The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

RESOURCE AND KNOWLEDGE DISCOVERY IN LARGE SCALE DYNAMIC NETWORKS

A Thesis in

Computer Science and Engineering

by

Mei Li

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2007

The thesis of Mei Li has been reviewed and approved* by the following:

Wang-Chien Lee
Associate Professor of Computer Science and Engineering
Thesis Co-Adviser
Chair of Committee

Anand Sivasubramaniam
Professor of Computer Science and Engineering
Thesis Co-Adviser

Thomas La Porta
Professor of Computer Science and Engineering

Chao-Hsien Chu
Associate Professor of Information Science and Technology

Peng Liu
Associate Professor of Information Science and Technology

Raj Acharya
Professor of Computer Science and Engineering
Department Head

*Signatures are on file in the Graduate School

Abstract

A massive amount of information, including multimedia files, relational data, scientific data, system usage logs, etc., is being collected and stored in a large number of host nodes connected as *large scale dynamic networks (LSDNs)*, such as peer-to-peer (P2P) systems and sensor networks. A wide spectrum of applications, e.g., resource locating, network attack detection, market analysis, and scientific exploration, relies on efficient discovery and retrieval of resources and knowledge from the vast amount of data distributed in the network systems. With the rapid growth in the volume of data and the scale of networks, simply transferring the data generated at different host nodes to a single site for storing and processing becomes impractical, incurring excessive communication overhead while raising privacy concerns. Thus, a major challenge faced by LSDNs is to design decentralized infrastructures and algorithms that enable efficient resource and knowledge discovery in large scale dynamic networks.

In this dissertation, various resource and knowledge discovery tasks ranging from simple tasks such as query processing to complex tasks such as network attack detection are systematically investigated, with a synergy of research efforts spanning multiple disciplines, including distributed computing, network and data management. Efficient and robust infrastructures and algorithms are proposed to support these tasks, with particular attention paid to various system issues including load balancing, maintenance, adaptivity to dynamic changes, data distribution and users access pattern in the networks. The superiority of these proposed ideas is demonstrated through extensive experiments

using both synthetic data and real data. This dissertation provides profound insights on exploiting the vast amount of data for different applications, e.g., system performance tuning, network attack detection, market analysis, opens the new research direction on distributed data mining, and provides a solid foundation for exploring various data management tasks in the networks systems. It is expected that this study will have a deep impact on the deployment of various applications that mandate efficient management and mining of the vast amount of data distributed in the network systems.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I am highly grateful to my advisors Dr. Wang-Chien Lee and Dr. Anand Sivasubramaniam for their valuable advice and guidance throughout my PhD study at Pennsylvania State University.

I am indebted to Dr. Lee for shaping me towards becoming an independent, confident and mature researcher. His endless hours of advice, guidance and encouragement taught me how to identify important research issues, write papers in a precise form, present in public, connect with peer researchers, move on when a paper is rejected, and raise the bar when a paper is accepted. Dr. Lee through his multidisciplinary expertise pointed out the connections between data management and peer-to-peer systems, and steered me towards this research area during the early stage of my dissertation work. He has contributed endless hours of guidance to my PhD study, which greatly improves the quality of my research work and this dissertation study. I am also grateful for his great support and confidence on my remote study in Pittsburgh during the later stage of my PhD work, which allows me to continue my work while being away from campus to live together with my husband and son. Although I am away from campus, he is still my source of advice and guidance through our constant exchanges of emails and phone calls. Above all, I am grateful that Dr. Lee has made my PhD study a fruitful process.

I am grateful to Dr. Sivasubramaniam for pushing me to always conduct research of the highest standard during my PhD study. He taught me how to set up a high standard for myself when I embarked on my PhD study. He constantly raises the standard

bar on research for himself and the students. From him, I learned the valueless lesson on research, i.e., always striving for the research work in the highest quality. In addition, he taught me how to systematically investigate and evaluate a research issue, which is the essential ingredient to perform solid research. His high standard and systematical methodology are and will always be beneficial for me throughout my research career. Dr. Sivasubramaniam has contributed greatly to the earlier stage of this dissertation work, which has shaped and greatly improved the quality of this study.

I am also grateful to Dr. Thomas La Porta in the Department of Computer Science and Engineering, Dr. Chao-Hsien Chu and Dr. Peng Liu in the School of Information Science and Technology for providing constructive feedback and serving on my committee from their busy schedule.

I thank Dr. John Metzner, who supervised me during my Master study. His patient guidance boosted my confidence and his dedication to education and research sparkled my interest in pursuing PhD in Computer Science.

Many other people have helped me during my PhD study. Dr. Dik Lun Lee from Hong Kong University of Science and Technology has provided valuable feedback on the study of SSW, which has shaped and greatly improved the study on SSW. Dr. Guanling Lee from National Dong Hwa University of Taiwan has helped me start the study on PENS through her expertise on data mining and contributed greatly in terms of ideas and presentation in the study of PENS. Our exchange of email messages is of great inspiration. Dr. Jie Lin from Xerox Innovation Group has opened the door for me to see the interesting and exciting works done in industry and challenged me to do research from a more practical perspective. Thanks also go to Jing Zhao from Hong

Kong University of Science and Technology for providing the real data set for the study on SSW and implementing the part of SSW with real data testing.

I am thankful to many great mentors and teachers of mine at Pennsylvania State University, including Dr. Guoray Cai, Dr. Guohong Cao, Dr. Chita Das, Dr. Ali Hurson, Dr. Thomas La Porta, Dr. Wang-Chien Lee, Dr. Furer Martin, and Dr. Anand Sivasubramaniam. Their mentoring and teaching enable me to pursue my PhD study without much hindrance.

Many of my fellow students, including Guiling Wang, Yingqi Xu, Jing Zhao, Ming Shao, Qingzhao Tan, Bing-rong Lin, Huajing Li, and Chi Keung Lee, from Pennsylvania State University offered their help during my PhD study, which makes the process a much more enjoyable experience.

My families and friends are always the source of support, comfort and joy. Although my family back in China endures the long term and long distance separation, their constant support and encouragement make me feel right back at home. My husband is always caring and understanding. He always listens to me patiently and responses with the sound sense and pure honesty that I rely on. Besides that, he has offered tremendous help on my research itself through frequent (heated) discussions, countless constructive comments, and painful editing. Without his support, encouragement, and sacrifice, I would not get this dissertation work done. My son is a god-sent gift, bringing abundant joy, energy, and inspiration to my life, and also helping me grow as a mom besides as a PhD student. My friends keep me sane while I am being a PhD student and a mom, and help me have a good laugh outside of my research, which I always appreciate.

——To my families and friends, I dedicate this dissertation work.——

Chapter 1

Introduction

A massive amount of information, including multimedia files, relational data, scientific data, system usage logs, etc., is being collected and stored in a large number of host nodes organized as peer-to-peer (P2P) systems and sensor networks. Since these networks typically have large scale and are dynamic by nature, in this dissertation, we refer to them as *large scale dynamic networks (LSDNs)*. A wide spectrum of applications, e.g., resource locating, network attack detection, market analysis, and scientific exploration, relies on efficient discovery and retrieval of resources and knowledge from the vast amount of data distributed in the network systems. With the rapid growth in the volume of data and the scale of networks, simply transferring the information generated at different host nodes to a single site for storing and processing becomes impractical, incurring excessive communication overhead while raising privacy concerns. Therefore, a primary challenge faced by these applications is to design decentralized infrastructures and algorithms that enable efficient resource and knowledge discovery in large scale dynamic networks. In this dissertation, we investigate the aforementioned challenge in depth mainly in the context of peer-to-peer systems, a representative LSDN.

