a *matching* state to not being in *matching* anymore. According to the specifications for *matching*, the *matching* condition is met if and only if there is at least one pair of *publisher-subscriber* from different *federates* that are in the same cell. There are three state-transition situations that will lead $m2u$ to be true: (1) from $\{u, v, 1 : u, v \geq 1\}$ to $\{0, v', 0 : v' \geq 1\}$; (2) from $\{u, v, 1 : u, v \geq 1\}$ to $\{u', 0, 0 : u' \geq 1\}$; and (3) from $\{u, v, 1 : u, v \geq 1\}$ to $\{0, 0, 0\}$. Therefore,

$$m2u = p_{p2u} \cdot p_{s2s} + p_{p2p} \cdot p_{s2u} + p_{p2u} \cdot p_{s2u} \tag{3.53}$$

Similarly, we have

$$u2m = p_{p2p} \cdot p_{u2s} + p_{u2p} \cdot p_{s2s} + p_{u2p} \cdot p_{u2s} \tag{3.54}$$

In subsection 3.5.2.1, we mentioned that $M2M$ is the state-transition probability of a cell from a previous state that was in *matching* to a current state that is continuously in *matching* and has new arrival or departure activities. Therefore,

$$\begin{aligned} M2M = {} & P_{P2P} \cdot P_{S2S} + (p_{p2p} - P_{P2P}) \cdot P_{S2S} \\ & + P_{P2P} \cdot (p_{s2s} - P_{S2S}) \end{aligned} \tag{3.55}$$

All of the other probabilities listed in Table 3.2 II do not need to be discussed here, because our interest is only in the state-transitions that are able to generate DDM messages.

3.5.3.4 Lower Boundary Analysis

According to the uniform distribution pattern applied to the simulation scenario (Section 3.3), the inputs of a simulation include the following: (1) *routing space* grid size ($C =$

$c \times c$); (2) total number of *federate*s (f); (3) number of *object*s owned by each *federate* (n_i s.t. $n = \sum_{i=1}^{f} n_i$); (4) publication region size of each *object* ($S_{P,j} = s_{p,j} \times s_{p,j}$); and (5) subscription region size of each *object* ($S_{S,j} = s_{s,j} \times s_{s,j}$). However, the design of input cases can become overly complex if we give each *object* an individual publication and subscription region size; in fact, this is not only unnecessary but also inefficient for computing.

In order to obtain a reasonable and meaningful simulation result, we let all *federate*s have the same number of *object*s ($n_i = n/f$), and all *object*s have the same publication region size ($S_P = s_p \times s_p$) and the same subscription region size ($S_S = s_s \times s_s$). We can prove the fact that

Fact 3.5.3. *When $c >> s_p$ and s_s[10], by applying the inputs (C, f, n/f, S_P, S_S) to a distributed simulation in the grid-based DDM environment, the performance of the simulation represents the lower boundary results of a series of simulation input cases in which the total number of grid cells is C, the total number of federates is f, the average number of objects in each federate is n/f, and the average publication and subscription region sizes of each object are S_P and S_S.*

Proof. By applying Jensen's Inequality Theory and General Means Inequality to probability equations at the *object*, *federate* and *federation* levels respectively, Fact 3.5.3 is correct at each level. □

[10]In practice, we set s_p, $s_s \leq c/8$

3.6 Experimental Results

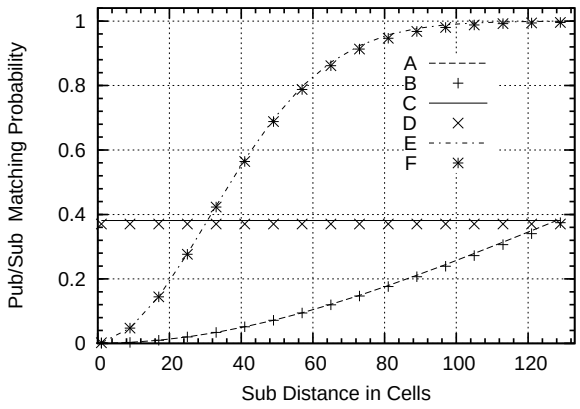
Our simulation is running on a cluster with 32 nodes, and follows the exact scenario we described in Subsection 3.3.1. All existing grid-based approaches have been implemented inside the RTI-Kit, which was originally developed by Georgia Institute of Technology [67]. These approaches include the Fixed Grid-based approach, the Dynamic Grid-based approach, and the Sender-based DDM approach. Because they all use the same fixed storage allocation strategy for dealing with any simulation inputs, we use grid-based approaches to refer to all of them together.

Although the implementation of the AGB DDM approach is based on the RTI-Kit, it is implemented as a separate version because the change of storage allocation strategy makes it incompatible with all existing approaches. The new version has separate schemes for the storage allocation, local processing flow control, and transmission control. In the transmission control, in addition to the existing SMT (Single Message Transmission) mode, we add AMGT (Adaptive Message Grouping Transmission) mode and FMGT (Fixed Message Grouping Transmission) mode. In order to ease our analysis, we also designed multiple simulation modes, like the Debug mode, the Lower Boundary Analysis mode, and the Group Case Testing mode.