1.1 Peer-to-Peer Systems

The peer-to-peer computing model starts to receive a lot of attention since 1999 when Shawn Fanning created Napster, which allows users on the Internet to share audio files (MP3) directly with each other. Napster consists of a centralized lookup directory and decentralized storage system. In Napster, each user stores its music files in its local disk and a centralized directory records the index for all the music files shared in the user community. To retrieve a file, a user first issues a keyword-based search to the central directory to obtain the IP address of the owner user who has the requested file. Then the user directly contacts the owner to download the file. The basic idea of Napster is simple. Nevertheless, within a year, the user community of Napster grew to 50 millions. In March 2001, Napster was shut down because the Recording Industry Association of America (RIAA) won the lawsuit against Napster that it violates the copyright law.

Following Napster, Gnutella [2] and Freenet [1], which are completely decentralized in both lookup and storage, emerged rapidly. The emergence of these systems is mainly motivated by legal reasons rather than technical reasons. Without centralized directory, each peer in the system is indistinguishable in terms of their functionality, which makes the system harder to control and censor. Within several years since their first emergence, P2P file sharing applications have become one of the most dominant applications in the Internet in terms of traffic [5, 6].

The initial popularity gained by P2P file sharing applications is mainly due to the benefits that they offered to end-users, i.e., sharing files freely in the Internet. These

applications raise many interesting legal and social issues [114]. Nevertheless, the peer-to-peer computing model initiated by such applications has its own technical savvy, which has received intensive attention in academia as well as in IT industry. The most desirable feature offered by peer-to-peer computing model is the natural scalability in different types of resources, including computing, storage, and bandwidth. Without relying on a centralized adminstration, the system gains natural scalability since as the size of the system grows, so do the resources in the system. This makes this computing model very attractive for deploying various kinds of low-cost and large scale applications, which would be too costly to be deployed in the conventional client-server computing model. As a matter of fact, in the recent years, we have seen rapid emergence of applications using the peer-to-peer computing model besides file sharing, e.g., distributed file systems [36, 109], distributed events notification [111], application-level multicast [107, 136], and digital library [119].

1.2 Research Issues

The major resources shared in P2P systems is various types of information, including multimedia files, relational objects, scientific data, system usage logs, etc. We denote the information shared in the user community as *user data*, e.g., relational data, scientific data, and multimedia data, and the information generated by the systems to record the system usage as *usage data*, e.g., the number of downloads for each file in a time window, the size of flow between each source-destination IP address pair, etc. User data generally change rather slowly with insertion/deletion performed by users. In the contrast, system usage data are constantly changing and they are streaming data

in nature. For the latter, the system or users are primarily interested in extracting the general patterns, trends, or anomalies for customer/market analysis, system performance tuning, and network attack detection. We call these operations as *knowledge discovery*. For the former, in addition to knowledge discovery, the users and applications may be interested in searching for information that matches the search criteria (or attribute values) specified by the users. We call these operations as *resource discovery*. In the following, we elaborate on both in terms of some specific applications and tasks.

1.2.1 Resource Discovery

Majority of the user data associated with the applications deployed or to be deployed on P2P systems involves data objects with multiple attributes, which can be viewed as points in a multi-dimensional space. For instance, mapping or location services involve data objects with two attributes (longitude and latitude). Data grid involves data objects with multiple attributes including types of operating systems, CPU speed, network address, storage capacity, etc. Relational data objects have multiple attributes, such as product ID, quantity, price, and manufacturer. Similarly, documents can be expressed as multiple-attributes data objects with each keyword or latent concept as an attribute.

Users often issue complex queries in addition to exact match queries (point queries) to retrieve data objects of interest from P2P systems. For instance, a user might issue the following queries: "return all the documents with similarity to x within r" (range query), or "return the k documents most similar to x" (K nearest neighbor query). Thus, one of the fundamental issues faced by P2P systems is to efficiently support complex queries (in

addition to point query) on multi-dimensional data objects. Two most common types of complex queries are range query and K nearest neighbor (KNN) query. Given a reference data object q and a radius r, a range query returns the data objects whose difference from q is less than r. Given a reference data object q and an integer K, KNN query returns the K data objects most similar to q.

1.2.2 Knowledge Discovery

The need for knowledge discovery in LSDNs is prevailing in a wide range of applications. For instance, detecting the most popular files (e.g., the files with the largest number of downloads), or detecting the most popular host nodes (e.g., the host nodes serving the largest number of requests) can help the design of cache mechanisms. As another example, monitoring trends or anomalies in the communication traffic on the network can help pinpoint the performance bottlenecks of the network, facilitate the designers and administrators make strategic decisions on the system design and resource management, ensure the network's proper operation, and even detect on-going network attacks [40, 71, 77].

The nature of different applications and data associated with these applications mandate different treatments for different applications. In this study, we investigate the following specific issues involved with various applications in LSDNs, with the first two in the category of resource discovery and the latter two in the category of knowledge discovery:

- Design of infrastructures to facilitate various types of queries on data objects identified with arbitrary number of attributes.

- Design of algorithms to process various types of queries on data objects with arbitrary number of attributes.
- Design of efficient mechanisms to monitor interesting/abnormal events in LSDNs, which facilitate network attack detection and system performance tuning.
- Design of efficient mechanisms to perform various data mining tasks, which reveal the intrinsic features of data objects to facilitate smart query answering and customer/market analysis.

1.3 Design Challenges

The unique characteristics of LSDNs, such as large scale, lack of centralized administration, and high degree of dynamics, make the design of the techniques to address the aforementioned issues extremely challenging:

- **Scalability:** The techniques should be easily scalable in terms of both the size of host nodes and the volume of information shared in the system.
- **Efficiency:** The techniques should be efficient in terms of delay and communication cost.
- **Adaptivity:** It is the norm instead of exception that LSDNs are very dynamic in terms of the host nodes participating in the system, the information shared in the system, and the data requested by the users. Therefore, the techniques should be adaptive to all these changes without incurring high overheads.

- **Fairness:** The processing load should be fairly distributed amongst the participating host nodes to guarantee the normal performance of the network system and also promote the health of the network.

1.4 Dissertation Overview

This dissertation work consists of five major components: *semantic small world (SSW)* for content-based search, *distributed peer tree (DPTree)* for multi-dimensional complex queries, *in-network filtering (netFilter)* for identifying frequent items, *wavenet* for monitoring significant changes on data distribution, and *peer density based clustering (PENS)* for decentralized clustering. SSW and DPTree are designed for resource discovery; netFilter, wavenet and PENS are designed for knowledge discovery. In the following, we briefly summarize these studies.

- **Semantic Small World.** The massive amount of information shared in P2P systems mandates efficient content based search for resources, e.g., documents, which are often identified and queried using a large number of attributes and thus can be treated as data points in a *high dimensional space*. This study presents the design of an overlay network, namely semantic small world (SSW), that facilitates efficient content based search in P2P systems. SSW achieves the efficiency based on the following four ideas: 1) semantic clustering; 2) dimension reduction; 3) small world network; 4) efficient search algorithms.

- **Distributed Peer Tree.** While semantic small world is specifically designed to support content-based search (i.e., search on high dimensional data objects), it

is not optimal to support data objects with a small or moderate number of attributes. We propose a framework, called distributed peer tree (DPTree), which, based on balanced tree indexes, efficiently supports various types of queries on multi-dimensional data (in low to medium dimensions) in P2P systems with reasonable maintenance overheads. In addition, DPTree adapts to data distribution and access pattern by effectively balancing the access load among peers. DPTree achieves the efficiency and effectiveness through the following designs: 1) distributing the tree structure among peers in a way preserving the nice properties of balanced tree structures yet avoiding single points of failure and performance bottlenecks; 2) organizing peers into an overlay structure that enables efficient navigation yet is easy to maintain; 3) an efficient navigation algorithm; 4) an innovative wavelet-based load balancing mechanism.

- **In-network Filtering.** The need of identifying 'frequent items' in LSDNs appears in a variety of applications, ranging from cache management to network attack detection. We define the problem of *identifying frequent items (IFI)* and propose an efficient in-network processing technique, called netFilter, to address this important fundamental problem.