3.6.1 Performance of ARAC Scheme

3.6.1.1 Simulation Environment

The details of the simulation setting are that (i) *routing space* is a $1024 \times 1024 \times 2$ grid, the third dimension is used for the team ID[45], and all *object*s play in the first two dimensions; (ii) corresponding to the cluster with 32 nodes, the *federation* consists of 32 *federate*s; (iii) the total number of *object*s is 1024, each *federate* has 32 *object*s; (iv) the publication region size of each *object* is the same, as is the subscription region size; (v) in order to obtain ideal randomized results, the number of total *timestep*s in each run



A: Expected Avg. Subscribing Probability; B: Avg. Subscribing Probability;
C: Expected Avg. Publishing Probability; D: Avg. Publishing Probability;
E: Expected Avg. Matching Probability; F: Avg. Matching Probability;

Figure 3.3: Performance Evaluation of Matching Model

of a simulation is $20,000$; (vi) the distance of one *timestep* for all *object*s is set to *unit distance*.

The experimental data used in this section is generated from two series of simulation runs. For both series, we increase the input size of each simulation run gradually, which is accomplished by fixing the *object* publication region size while increasing the subscription region size[11] in each series of runs.

3.6.1.2 Performance Evaluation

We use experimental data generated from the first series of runs to evaluate the performance variations of the *matching model.* In this series, the *object* publication region size is fixed and set to 128×128, while the *object* subscription region size varies from 1×1 to 129×129 with an eight *unit distance* increase from the first to the last run. In Figure 3.3, the average subscribing, publishing, and *matching* probabilities are simulation results that are computed using experimental data collected from each *federate* in each run. These results are drawn using three distinct types of points (shown as $B, D,$ *and* F in Figure 3.3); in addition, for the purpose of comparison, the corresponding expected

[11]Subscription region size = Sub Distance $\times$ Sub Distance

average values of these probabilities are drawn by three distinct type of lines (shown as $A, C,$ *and* E in Figure 3.3), and they are the theoretical results that are computed using lower boundary inequalities 3.17 and 3.19.

As shown in Figure 3.3, the horizontal line represents the publishing performance of a *federate*, which is horizontal because the *object's* publication region size is fixed. The lower curve represents the subscribing performance of a *federate*, which rises slowly with the increase in the *object's* subscription region size. Finally, the upper curve represents the *matching* performance, which rises quickly before approaching 1 because the *matching* probability is the accumulated effect of the publishing and subscribing probabilities *federation*-wide.

As we can see from this figure, all experimental results (plotted by points) match their theoretical results (plotted by lines) very well. In fact, there is about a 1 percent difference between them, but the tolerance of the figure is greater than that.

3.6.1.3 Storage Allocation Rate

The *storage allocation rate* is the rate of the total storage allocated over the total storage required. It can be expressed as

$$r_{STG} = \frac{STG_{alloc}}{STG_{req}}$$

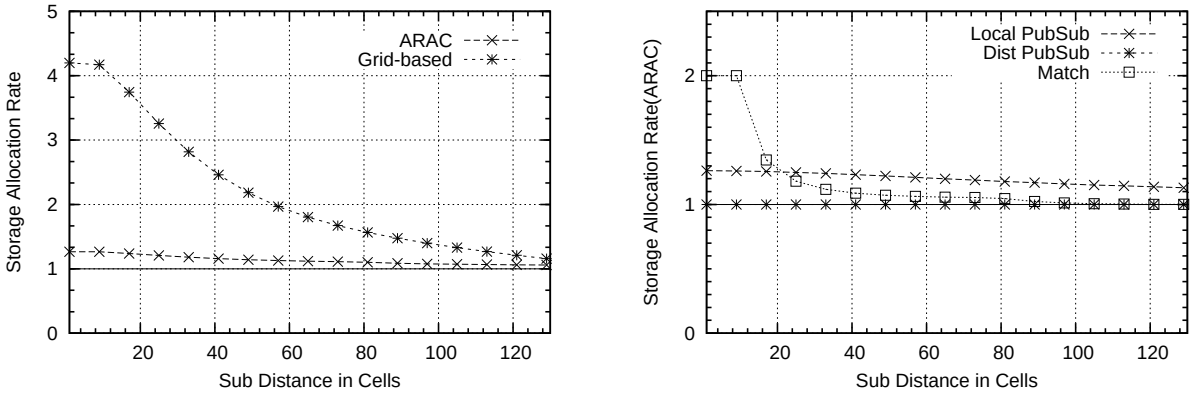
Accordingly, the *storage allocation rate* for the *local Pub/Sub data,* the *distributed Pub/Sub data,* and the *matching data* can be expressed as

$$r_{local\ PubSub} = \frac{STG'_{local\ PubSub}}{STG_{loca\ PubSub}}$$

$$r_{dist\ PubSub} = \frac{STG'_{dist\ PubSub}}{STG_{dist\ PubSub}}$$

$$r_{match} = \frac{STG'_{match}}{STG_{match}}$$

Since the storage required (STG_{req}) is computed using the results from the *Lower Boundary Analysis,* the actual storage (STG_{alloc}) allocated must always be a little larger



(a) Comparison of Storage Allocation Rates (b) Inside Storage Allocation Rate of ARAC