- **Wavenet.** In this study, we investigate *monitoring changes on the data distribution in the network (MCDN)*, which is a prevailing task in network attack detection, distributed query optimization, and various distributed data mining operations. To address this problem, we propose wavenet. The basic idea is to compress the

local item set of each host node into a compact yet accurate summary, called *localwavelet*, for communication with the coordinator to facilitate MCDN.

- **Peer Density-based Clustering.** We propose a fully distributed clustering algorithm, called peer density-based clustering (PENS), which overcomes various challenges raised in performing clustering in peer-to-peer environments, including cluster assembly and cluster membership storage. Additionally, an important feature of our algorithm is incremental clustering, which is essential to efficiently perform clustering on a dynamic data set (with constant insertion or deletion).

To the best of our knowledge, our studies on netFilter, wavenet and PENS are among the first studies on knowledge discovery in LSDNs. There are some studies that explore resource discovery in LSDNs, e.g., [27, 59, 122]. Some of these studies are designed specifically to support a specific type of query, and thus are not adequate to support other types of queries; others only focus on search performance and ignore other important system issues in LSDNs, such as load balance, maintenance, and adaptivity to dynamic changes. In contrast to these studies, our studies on resource discovery provide a suit of solutions that efficiently support different types of queries while systematically taking into considering the aforementioned system issues.

In the next chapter, we compare carefully with the related studies. In the following, we first summarize the major contributions made by this dissertation.

1.5 Contributions

This dissertation work constitutes the following four major contributions.

1. **Providing a solid foundation for exploring various data management tasks in LSDNs.** We have designed a variety of data management infrastructures, including semantic small world (SSW) [84, 87, 88] and distributed peer tree (DPTree) [85, 86]. These infrastructures are designed through a systematic approach, and the best tradeoff among various system metrics, including scalability, ease of maintenance, adaptivity to dynamic changes, and load balancing, is considered. These infrastructures provide a solid foundation for exploring different data management tasks in LSDNs.

2. **Paving the way to investigate different complex queries in LSDNs.** We have designed a suite of algorithms to support efficient processing of complex queries in LSDNs [81]. These algorithms intelligently retrieve the relevant information requested by the users from a plethora of information hosted in LSDNs. They are efficient in terms of both the retrieval speed and the incurred communication traffic. The design of these algorithms paves the way to process other complex queries in LSDNs.

3. **Providing profound insights on exploiting the vast amount of data for different applications.** We have investigated monitoring interesting/abnormal events in LSDNs for network attack detection and system performance tuning. We have designed distributed mechanisms, i.e., in-network filtering (netFilter) [80] and wavenet [89], that efficiently identify frequent items and changes on the data distribution, respectively, in LSDNs. The design of these mechanisms for network monitoring provides profound insights on exploiting the vast amount of data for

different applications, e.g., system performance tuning, network attack detection, market analysis, etc.

4. **Opening the new research direction on distributed data mining in LSDNs.** We have designed a fully distributed clustering algorithm (peer density-based clustering (PENS) [79]) that efficiently clusters dynamic data objects distributed in P2P systems. This study opens the door to an important new research direction on distributed data mining in LSDNs.

1.6 Roadmap

The rest of this dissertation is organized as follows. We first review the related studies in Chapter 2. The five major studies composing this dissertation work, i.e., SSW, DPTree, netFilter, wavenet, and PENS, are presented in Chapter 3 to Chapter 7, respectively. We summarize this study, discuss some open issues and outline the future works in Chapter 8.

Chapter 2

Related Works

Since many techniques proposed in the literature within the context of LSDNs reminisce the relevant techniques proposed in centralized systems, we first briefly summarize the relevant techniques proposed to address resource discovery and knowledge discovery in centralized systems. We then review some relevant studies on distributed resource discovery and knowledge discovery.

2.1 Resource Discovery and Knowledge Discovery in Centralized Systems

We first summarize the studies in resource discovery, followed by the centralized clustering techniques.

2.1.1 Resource Discovery in Centralized Systems

A surge of studies have been done to index large data sets to facilitate efficient access of these data sets (please see [47] for a in-depth survey). The representative index structures proposed in the literatures are B+tree [22], Quad-tree [44], k-d-tree [23], R-tree [54]. Many variants of these index structures, e.g., [24, 25, 67, 90, 126, 130], have been proposed to index high dimensional data objects.

Among these studies, B-tree is a balanced tree structure indexing data objects with a single attribute, i.e., one-dimensional data objects. Quad-tree, grid file, k-d-tree,

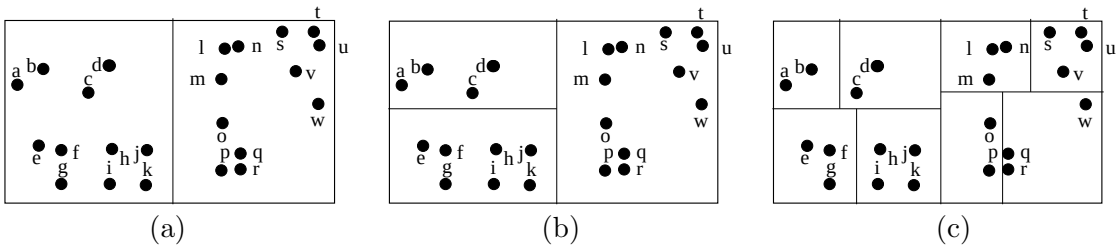


Fig. 2.1. An illustrative example of k-d-tree.

and R-tree are designed for data objects with multiple attributes, i.e., multi-dimensional data objects. We can broadly classify these multi-dimensional index structures into two classes: *space partition based indexes* (Quad-tree, grid file, k-d-tree) and *object grouping based indexes* (R-tree). Space partition based indexes recursively partition the whole data space (defined by the domain range of data attributes) into smaller sub-regions. Object grouping based indexes recursively group data objects nearby into hierarchical structures. Figure 2.1 illustrates an example for k-d-tree, a representative space partition based index. Figure 2.1(a–c) represent the two subregions, four subregions, and eight subregions after the first partition, second partition and final (seventh) partition, respectively. Figure 2.2 illustrates an example for R-tree, a representative object grouping based index. The left side depicts the object grouping and the right side depicts the hierarchical structure.

These two classes of indexes mainly differ in the following three aspects:

- Space partition based indexes index the whole data space, including the *dead space*, i.e., the space that is not populated by data objects. In contrast, object grouping

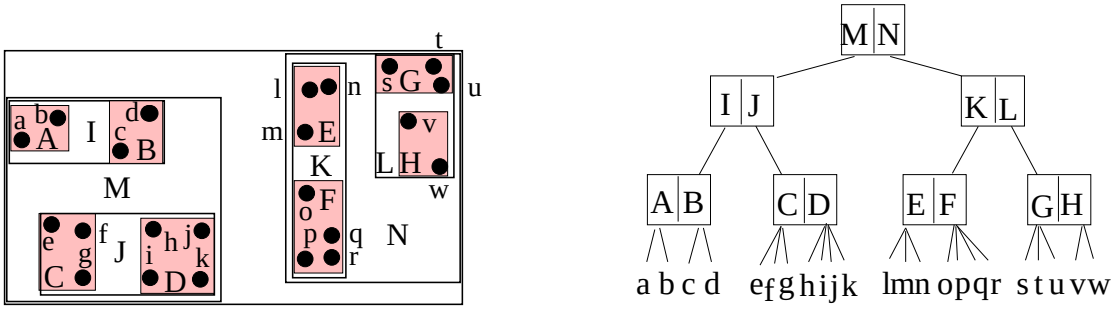


Fig. 2.2. An illustrative example of R-tree.

based indexes mainly index the space populated by data objects. This difference becomes more obvious when the data objects are not uniformly distributed in the data space.

- Space partition based indexes are very sensitive to the changes on the distribution of data objects in the space. The reason is that once a partition is made, it is very costly to change the partition, especially if the partition is made early on. Object grouping based indexes do not have this restriction and they can adaptively change the object grouping in accordance with the change on the data distribution.

- Object grouping based indexes may have overlapping among groups, which degrades their performance, especially when the dimensionality is high.

2.1.2 Clustering in Centralized Systems

Clustering, which groups a set of data objects into clusters of similar data objects, can be applied in many different problem domains, such as spatial data analysis, scientific pattern discovery, document categorization, taxonomy generation, customer/market analysis, etc. Many different clustering algorithms for centralized systems have been proposed. These algorithms can be classified into five classes: partition-based clustering, hierarchical clustering, grid-based clustering, density-based clustering, and model-based clustering. Partition-based clustering algorithms partition n data objects into k partitions which optimize some objective function, e.g., K-mean [93]. Hierarchical clustering algorithms create a hierarchical decomposition of the data set, which can be represented by a tree structure called *dendrogram*, e.g., [39, 52, 134]. Grid-based clustering algorithms divide the data space into rectangular grid cells and then conduct some statistic analysis on these grid cells, e.g., [11, 116, 124]. Density-based clustering algorithms cluster data objects in densely populated regions by expanding a cluster towards the neighborhood with high density recursively, e.g., [11, 13, 42]. Model-based clustering algorithms fit the data objects to some mathematical models, e.g., [29]. These techniques require all the data to reside in a single site. However transferring all the data from widely spread data sources to a centralized server for clustering is not desirable in P2P systems due to the lack of a centralized server, excessive communication overhead, privacy concern, etc. In addition, these clustering techniques were designed to minimize the computation cost and/or disk access cost. However, minimization of the communication cost is the primary goal for the design of clustering techniques in P2P systems.

2.2 Distributed Resource Discovery

Resource Discovery in LSDNs goes through several stages: design of search techniques in *unstructured overlays*, network topology optimization, and design of *structured overlays*. In the following, I summarize the representative studies in each stage.

2.2.1 Search in Unstructured Overlays

In a unstructured P2P overlay, like Gnutella [2], peers store data objects locally and form a random topology. Primary search techniques proposed for unstructured P2P overlays are flooding and random walk [9, 92]. While the search costs in unstructured P2P overlays may not be low in terms of the total number of messages and/or the number of hops traversed per search, the advantages are in the low maintenance cost, (no additional information maintained at each node), making them relatively easy to handle membership changes[1] and data content changes. In addition, unstructured P2P overlays pose no restrictions on the types of queries that can be supported effectively.

Two main strategies have been explored for searching in unstructured P2P overlays:

- **Blind search:** This strategy lets messages poll nodes, without having any idea of where the data may be held, till the required items are found. Gnutella and random walk use such a strategy. In Gnutella, a search message is forwarded by a peer to all its neighbors until the message reaches a certain preset distance. The down side of this strategy is the possible network overload due to a large number

[1] We call peer join/leave/failure collectively as membership changes.

of generated search messages. To address the issue of excessive traffic caused by the flooding search, random walk chooses to forward a search message from a peer to one or more randomly selected neighbors. However, this approach incurs a long latency to satisfy a request.

- **Directed search:** This strategy maintains additional information in the peer nodes (which blind search does not require) in order to reduce network traffic. Consequently, messages are directed specifically along paths that are expected to be more productive. The additional information is typically maintained as indexes over the data that are contained either within hierarchical clusters [3] or by nearby neighbors [35, 76, 128]. In addition to the high storage cost incurred by storing the index and high maintenance overhead incurred by index update, this indexing approach requires determining what attributes to index a priori, thus constraining the search that can be supported. Our previous study in [82] applies *signature* techniques to provide flexible search ability (i.e., supporting arbitrary queries) and better search message traffic behavior at a lower storage cost and maintenance overhead than index-based mechanisms. In [100], the authors propose to replicate the summary of local data stored at each peer in the entire network to facilitate query processing. While this proposal is expected to work well in a relatively small scale and static P2P system, the overhead of disseminating the summary information to a large scale dynamic network is expected to be too high.

2.2.2 Topology Optimization

Orthogonal to the above studies on search in a given network topology, some studies propose to optimize the network topology by organizing peers sharing similar documents or issuing similar queries together in the network [21, 57, 102, 103]. In [21], a centralized server clusters documents and then organizes peers sharing similar documents into peer clusters. In [102], the super-peer in a cluster decides which peer can join its cluster. [57] also mentions the idea of clustering peers with similar interest together without discussing how to define the interest similarity among peers and how to form clusters. [103] relies on periodic message exchanges among peers to keep track of other peers with similar documents, which incurs high message overhead.

A couple of studies, e.g., [63, 133], improve the search performance of unstructured overlays by organizing the peers into a small world like network. Due to the localized optimization of the network topology in [133], queries still need to be propagated to a large portion of the network, incurring high search cost. Reference [133] caches search results at peers following the principle of small world network. The effectiveness of this proposal highly depends on the query access pattern.

2.2.3 Design of Structured Overlays

With improved search techniques and optimized network topology, the search performance in P2P systems is improved. Nevertheless, the communication overhead and search latency incurred by majority of the aforementioned techniques are still very high, i.e., they are almost linear to the network size. Thus, a surge of studies propose to systematically organize the peers and data shared among the peers into structured overlays.

The basic idea of structured overlays is to assign identifiers to peers and data objects. A peer stores the location information (index) of the data objects whose identifiers are close to its identifier. Peers themselves organize into an overlay structure in accordance with the order defined by their identifiers. Different proposals differ in how they assign identifiers to data objects and peers, and what overlay structures the peers organize into. These differences in turn determine the functionality of the proposed overlays, such as which types of queries are supported by the overlays, what is the number of attributes per data object that the overlays can support, whether the overlays are adaptive to dynamic changes on data distribution, and whether the access load on peers are balanced. Table 2.1 highlights the major differences among some representative overlay structures. In the following, I briefly review these representative overlay structures roughly following the order in the table, together with some other similar overlay structures.

Table 2.1. Comparison of representative overlay structures

Overlays	Dimensions	Queries supported	Maintenance cost	Adaptivity	Access load balancing
CAN	low to medium	equality	low to high	no	no
Chord	1	equality	low	yes	no
Skipgraph	1	equality, range, KNN	low	yes	no
Mercury	low	equality, range	high	yes	yes
Brushwood	low	equality, range, KNN	low	no	no
BATON*	low	equality, range, KNN	low	yes	skewed
pSearch	high	equality, range	high	no	no

Distributed hash tables (DHTs), e.g., CAN [106], Chord [117], Tapestry [135], Pastry [110], Symphony [97], and P-Grid [8], organize peers and data objects in accordance with randomly hashed keys and can efficiently support simple key based equality (exact match) queries. However, they can not support complex queries, such as range query or K nearest neighbor queries. Since these overlays adopt random hashing to assign data objects to peers, they distribute the storage load evenly among peers. Nevertheless, they assume that all data objects are accessed uniformly, and thus they are sensitive to nonuniform access pattern.

In order to support complex queries in LSDNs, studies following DHTs organize peers and data objects in accordance with the attribute values instead of randomly hashed values as in DHTS. Different from DHTs, storage load balance does not come for free for this set of overlays since it is rarely the norm that data objects are uniformly distributed in the data space. However, majority of these studies, except [27], does not address the issue of load balancing. In the following, I briefly review this set of studies.

Skipgraph [14] and skipnet [55] organize peers into a distributed skip list. They support complex queries in one-dimensional space. However, they are not designed to support multi-dimensional data objects.

Quite a few studies investigate how to support multi-dimensional range queries in LSDNs. These works can be largely classified into two classes according to their underlying data index structures: *space partitioning based approach*, e.g., [28, 49, 132], and *object grouping based approach*, e.g., [34, 59, 60]. The basic idea of space partitioning based approach is to recursively partition the data space into subregions. Each peer is responsible for a subregion and stores/maintains the location information of all the data

objects in the subregion. Object grouping based approach groups data objects nearby recursively into a hierarchical structure. Peers take the responsibility of different nodes in the hierarchy. The index structures in both of these two approaches are similar to their counterparts developed in the database community. The space partition based approach mainly follows the design principle of k-d-tree [23], and object groups based approach mainly follows the design principle of R-tree [54]. Although the underlining index structures in these overlays are similar to centralized index structures, the design of these overlays are nontrivial given the decentralized nature of P2P systems.