Figure 3.4: Comparison of Storage Allocation Rates

than that or the simulation will crash. The same condition has to be true even for the storage allocated for each part of the *static data*. Therefore, the *bottom line* for allocating storage in the distributed simulation is

$$r_{STG} \geq 1 \; and \; r_{local\,PubSub} \geq 1 \; and \; r_{dist\,PubSub} \geq 1 \; and \; r_{match} \geq 1 \tag{3.56}$$

3.6.1.4 Performance Comparisons

In this part of simulations, the *storage allocation rate* is used to compare the performance of two storage allocation strategies employed by the ARAC scheme and grid-based approaches. As we can see, there is a horizontal line with the *storage allocation rate* of 1 in all of these figures. This line is called the *bottom line*, and it is used to show an ideal solution in which the storage allocated is the storage required.

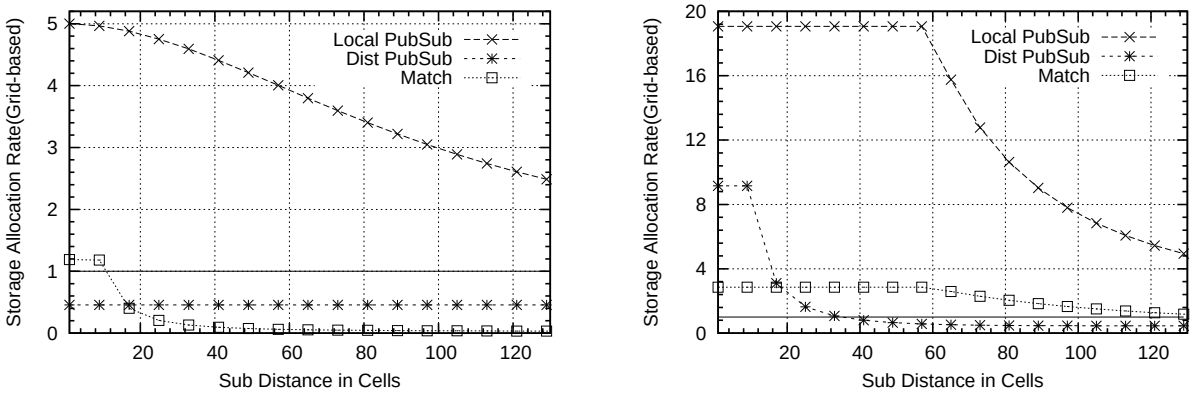
In Figure 3.4 (a), we use the same experimental data from the first series of runs, but instead for the purpose of comparing the *storage allocation rates* of two different storage allocation strategies. As shown in this figure, the *bottom line* is drawn using a solid line, and the performance lines of both schemes start approaching the *bottom line* from

the highest values above it and end with the lowest values almost on the *bottom line*. The reason for this phenomenon can be explained using Equations 3.24, 3.25, and 3.26: (1) when the simulation size is very small, the minimum amount of storage required for starting up is larger than the actual storage required, so that the *storage allocation rate* is highest at the beginning; (2) when the simulation size becomes large, the minimum amount of storage needed for starting up is very small compared to the actual storage required, and thus, the *storage allocation rate* tends to decrease toward 1.

In Figure 3.4 (a), although we do not know exactly where the performance line of the optimum solution lies, there is no doubt that it is located in the area between the *bottom line* and the performance line of the ARAC scheme. Because the *storage allocation rates* of the ARAC scheme are much closer to the *bottom line* than all existing grid-based approaches, the optimum solution has been best approached by the ARAC scheme.

Figure 3.4 (a) is very good at illustrating the outstanding performance of the new ARAC scheme, but it is not good enough because it is misleading. By viewing this figure directly, anyone might think that existing grid-based approaches are indeed not as efficient as the ARAC scheme but are nonetheless still able to run. In actuality, the condition of existing grid-based approaches is much worse. In Figures 3.4 (b) and 3.5 (a), the *allocation rates* for three parts of the *static data* are presented separately. In the first figure, we can see that all three performance lines of the ARAC scheme are above the *bottom line,* while in the second figure the performance line for the *distributed Pub/Sub data* of grid-based approaches remains below the *bottom line* from 0 to the end. According to condition 3.56 in subsection 3.6.1.3, the existing grid-based DDM approaches crash in all runs of the first series of simulations.

In fact, the existing grid-based DDM approaches are able to run only in very limited input cases. Figure 3.5 (b) demonstrates some of these cases. The experimental data used in this figure is collected from the second series of simulation runs, in which the *object* publication region size is still fixed but set to 1×1 instead of 128×128, and the *object* subscription region size varies from 1×1 to 129×129. If we compare this series