Space partition based overlays inherit the following drawbacks of space partition based indexes. First, they have to index the whole data space including the dead space, increasing the index maintenance overhead as well as query cost. Second, these techniques do not adapt to the dynamic changes on data distribution. Although these works also construct tree structures, these tree structures are static and can become unbalanced, affecting the performance of these proposals in different aspects.

Among the object grouping based approach, VBI [60] and BATON* [59] simply assign each tree node to a peer and then establish a chord-like routing structure on each level of the tree. They create skewed traffic distribution. While the peers responsible for the lowest level of the tree perform majority of the routing, the peers responsible for the higher levels of the tree are hardly used. Fat-Btree [129] is a distributed balanced tree designed for parallel database system. It can only supports one-dimensional data objects. In addition to the aforementioned limitations, all these works couple the tree data structure with the overlay, making the maintenance and load balancing complicated and costly.

Besides the above works on multi-dimensional range queries in P2P systems, Mercury [27] addresses multi-dimensional range query by constructing multiple index structures with one for each attribute. In addition, it tackles the issue of load balancing by proposing a sampling based mechanism, which will be compared with in Section 4.3.3. In Mercury, each data object needs to be inserted to multiple index structures. It incurs high index maintenance overheads, especially when the number of attributes is large. In addition, since different attributes are indexed separately, it is not clear how Mercury can be extended to support queries that require the interplay among different attributes, e.g., KNN query.

Some works propose techniques to support content-based search in P2P systems by leveraging DHTs, e.g., [108, 122]. They share the similar design principle as Mercury. The basic idea proposed in [108] is to publish each keyword of a document on top of DHTs. Since the number of keywords associated with each document is large (in the order of hundreds or even thousands), publishing each of these individual keywords of a document on top of DHTs incurs excessive overhead. In addition, answering the searches involving multiple keywords requires join among the peers that index these keywords, which is an very expensive operation in P2P systems. Although [108] proposes various techniques to address the aforementioned issue associated with search, it does not address the excessive overhead incurred by publishing the keywords to the system. pSearch [122] first obtains the latent semantic index [26], a type of high dimensional semantic vector for text document, groups it into multiple sub-vectors, and then indexes each of these sub-vectors through CAN overlay network. Similarly, it incurs high index publishing overheads and search costs since each index publishing/search involves multiple

operations with one corresponding to each sub-vector. In addition, since pSearch does not capture all data dimensions, it needs to rely on complex heuristics to find similar enough data objects.

Reference [19] proposes a technique based on locality preserving hashing, which mandates complex/costly index construction and only provides approximate query answers. Reference [91] proposes a networked R-tree structure in super-peer networks. This study is different from our work since we focus on query processing in pure peer-to-peer systems without assuming the existence of super-peers. Reference [123] proposes an algorithm for 1NN queries on two-dimensional data (spatial data) based on Quad-tree structure. Quad-tree can only support two-dimensional data and constructing similar structures as Quad-tree for high dimensional data in P2P systems is quite costly. Reference [17] conducts similarity queries based on an accurate Voronoi diagram, which is too costly to construct and maintain. In addition, several works investigate how to process range queries on top of an existing overlay structure, e.g., [12, 48, 50, 53, 112].

In summary, the aforementioned overlays are deficient in one or more of the following aspects: 1) They can not support high dimensional data object (such as documents) efficiently; 2) They are constrained to support simple queries; 3) They are static structures and sensitive to changes on data distribution; 4) They assume that data objects are accessed uniformly and ignore the nonuniform access pattern.

2.3 Distributed Knowledge Discovery

Although quite a few studies focus on information retrieval and query processing in LSDNs, knowledge discovery in LSDNs is still in its infancy. While majority of the

studies in this area targets on extracting simple aggregate values from the system, only a few studies start to investigate extracting complex statistics from the system. In the following, I briefly review these studies starting from the ones collecting simple aggregate values to the ones extracting complex statistics.

Some works investigate collecting the approximate count for an item in P2P systems (e.g., [20, 131]) and in sensor networks (e.g., [101]). These works stem from probabilistic counting [45]. [68] proposes a gossip mechanism to obtain the aggregates in P2P systems. [66] improves upon the proposal in [68] with better time and message complexity.

Reference [43] proposes techniques to address iceberg query, which is relevant to identifying frequent items. However, the technique is designed for centralized systems where all items reside at a single site. In addition, the proposed techniques are designed to minimize the number of disk accesses. Here, our primary focus is to minimize the communication overhead. [40, 71] propose techniques to identify large flow or frequent byte sequences in the network. Similarly these works assume that all the data are transferred to a centralized coordinator or multicasted to a small number of sites for processing. In our work, the data are inherently distributed among peers. Thus, transferring them to a central site or multicasting them to some sites is infeasible due to the excessive communication overhead.

[16] proposes a technique to perform top-k retrieval, i.e., retrieving the k items with the largest values, in P2P systems. This work is different from our study of netFilter in the following two aspects. First, top-k retrieval only needs to report k items, while in identifying frequent items, the number of items that need to be reported might be very

large. Second, in [16], the authors assume that each item only appears in one peer. On the contrary, we do not make such an assumption. Instead, we need to address which items to collect the global values for and how to collect the global value for an item from the system. Therefore, the problem of identifying frequent items that we are addressing is much more complex than the problem address in [16].

[69, 96] investigate obtaining an approximate set of frequent items with ϵ error tolerance. The item sets returned in these studies have two types of errors: 1) false positives; and 2) errors on the reported global values of items. The communication cost incurred in these studies is proportional to $\frac{a}{\epsilon}$ (where a is either a constant or proportional to $log(n)$), which could be very high when ϵ is small. Different from these studies, we investigate obtaining a precise set of frequent items without the aforementioned errors in the study of netFilter. Obtaining a precise set of frequent items is necessary in certain applications. For instance, false positives are not desirable in network attack detection. As another example, in cache management, the precise global values of the frequent items (e.g., keywords or documents) are required to facilitate cache replacement. The proposed techniques in [69, 96] are not applicable to address the problem that we are addressing here.

The studies in [32, 115] investigate detecting significant changes on individual items. Our focus in the study of wavenet is detecting changes on the data distribution, which is different from detecting changes on individual items. Heavy changes on individual items may not imply changes on the data distribution. Similarly, certain changes on the data distribution do not imply heavy changes on individual items, i.e., they may be caused by small changes on a large number of individual items. [70] proposes a technique

to detect changes in one data stream. This technique is not applicable in our case since we need to monitor change on the data that are inherently distributed among a large number of host nodes.

The work in [120] proposes to summarize the data at a host node based on kernel density estimation. This work assumes that data are continuous, and thus kernel density estimation can be applied to approximate the underlying density of the data. Here we do not make such an assumption, and the data that we are considering here are discrete in nature (such as the source-destination IP address pairs). In addition, kernel density estimation itself can not compress the data. Instead, it needs to rely on some other mechanisms, such as random sampling, to compress the data. However, random sampling is not sufficient to address MCDN.

A few recent works investigate monitoring certain values/items in the networks. For instance, [15] monitors the k items with the largest values. [30, 61, 104] answer simple continuous queries. [33] monitors duplicate-resilient aggregates (count, sum, quartile) in the networks, which is also based on probabilistic counting [45]. [37] evaluates set expressions, and [31] tracks approximate quartiles. The problem of MCDN, which we consider in the study of wavenet, is dramatically different from the problems addressed in these previous studies. The nature of MCDN requires a complete picture of every distinct item in the whole system, which is not addressed in these studies.

Some studies have focused on parallel clustering and distributed clustering. References [38, 46] propose parallel K-mean clustering algorithms, which first distribute the data to multiple processors and then let these processors execute iterative K-mean algorithm in parallel. A processor broadcasts its currently obtained k centroids to all

other processors. Once a processor receives centroids from all other processors, it forms global centroids before starting the next iteration of K-mean algorithm. Reference [18] proposes a distributed version of K-mean algorithm in sensor networks. The idea is very similar to the above works. The difference is that the algorithm is unsynchronized, i.e., different sites are not required to execute the same iteration of the algorithm at the same time. References [64, 113] propose distributed hierarchical clustering algorithms. Each site first performs clustering locally, then transfers some local statistics to a central site, which performs global clustering based on the local statistics. Reference [127] proposes a parallel density-based clustering algorithm. Similar to the parallel K-mean algorithms [38, 46], the data are first grouped into partitions and distributed to different processors. Each processor first conducts clustering on its local partition, and then sends certain data objects (called merging candidates) to a central site for global clustering. Reference [62] proposes a distributed density-based clustering algorithm. The basic idea of [62] is similar to [127]. The difference is that the data are inherently distributed among different sites, and various representations are proposed to summarize local statistics to be sent to the central site. In summary, the above algorithms either rely on a central site or multiple rounds of message flooding to form a global clustering model. Our study is different from these works fundamentally since we focus on large scaled and fully distributed environments where a central site is not available and flooding is not desirable.

Chapter 3

Content Based Search

In this chapter, we first present in Section 3.1 the design of semantic small world (SSW), which facilitates efficient content based search in P2P systems. We then present the research issues associated with processing complex queries in the framework of SSW and provide the detailed algorithms for complex queries in Section 3.2.

3.1 Design of Semantic Small World

SSW are based on the following four ideas: 1) semantic clustering: peers with similar semantics organize into peer clusters; 2) dimension reduction: to address the high maintenance overhead associated with capturing the high-dimensional data semantics in the overlay, peer clusters are adaptively mapped to a one-dimensional naming space; 3) small world network: peer clusters form into a one-dimensional small world network, which is search efficient with low maintenance overhead; 4) efficient search algorithms: a search is processed efficiently through the careful design of search algorithms.

3.1.1 Introduction

Given the vast repositories of information, it is undesirable to require users to remember the key or identifier associated with a document in order to search for such a document. Instead, it is more favorable to allow users to issue searches based on the content of documents (just as in the Internet today). This mandates the employment

of *content/semantic-based searches*[1]. The primary goal of this study is to design a P2P overlay network that supports efficient semantic based search.

Various digital objects, such as documents and multimedia, can be represented and stored as data objects in P2P systems. The semantics or features of these data objects can be identified by a k-element vector, namely, *Semantic Vector (SV)* (or called *feature vector* in the literature). Semantic vectors can be derived from the content or metadata of the objects. Each element in the vector represents a particular feature or attribute associated with the data object with weight representing the importance of this feature element in representing the semantics of the data object. The SV of a data object can be mapped to a point in a k-dimensional space. Thus, each data object can be seen as a point in a multi-dimensional *semantic space*. As a result, queries on data objects in this semantic space can be specified in terms of these attributes. The number of attributes (elements) capturing the semantics of data objects is normally very large, and thus the dimensionality of the semantic space (k) is high. We assume that semantic vectors are unit vectors[2]. Euclidean distance is used to represent the *semantic distance* between two SVs. In this study, we use document sharing as one representative application. Nevertheless, the issues and solutions are applicable to other applications as well. Latent semantic index [26], the semantic vector capturing the semantics of documents, is derived from the content of the documents. Each element in latent semantic index indicates a latent concept, and the value of an element indicates the

[1]We do not exploit the differences between semantic and content based searches. These two terms are used interchangeably in the paper.

[2]We can always normalize the vectors that are not unit vectors to unit vectors.

relevance of the document to the corresponding concept. The dimensionality of latent semantic vector is around 50-300.

There are several challenges faced by realizing our goal to design a P2P overlay network that supports efficient semantic based search.

- Based on the accumulated knowledge of *clustered indexes* in database research community, it is safe to assume that clustering data objects with similar semantics close to each other and indexing them in certain attribute order can facilitate efficient search of these data objects based on indexed attributes. Thus, to support efficient semantic based search, the peer hosts and data objects should be organized in accordance with the semantic space that they are located in.

- For many real life applications, the number of attributes used to identify data objects and to precisely specify queries is large. The high dimensionality associated with the large number of attributes raises very challenging research issues, which are not addressed in existing studies. A well designed P2P overlay network needs to be able to facilitate efficient navigation and search in a high dimensional space without incurring high maintenance overhead, which requires the design be radically different from what have been proposed so far in the literature.

- The P2P overlay network of our goal mandates all the good properties of a robust network such as scalability, load balance, and tolerance to peer failures.

- Searches should be efficiently supported in the overlay.

In the following, we presents the design of a P2P overlay network, SSW, which overcomes the above challenges to facilitate semantic based search. The primary contributions of this study are five-fold.

1. We adopt an effective semantic clustering strategy that places peers based on the semantics of their data.

2. We show a way to build a small world overlay network for semantic based P2P search, which is scalable to large network sizes yet adapts to dynamic membership and content changes.

3. To address the high maintenance overhead associated with the high dimensionality of semantic space, we propose a dimension reduction technique, called *adaptive space linearization* (ASL), for constructing a one-dimensional SSW (called *SSW-1D*).

4. We introduce the concept of *local partition tree* and *global partition tree* to facilitate search in SSW.

5. We conduct extensive experiments using both synthetic data set and real data set to evaluate the performance of SSW on various aspects, including scalability, maintenance, adaptivity to data distribution, resilience to peer failures, and load balancing.

The rest of this section is structured as follows. Background on small world network and pSearch are provided in Section 3.1.2. In Section 3.1.3, we provide an overview on the design of SSW. The details on dimension reduction and search are given

in Section 3.1.4 and 3.1.5, respectively. The methodology and results of performance evaluation are presented in Section 3.1.6 and 3.1.7, respectively. Finally, we summarize this study and point out the relevant issues that can be explored further in Section 3.1.8.

3.1.2 Background

3.1.2.1 Small World Network

The *small world phenomenon* was first observed in 1967 by Stanley Milgram through an empirical experiment, which suggested that any two individuals in a social network are likely to be connected through a short sequence of intermediate acquaintances [99]. Later research studies show that many natural and technological networks, such as neural network, film collaboration network, and power grid, etc. also display similar characteristics [125].

Small world networks can be characterized by *average path length* between two nodes in the network and *cluster coefficient* defined as the probability that two neighbors of a node are neighbors themselves. A network is said to be small world if it has small average path length (i.e., similar to the average path length in random networks) and large cluster coefficient. (i.e., much greater than that of random networks). Studies ([73, 74]) on a spectrum of networks with small world characteristics show that searches can be efficiently conducted when the network has the following properties: 1) each node in the network knows its local neighbors, called *short range contacts*; 2) each node knows a small number of randomly chosen distant nodes, called *long range contacts*, with probability proportional to $\frac{C}{d}$ where d is the distance. and C is the normalization constant that brings the total probability to 1. A search can be performed in $O(log^2 N)$

steps on such networks, where N is the number of nodes in a network [74]. The constant number of contacts (implying low maintenance cost) and small average path length serve as the motivation for constructing a small world overlay network in our approach.

3.1.2.2 pSearch and Rolling Index.

pSearch [122] is the work most relevant to our study based on the authors' best knowledge, and thus we compare with it when necessary in this paper. While pSearch is intended for searching text documents only and can not be simply extended to support other types of digital data objects, our proposal is rather generic and can support searches for a variety of digital data objects in P2P systems. pSearch applies a dimension reduction technique, *rolling index*, on top of CAN to realize a semantic-based search engine. Rolling index partitions the lower dimensions of the semantic vectors into p subvectors where each subvector consists of m dimensions with m as the dimensionality of CAN overlay. The partial semantic space corresponding to each subvector is then mapped into the key space of CAN. To process a semantic based search, p separate searches are performed on the CAN key space based on some heuristics. The most similar data object(s) in the result of these p searches is returned as the answer. Rolling index can be applied on top of other overlay networks, such as CHORD, or small world network (as we demonstrate later).