(a) Pub Distance = 128 cells

(b) Pub Distance = 1 cells

Figure 3.5: Inside Storage Allocation Rate of Grid-based Schemes

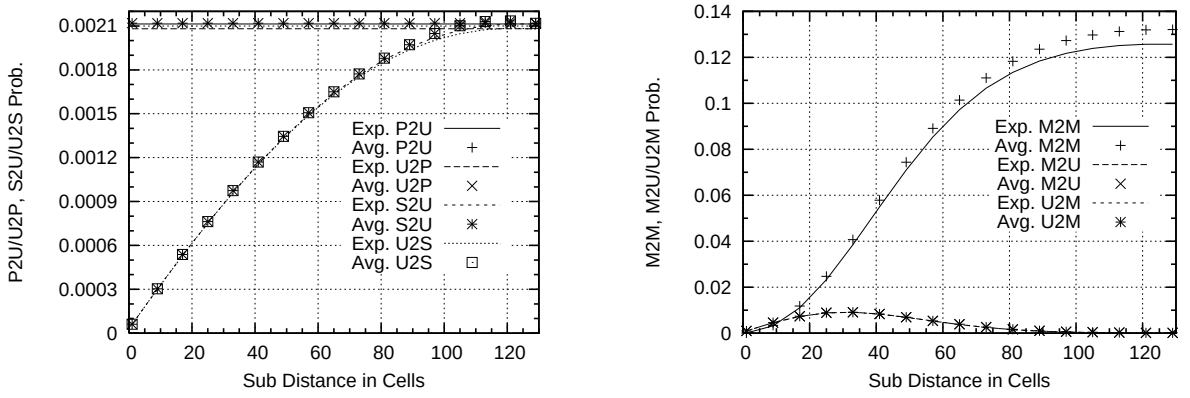
of runs with the first series, the size of the simulation in this series is very limited and the performance of the *matching model* is considerably lower.

3.6.2 Performance of ATC Scheme

3.6.2.1 Simulation Environment

We designed two series of average simulation inputs (C, f, n/f, S_P, S_S) for the performance evaluation. The simulation runs for each input case. In both series, the grid-based *routing space* is set to $C = 1024 \times 1024$, the total number of *federate*s is fixed to $f = 32$, and the total number of *object*s is $n = 1024$. In the first series of inputs, we fix the publication region size to $S_P = 128 \times 128$, but vary the subscription region size from $S_S = 1 \times 1$ to 129×129. In Figures 3.6 and 3.7, we use Pub Distance/Sub Distance or Pub/Sub to represent the publication/subscription region size in each dimension. The first series of inputs is used to validate the correctness of the *switching model*, as well as to compare the simulation performance between the AMGT mode (as shown in Figure 3.7 (a) and (b)), which is employed by our new ATC scheme, and the FMGT mode (as shown in Figure 3.7 (a) and (b)).

In the second series of inputs, we fix the publication region size to $S_P = 1 \times 1$, but



(a) Federate Level (b) Federation Level

Figure 3.6: Performance Evaluation of Switching Model

vary the subscription region size from $S_S = 1 \times 1$ to 65×65. Compared to the first series of inputs, the maximum input case in the second series is about half the size of the minimum input case in the first series. The second series of inputs is used only for the SMT mode (as shown in Figure 3.7 (a) and (b)), which has been adopted in our previous DDM implementations. The purpose of designing the second series of inputs is to generate some simulation results for the SMT mode, because this is the right range in which this transmission mode works. In both series, the total number of *timesteps* in each run is 20000. The distance of each *timestep* is set to the *unit distance*.

3.6.2.2 Performance Evaluation

Figure 3.6 (a) and (b) demonstrate our performance concerns at the *federate* and *federation* levels respectively. In both figures, we use the expected performance and the average performance to validate the correctness of our *switching model*. The expected performances ($Exp.\ P2U$, etc.), which are computed using the *switching model,* are drawn using lines in different styles, while the average performances ($Avg.\ P2U$, etc) collected from the simulation results are plotted by dots in different styles. The state-transition probabilities are represented by using their capitalized forms to emphasize the fact that we are considering the state-transition performance of a cell with new arrival and/or

departure activities. From the direct observation of these two figures, we can see that the expected performance matches the simulation results very well, and thus that the *switching model* correctly models a distributed simulation in the grid-based DDM environment.

3.6.2.3 Performance Comparisons

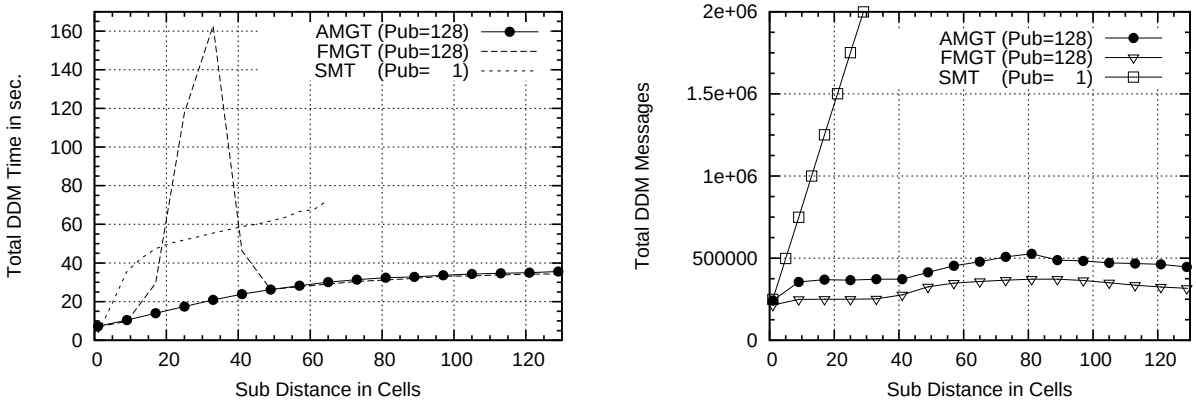
In Figures 3.7 (a) and (b), we use the experimental results from the first series of input cases to compare the performance of DDM time and DDM messages between the AMGT mode and the FMGT mode. The experimental results of the SMT mode are from the second series of input cases, but they are shown together with the AMGT and FMGT in these two figures. Thus, when we compare the performance of these three transmission modes, we should keep in mind that the the maximum input of the second series of simulation runs is about half the size of the minimum input of the first series.

In Figure 3.7 (a), the spike problem generated by the FMGT mode is fixed by applying the AMGT mode. The reason for this is that, in the AMGT mode, DDM messages are grouped adaptively according to the estimation of average DDM messages generated in each *timestep*, and this mechanism ensures that the propagation delay remains being the dominant part of the DDM time in all input cases. Therefore, the total DDM time does not appear to increase as much even as the simulation size increases considerably.

In addition, for the same reason, the DDM time performance of the SMT mode at the maximum input case ($caseA$) is over 70 seconds, while it takes no more than 8 seconds for the AMGT mode in the minimum input case ($caseB$)[12]. In the SMT mode, because the DDM messages are transmitted individually, the total local transmission time rises proportionally with the increase in the simulation size, and this makes the transmission time become the dominant part of the DDM time.

Figure 3.7 (b) presents another viewpoint on the simulation performance. It shows

[12]The input size of $caseA$ ($Pub = 1$ and $Sub = 65$) is only about half of the input size of $caseB$ ($Pub = 128$ and $Sub = 1$).



(a) Comparison of DDM Time (b) Comparison of DDM Messages

Figure 3.7: Performance Comparisons

that the total number of DDM messages for the SMT mode goes up very quickly as the input size increases in each run. As a matter of fact, the number of DDM messages is not as meaningful when the AMGT mode or the FMGT mode is employed, because we manipulate the buffer size of the DDM messages in these two transmission modes to obtain a performance gain in DDM time. Therefore, the DDM messages generated at a *federate* are no longer equivalent to the messages transmitted by the same *federate*.

3.7 Summary

The goal of proposing a novel AGB DDM approach is to control a distributed simulation running in the most efficient mode. In order to achieve this goal, we have developed two adaptive control schemes: the ARAC scheme and ATC scheme. In the ARAC scheme, we optimized the storage allocation strategy for the grid-based DDM system and made it adaptable to various sizes of simulations. We summarize our conclusions for this part of the work as follows: (1) the *matching model* measures the static-state probability of a grid cell in each *timestep* of a simulation; (2) based on the *matching model*, the ARAC scheme is able to evaluate the *static data* generated in each *timestep* of a simulation; (3) *static data* determines the local storage allocated by a *federate*; (4)

Lower Boundary Analysis simplifies the computation work in the ARAC scheme and makes the analysis of the experimental data feasible; (5) the experimental results not only show that the ARAC scheme in our new Adaptive Grid-based DDM approach is adaptable to various simulation sizes, but also indicate that existing grid-based DDM approaches can be applied only to simulations with a very limited size; and lastly, (6) the experimental results show that the optimum solution for the storage allocation has thus been approached best by the ARAC scheme.

On the other hand, the ATC scheme optimizes the transmission control in a simulation according to the input size. We summarize our conclusions as follows: (1) the *switching model* measures the state-transition probability of a grid cell between each two consecutive *timesteps* of a simulation; (2) based on the *switching model*, the ATC scheme is able to evaluate *dynamic data* generated in each *timesteps* of a simulation; (3) *dynamic data* determines the number of DDM messages generated by a *federate* in a grid-based distributed environment; (4) DDM messages generated in a distributed simulation should be grouped before transmission, and the buffer size for grouping should be set adaptively according to the input size; (5) given the distribution pattern of a simulation, the total amount of *dynamic data* is predictable; (6) *lower boundary analysis* simplifies the computation work in the ATC scheme and makes the analysis of the experimental data feasible; (7) experimental results show that the *switching model* correctly demonstrates the state-transition behavior of a cell in the grid-based DDM system; and (8) in comparison with existing DDM implementations, the experimental results show that the ATC scheme achieves a considerable performance gain in DDM time by applying the AMGT transmission mode.

Current experimental results are obtained by applying a set of uniform distribution patterns to the simulation scenario. In our future work, the *matching model* and *switching model* should be capable of handling a simulation scenario with a set of more generalized distribution patterns.