Figure 3.1 shows an example of pSearch mapping a data object to a 2-dimensional CAN. The example shows that the first six elements of the semantic vector (totally 300 elements) are grouped into three 2-dimensional subvectors and mapped to three partial semantic spaces realized in one CAN.

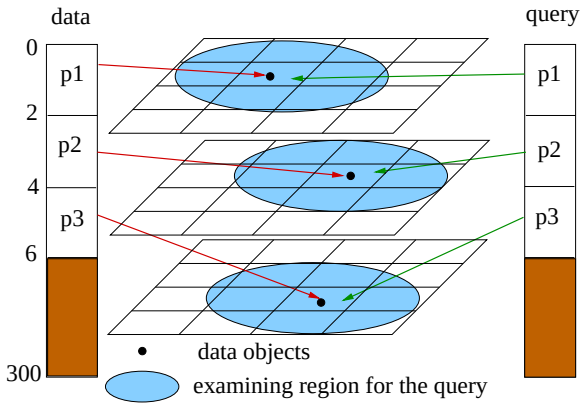


Fig. 3.1. An illustrative example of pSearch.

Although simple, rolling index incurs high index publishing overheads and search costs since each index publishing/search involves p operations with one corresponding to each subvector. In contrast, the dimension reduction technique we propose, ASL, requires only a single operation for index publishing and search. In addition, ASL considers all semantic information during a search, instead of only m dimensions as in rolling index, thereby no complex heuristics are required to direct the search (which are required by pSearch).

3.1.3 Semantic Small World

In this section, we present the overview on the design of semantic small world (SSW) and provide the technical details on constructing a k-dimensional SSW.

3.1.3.1 Overview

SSW plays two important roles: 1) an overlay network that provides connectivity among peers; and 2) a distributed index that facilitates efficient search for data objects. In SSW, the peers and data objects are organized in accordance with the k-dimensional semantic space populated by the k-dimensional SVs capturing the semantics of data objects. As such, in addition to navigation, a peer in the overlay network is responsible for management of data objects (or the location information of data objects stored at other peers - referred to as *foreign indexes*) corresponding to a *semantic subspace*. Foreign indexes, similar to the leaf node pointers of typical tree-based index structures, provide location information regarding to where data objects are physically stored[3]. To enhance the robustness of SSW, instead of assigning each individual peer to a semantic subspace, several peers form into a *peer cluster* to share the responsibility of managing a semantic subspace. These peer clusters self-organize into a small world network.

Corresponding to a k-dimensional semantic space, a k-dimensional SSW can be formed as follows. Each peer in this k-dimensional SSW maintains s *short range contacts* and l *long range contacts*. The short range contacts are selected to ensure the connectivity among peers so that a search message issued from any peer can reach any other peer in SSW. For a k-dimensional semantic space, the short range contacts of a peer P_1 can be intuitively set to the peers in the neighboring clusters next to P_1 in both directions of the k dimensions ($s = 2k$). Note that it is possible to use a s smaller than

[3]Due to the potential high cost of redistributing a large number of data objects within the overlay network, we choose to have a newly joined peer to publish the location information of its locally stored data objects to the other peers managing the subspaces corresponding to semantics of those data objects.

$2k$ as long as the short range contacts can ensure the connectivity among peers. On the other hand, the long range contacts aim at providing short cuts to reach other peers quickly. Via short range contacts and long range contacts, navigation in the network can be performed efficiently.

There are several critical issues that need to be addressed in the design of SSW: 1) *semantic clustering* - how to organize peers with similar semantics into peer clusters; 2) *dimension reduction* - how to handle the maintenance issue of the overlay network given the high dimensionality of the semantic space; 3) *search* - how to conduct searches efficiently in SSW. In the following, we briefly discuss our strategy for semantic clustering while introducing the tasks of constructing a k-dimensional SSW. We leave the discussions on dimension reduction and search to the following two sections.

3.1.3.2 Constructing a k-Dimensional Semantic Small World

Constructing a k-dimensional SSW depicted above involves two major tasks: 1) organizing peers with similar semantics into peer clusters; 2) constructing an overlay network across the peer clusters to form a semantic small world network.

Semantic Clustering. In order for peers with similar semantics to form into peer clusters, we need to address the following two sub-issues: 1) *peer placement* - where in the semantic space a peer should be located; 2) *cluster formation* - what is the strategy for forming clusters.

Peer Placement. We assume that the new peer obtains the SVs of its local data objects by local computation. Then, it executes a clustering algorithm (e.g., k-means) on its local data objects to clusters them into *data clusters* consisting of data objects with

similar semantics. A peer chooses the *centroid of its largest data cluster* as its position in the data space. This position is called *semantic position* of this peer, determining which semantic subspace (and which peer cluster) that this peer is to be placed in. While we assume a single join point here, multiple join points can be used if the peer node has sufficient resources.

Using centroid of the largest data cluster in a peer node to decide the peer's position in the semantic space has several positive effects. For example, if a node has relatively homogeneous data set (which is likely to be the case in real life), the semantic subspace where a peer resides in is also where most of its data objects fall into, thereby reducing the cost to publish foreign indexes. Moreover, the queries issued by the peers in the nearby subspace usually exhibit similar locality, i.e., a peer is likely to query for data objects with similar semantic meaning as its own data objects. Our construction of SSW exploits these characteristics naturally and still works better than other state-of-the-art P2P search techniques even without these localities (demonstrated later in the paper).

Cluster Formation. A cluster of peers share the responsibility of managing a data subspace to make the system adaptive to dynamic membership changes and achieve fair load distribution. A new peer joins a peer cluster based on its semantic position obtained as above. If the number of peers in a peer cluster exceeds a predefined threshold value M, the peer cluster and the corresponding semantic subspace is partitioned into two in a way adaptive to data distribution based on our space partition strategy as discussed next.

Our space partition strategy is aiming at load balancing (based on the data distribution within a subspace rather than the covered area of the subspace). To proceed, two

peers in the peer cluster that are semantically farthermost from each other are selected as the seeds for the two sub peer clusters. Then, peers in the original peer cluster are alternatively assigned to the two sub peer clusters based on the shortest distance to the seeds. Finally, the corresponding subspace is partitioned at the middle point of the dimension that has the largest span between the centroids of the SVs for the data objects in the two sub peer clusters (low order dimensions are used to break ties). This is similar to how R-tree nodes are split [54]. Based on this strategy, we obtain two subspaces that have relatively equal load (in terms of the number of foreign indexes) even though the physical size of the two subspaces may not be equal. Existing overlay networks, such as CAN and CHORD, simply partition a space into two equal sized subspaces without considering the load distribution in the two subspaces.

Overlay Network Construction. To construct the overlay, each peer maintains a set of short range contacts, each of which points to a peer in a neighboring peer cluster, and a certain number of long range contacts. A long range contact is obtained as the peer responsible for a point randomly drawn from the semantic space following a distribution, $\frac{1}{d^k}$ where k is the dimensionality of the semantic space and d is the semantic distance. These extra long range contacts reduce the network diameter and transform the network into a small world with poly-logarithmic search cost (Theorem 3.1). In addition, there are no rigid rules on which specific distant clusters should be pointed to by long range contacts. This flexibility of long range contacts selection can be utilized easily to make SSW adapt to locality of interest (to be detailed in Section 3.1.5).

THEOREM 3.1. *Given a k-dimensional semantic small world network of N peers, with maximum cluster size M and number of long range contacts l, the average search path length for navigation across clusters is* $O(\frac{log^2((\frac{2N}{M})^{1/k})}{l})$.