Chapter 4

Conclusions and Future Works

4.1 Conclusions

Because of the fundamental differences at both the application layer and the underlying network layer, DDM in P2P overlay networks and cluster-based network environments follows very different methodologies. In P2P overlay networks, we addressed DDM using a novel distributed data structure *HD Tree*. *HD Tree* is constructed over a complete tree structure, and it doubles the number of neighbors at non-root nodes at the cost of doubling the total number of links of a tree. *HD Tree* inherently supports all important features of a P2P overlay network, like recursive decomposition, scalability, self-organizing, error-resilience, and dynamic loading balancing. *HD Tree* is an optimal solution for supporting multi-dimensional range queries in the P2P overlay network. It is the first distributed data structure proposed in this area that is able to not only adapt the hierarchical structure into the distributed environment, but also to give a complete view of system states when processing multi-dimensional range queries at different levels of selectivity and in various error-prone routing environments. In order to build a complete *HD Tree* overlay network, we proposed the error-resilient *DROCR* algorithm to support

routing in *HD Tree*, Join and Leave algorithms to manage nodes' dynamic feature, *H2H* and *H2D* local adaptation algorithm to balance overly loaded nodes among neighbors in a static manner, and *D2H* repartitioning algorithm to redistribute load system-wide in a dynamic manner. Both theoretical analysis and experimental results indicate that *HD Tree* achieves better routing performance than its equivalent tree. The performance of all basic operations is bound by $O(lg(n))$ in an ideal routing environment, and it survives with a maximum of 10 percent of routing nodes' failures with very limited performance variations in an error-prone routing environment.

On the other hand, in cluster-based network environments, we optimized DDM performance by employing probability models to evaluate the size of a simulation. In particular, we addressed DDM using a AGB DDM approach in the HLA/RTI simulation middle-ware. AGB DDM approach is a novel adaptive grid-based DDM protocol that improves DDM performance by optimizing resource allocation strategy and transmission control strategy for the grid-based DDM systems. AGB DDM approach is the first DDM approach in the HLA/RTI simulation middle-ware that is able to control and adapt a simulation running in the most efficient mode by evaluating the input size of a simulation. We proposed two adaptive control schemes for the AGB DDM approach. The ARAC scheme is used to evaluate and control DDM data that is generated and allocated locally at each cluster node, and it is based on a *matching model* which is a probability model built to evaluate the steady-state probability of a grid cell in a simulation. The experimental results show that the AGB DDM approach is adaptable to various simulation input sizes and the optimum solution for the resource allocation has thus been approached best by the ARAC scheme. For the comparison, the experimental results indicate that existing grid-based DDM implementations can be applied only to *simulations* with a very limited input size. Moreover, the ATC scheme is used to evaluate and control DDM data that should be distributed remotely to other cluster nodes, and it is based on a *switching model* which is another probability model built to evaluate the state-transition probability of a grid cell in a simulation. The experimental results show

that the ATC scheme achieves a considerable performance gain in DDM time over all existing DDM implementations.

In summary, we conclude that *HD Tree* is an effort made towards an efficient, reliable, and scalable P2P search network. It provides a DDM solution for more general purposes, and it supports P2P overlay applications at the architectural level and at an Internet scale. As a comparison, AGB DDM approach is another effort made towards an efficient DDM service in HLA/RTI based simulation systems. It improves the performance of DDM service for HLA/RTI, and it supports large-scale, real-time, distributed, and interactive simulation systems at the implementation level and in cluster-based network environments.

4.2 Future Works

Both parts of the thesis work are conducted for the unique purpose of providing a data indexing service for various large-scale distributed environments. Our first part of the thesis provides a preliminary overview of the *HD Tree* overlay network. Although our conclusions are supported by both theoretical analysis and experimental results, the experimental part is obtained from simulations and by using an abstract data space. Nevertheless, *HD Tree* presents outstanding properties to support multi-dimensional range queries in the distributed environment. However, we need to fully explore the fault tolerance capacities in *HD Tree*, and, at the same time, we have to apply real application scenarios to the *HD Tree* based overlay networks.

The second part of the thesis explores the potential of an optimum DDM solution by modeling and analyzing the simulation scenario. However, this part of experimental results are obtained by applying a set of uniform distribution patterns to the simulation scenario. As a next step, the *matching model* and *switching model* should be capable of handling the simulation scenario with a set of more generalized distribution patterns.

In our future work, we intend to adapt the *HD Tree* data structure into both wired

and wireless distributed environments. We believe that constructing an *HD Tree* based overlay network across heterogeneous network infrastructures and over multiple administrative domains will serve distributed applications and large-scale distributed simulations for much more general purposes. Our final objective is to support P2P overlay applications, including the distributed simulations involving aggregation of large-scale computing based systems in cluster-based environments, like grid computing or cloud computing systems. Our future work will be conducted further towards this end.

Bibliography

[1] Keong Lua, J Crowcroft, M Pias, R Sharma, & S Lim, *A Survey and Comparison of Peer-to-Peer Overlay Network Schemes*. Communications Surveys & Tutorials, IEEE (2005), pp. 72-93.

[2] S. Ratnasamy et al., *A Scalable Content Addressable Network*. Proc. ACM SIGCOMM, 2001, Vol. 31, No. 4. (Oct. 2001) pp. 161–172.

[3] 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.

[4] A. Rowstron & P. Druschel, *Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems* . In IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware) (November 2001), pp. 329-350.

[5] B. Y. Zhao et al., *Tapestry: A Resilient Global-Scale Overlay for Service Deployment* . Selected Areas in Communications, IEEE Journal on (IEEE JSAC), Vol. 22, No. 1. (2004), pp. 41-53.

[6] P. Maymounkov & D. Mazieres, *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric* . Proc. IPTPS, Cambridge, MA, USA, Feb. 2002, pp. 53–65.

[7] D. Malkhi, M. Naor, & D. Ratajczak, *Viceroy: A Scalable and Dynamic Emulation of the Butterfly* . Proc. ACM PODC 2002, Monterey, CA, USA, July 2002, pp. 183–92.

[8] http://groups.yahoo.com/group/the gdf/files/, Gnutella development forum. (Last accessed in Jan. 2010)

[9] http://rfcgnutella.sourceforge.net/Proposals/Ultrapeer/, Gnutella ultrapeers. (Last accessed in Jan. 2010)

[10] I. Clarke et al., *Freenet: A Distributed Anonymous Information Storage and Retrieval System* . Lecture Notes in Computer Science, Vol. 2009 (2001), pp. 46-66.

[11] http://developer.berlios.de/projects/gift-fasttrack/. Fasttrack P2P Technology. (Last accessed in Jan. 2010)

[12] http://en.wikipedia.org/wiki/Kazaa, Fasttrack from Wiki. (Last accessed in Jan. 2010)

[13] http://www.kazaa.com/, Kazaa Media Desktop. (Last accessed in Jan. 2010)

[14] http://en.wikipedia.org/wiki/Fasttrack/, Kazaa from Wiki. (Last accessed in Jan. 2010)

[15] http://www.bittorrent.com/, official BitTorrent website. (Last accessed in Jan. 2010)

[16] http://www.bittorrent.org/, official BitTorrent specification. (Last accessed in Jan. 2010)

[17] http: //www.overnet.com/, the Overnet File-sharing Network. (Last accessed in Jan. 2010)

[18] http://en.wikipedia.org/wiki/Overnet/, the Overnet from Wiki. (Last accessed in Jan. 2010)

[19] http://www.edonkey2000.com/, official eDonkey200 website. (Last accessed in Jan. 2010)

[20] http://en.wikipedia.org/wiki/EDonkey_network/. eDonkey2000 from Wiki. (Last accessed in Jan. 2010)

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

[22] Hanan Samet, *The Quadtree and Related Hierarchical Data Structures* . ACM Computing Surveys(CSUR), Vol. 16, Issue 2 (Jun. 1984), pp. 187-260.

[23] Jon Louis Bentley, *Multidimensional Binary Search Trees Used for Associative Searching* . Commun. ACM, Vol. 18, No. 9. (Sep. 1975), pp. 509-517

[24] G.M. Morton, *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing* . Technical report, Ottawa, Canada, IBM Ltd. 1966.

[25] J. A. Orenstein & T. H. Merrett, *A Class of Data Structures for Associative Searching* . In PODS '84: Proc. of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems (1984), pp. 181-190.

[26] H. V. Jagadish, *Linear Clustering of Objects with Multiple Attributes* . In SIGMOD '90: Proc. of the 1990 ACM SIGMOD Int. Conf. on Management of Data (1990), Vol 19, Issue 2, pp. 332-342.

[27] H. V. Jagadish, *Linear Clustering of Objects with Multiple Attributes* . In SIGMOD '90: Proc. of the 1990 ACM SIGMOD Int. Conf. on Management of Data (1990), Vol 19, Issue 2, pp. 332-342.

[28] http://www.itl.nist.gov/div897/sqg/dads/HTML/kdtree.html. (Last accessed in Jan. 2010)

[29] D. R. Karger et al. *Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web*. In ACM Symposium on Theory of Computing (1997), pp. 654-663.

[30] F. Dabek et al., *Towards a Common API for Structured Peer-to-peer Overlays*. Peer-to-Peer Systems II (IPTPS 2003), Berkeley, CA, USA, Feb. 2003, pp. 33-44.

[31] 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.

[32] P. Ganesan et al., *One Torus to Rule Them All: Multi-dimensional Queries in P2P Systems*. In WebDB '04: Proc. of the 7th Int. Workshop on the Web and Databases (2004), pp. 19-24.

[33] 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.

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

[35] C. Schmidt et al., *Flexible Information Discovery in Decentralized Distributed Systems*. Proc. of the 12th IEEE Int. Symposium on High Performance Distributed Computing (HPDC'03) pp 226-235.

[36] T. Schütt et al., *A Structured Overlay for Multi-dimensional Range Queries*. Euro-Par 2007, Parallel Processing (2007), pp. 503–513.

[37] James Aspnes, Gauri Shah, *Skip graphs*. In Proc. SODA, 2003., pp. 384–393.

[38] J. Risson, T. Moors, *Survey of Research towards Robust Peer-to-Peer Networks: Search Methods*. Computer Networks, Vol 50, No 17, (Dec 2006), pp 3485-3521.

[39] N. Harvey, M.B. Jones, S. Saroiu, M. Theimer, A. Wolman, *SkipNet: A Scalable Overlay Network with Practical Locality Properties*. in Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems USITS'03, Mar. 2003, pp 9-23.

[40] K. Aberer, P. Cudre-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, R. Schmidt, *P-Grid: A Self-organizing Structured P2P System.* ACM SIGMOD Record 32 (3) (2003) pp 29–33.

[41] A. Bharambe, M. Agrawal, S. Seshan, *Mercury: Supporting Scalable Multi-attribute Range Queries.* SIGCOMM, Vol 34, 2004, pp 353-366.

[42] M. Cai, M. Frank, J. Chen, P. Szekely, *MAAN: A Multiattribute Addressable Network for Grid Information Services.* in Proceedings of the 4th International Workshop on Grid Computing, Nov. 2003, pp 184-191.

[43] H.V. Jagadish, B. C. Ooi, Q.H. Vu, *BATON: A Balanced Tree Structure for Peer-to-Peer Networks.* Proceedings of the 31st international conference on Very large data bases, 2005, pp 661-672.

[44] H. V. Jagadish, B.C. Ooi, Q.H. Vu, R. Zhang, A.Y. Zhou, *VBI-Tree: A Peer-to-Peer Framework for Supporting Multi-Dimensional Indexing Schemes.* the 22nd IEEE International Conference on Data Engineering (ICDE), 2006, pp 34-44

[45] A. Boukerche and A. Roy. *Dynamic Grid-based Approach to Data Distribution Management.* JPDC, Vol62, Issue3 : 366–392, 2002.

[46] DMSO. *Data Distribution and Management Design Document V. 0.2.* Department of Defense, Washington D.C. U.S., Dec 1996.

[47] DMSO. *High Level Architecture Interface Specification, V. 1.3.* Department of Defense, Washington D.C. U.S., 1998. http://hla.dmso.mil.

[48] J. S. Dahmann and K. L. Morse. *High Level Architecture for Simulation: An Update.* In 3rd IEEE DS-RT, 32-40, 1998.

[49] G. Tan Y. Zhang and R. Ayani. *A Hybrid Approach to Data Distribution Management.* In IEEE (DS-RT), p. 55, 2000.

[50] A. Boukerche N.J. McGraw C. Dzermajko and K. Lu. *Grid-filtered Region-based Data Distribution Management in Large-scale Distributed Simulation Systems.* In 38th ANSS'05 Proceeding 1080-241X/05.

[51] G. Tan L. Xu F. Moradi and Y. Zhang. *An Agent-based DDM Filtering Mechanism.* In 8th MASCOTS, p. 374, 2000.

[52] R. Fujimoto T. Mclean K. Perumalla and I. Tacic. *Design of High Performance RTI Software.* In IEEE DS-RT, 89-96, 2000.

[53] M. Moreau. *Documentation for the RTI.* George Mason Univ., 1997.

[54] K. Morse. *Interest Management in Large Scale Distributed Simulations*. Technical Report, U. of California, TR 96-27, 1996.

[55] M. Macedonia, M. Zyda, D. Pratt, & P. Barham. *Exploiting Reality with Multi-cast Groups: A Network Architecture for Large Scale Virtual Environments*. In "Virtual Reality Annual International Symposium, p. 2, 1995."

[56] H. Abrams K. Watsen and M. Zyda. *Three-tiered Interest Management for Large-scale Virtual Environments*. ACM Symposium on Virtual Reality Software and Technology, 125-129, 1998.

[57] A. Berrached. *Alternative Approaches to Multicast Group Allocation in HLA Data Distribution Management*. In SIW, 98S-SIW-184, March 1998.

[58] A. Boukerche. *Time Management in Parallel Simulation*. In *High Performance Cluster Computing*. Vol l2, pages 375–394. Prentice Hall, Englewood Cliffs, NJ, (B. Rajkumar ED), 1999.

[59] J. O. Calvin and D. J. Van Hook. *Agents: An Architectural Construct to Support Distributed Simulation*. In 11th Workshop on Standards for the Interoperability of DIS, 94-11-142, Sept 1994.

[60] S. J. Rak. *Evaluation of Grid Based Relevance Filtering for Multicast Group Assignment*. 96-14-106, In 14th DIS, 1996.

[61] D. J. Van Hook, S. J. Rak & J. O. Calvin. *Approaches to RTI Implementation of HLA Data Distribution Management Services*. 96-14-084, in "15th DIS, 1996"

[62] F. Weiland. *An Empirical Study of Data Partitioning and Replication in Parallel Simulation*. In "Proceedings of the 5th Distributed Memory Computing Conference" Vol. II, pp. 915-921, 1990

[63] K. Pan, S. J. Turner, W. Cai & Z. Li. *An Efficient Sort-based DDM Matching Algorithm for HLA Applications with a Large Spatial Environment*. In "Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation" pp. 70-82, 2007

[64] R. Minson and G. Theodoropoulos. *Adaptive Interest Management via Push-Pull Algorithms*. In "Proceedings of the 10th IEEE International Symposium on Distributed Simulation and Real-Time Applications" pp. 119-126, 2006

[65] E. S. Liu, M. K. Yip, & G. Yu. *Scalable Interest Management for Multidimensional Routing Space*. In "Proceedings of the ACM on Virtual Reality Software and Technology" pp. 82-85, 2005

[66] A. Boukerche, C. Dzermajko, & K. Lu. *Alternative Approaches to Multicast Group Management in Large-scale Distributed Interactive Simulation Systems.* In Future Generation Comp. Syst. 22(7): 755-763 (2006)

[67] http://www.cc.gatech.edu/computing/pads/fdk.html, FDK - Federated Simulations Development Kit. (Last accessed in Jan. 2012)