Proof: Before proceeding to prove the theorem, we derive the normalization constant C which brings the sum of the probabilities for all candidate long range contacts of a peer to 1. In order to derive C, we first derive the sum of appearance frequencies for all candidate long range contacts of a peer, F. After deriving F, C is simply $\frac{1}{F}$.

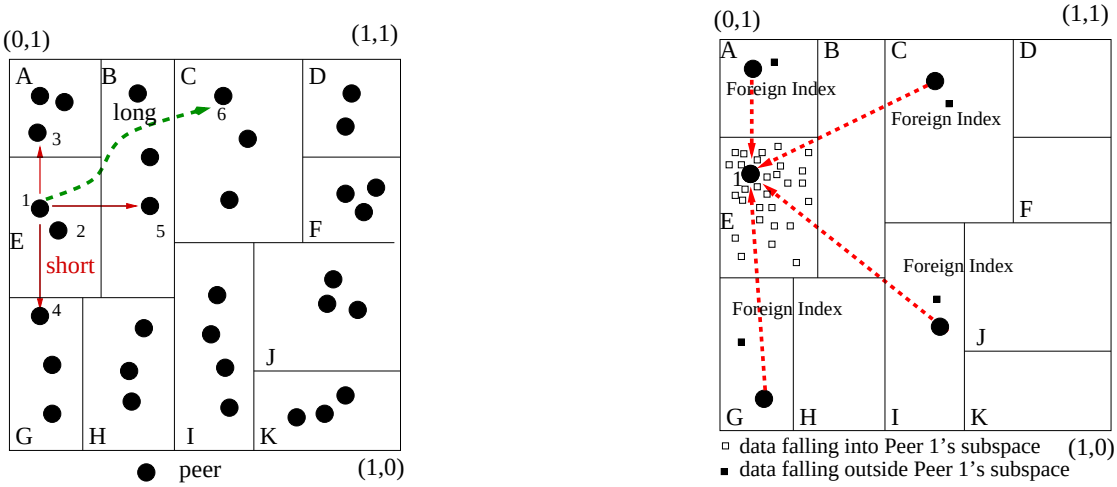
A peer P_1 chooses another peer at distance x as one of its long range contacts using the pdf: $f_x = \frac{1}{x^k}$ for $x \in [r, 1]$ where r, the minimum distance of a long range contact, is the average diameter of cluster subspace (will be derived later). All of the peers at distance x to Peer P_1 form the surface of a volume with P_1 as the center and x as the radius, whose area is x^{k-1}. Therefore, the sum of the appearance frequency for all candidate long range contacts at distance x is $F_x = x^{k-1} \cdot f_x = \frac{1}{x}$. We can derive the sum of appearance frequency for all candidate long range contacts thereby as $F = \int_r^1 F_x dx = ln\frac{1}{r}$. Thus, $C = \frac{1}{F} = \frac{1}{ln(1/r)}$.

After deriving C, we proceed to prove the theorem. Similar to the proof process employed in [74], we separate search process into phases $1, ..., log(1/r)$. Let d be the distance from a message's current node to the destination and $d_i = \frac{1}{2^i}$. Search is at phase i if $d_{i+1} \leq d < d_i$. Phase i ends when the message is forwarded to a peer less than d_{i+1} distance away from the destination. The set of peers less than d_{i+1} distance away from the destination is denoted as A_{i+1}, whose volume is d_{i+1}^k. The largest distance from a peer at phase i to a peer in set A_{i+1} is $d_i + d_{i+1}$ (denoted as $\hat{d}_{i+1}$). Since a peer

has l long range contacts, the probability that a peer at phase i has contacts to set A_{i+1} is at least $l \cdot d_{i+1}^k \cdot f_{\hat{d}_{i+1}} = l \cdot C \cdot (\frac{1}{3})^k = \frac{l}{3^k ln(1/r)}$. Therefore, search message requires $\frac{3^k ln(1/r)}{l}$ steps to reach next phase on average. Since there are total $log(1/r)$ phases, the total search path length is $O(\frac{3^k log^2(1/r)}{l})$.

Now we need to derive r. In the system, the average size of a cluster is $M/2$, so there are totally $\frac{N}{M/2}$ peer clusters and each cluster takes charge of $\frac{M}{2N}$ portion of the semantic space on average. Therefore, the diameter of each partition, r, is approximately $(\frac{M}{2N})^{1/k}$. We can then obtain the path length for search across clusters as $O(\frac{log^2(\frac{2N}{M})^{1/k}}{l})$. □

Figure 3.2 shows an example of SSW ($k = 2$). As shown in the figure, the search space is partitioned into 11 clusters after a series of peer joins and leaves. Figure 3.2(a) shows the overlay structure. Peer 1 in cluster E maintains three short range contacts to neighboring peer clusters A, B and G and one long range contact to a distant peer cluster C. The contacts of other peers are not shown here for clarity of presentation. Figure 3.2(b) illustrates the concept of foreign indexes. The dark circles denote the semantic positions of peers in the semantic space. The small rectangles represent the data objects stored in Peer 1. Most of them (the white rectangles) are located in the subspace of Peer 1, but some of them (the dark rectangles) are mapped to other subspaces. Thus, the location information of those data objects are stored as foreign indexes at the peers in charge of those subspaces.



(a) overlay structure (b) foreign indexes

Fig. 3.2. An illustrative example for SSW.

3.1.4 Dimension Reduction

The k-dimensional SSW obtained by simply assigning short range contacts in all dimensions of the corresponding semantic space (as described in above section) is feasible when k (the dimensionality) is small. When k is large, the maintenance of such an overlay becomes costly and non-trivial due to the decentralized and highly dynamic nature of P2P systems. Unfortunately, as pointed out earlier in this section, the number of attributes capturing the semantics of data objects is normally very large, and thus the dimensionality of the semantic space (k) is high. For instance, the latent semantic vector capturing the semantics of documents is around 50-300 [26].

One idea to address this issue is to construct an overlay network of low dimensionality to support the function of semantic based search in the high dimensional semantic

space. This idea can be realized by linearizing the peer clusters from the high dimensional semantic space to a one-dimensional naming space, i.e., assigning a *ClusterID* to each peer cluster, and then constructing a one-dimensional SSW (SSW-1D) over the linearized naming space. SSW-1D is constructed as a double linked list consisting of peer clusters connected via two short range contacts of each peer. In addition to this linear network structure that provides basic connectivity, long range contacts provide short cuts to other peer clusters, which facilitate efficient navigation. While the original semantic space has been partitioned and then linearized, the peer clusters in SSW-1D are still corresponding to their original semantic subspace of high dimensionality.

In order to facilitate efficient navigation in SSW-1D based on the high dimensional semantic information (i.e., SVs) corresponding to the original semantic space, the following two issues should be addressed carefully:

- *Naming Encoding.* The mapping from the high dimensional semantic space to a one dimensional naming space should preserve data locality, i.e., clusters located nearby in the semantic space should be assigned ClusterIDs with similar values.

- *Naming Embedding.* The aforementioned mapping should be recorded or embedded in the overlay network so that an arbitrary peer can determine the ID of the peer cluster responsible for a given data object (this is necessary to process a search, which will become clear in the following section).

The well known space filling curves such as Hilbert curve, Z-curve, etc., can only be employed to map a regularly coordinated high-dimensional space to a low-dimensional

space. In our case, the high-dimensional semantic space is adaptively (irregularly) partitioned according to data distribution, and thus these techniques are not applicable here.

In this study, we propose a technique, called *adaptive space linearization* (ASL), which linearizes the peer clusters in the high dimensional semantic space into a one-dimensional naming space during the process of cluster split in SSW construction. In the following, we explain how ASL addresses the aforementioned two issues, i.e., naming encoding and naming embedding.

3.1.4.1 Naming Encoding

We observe that the partition of the data space can be represented by a 'virtual' binary tree and encoding this binary tree properly can lead to a location-preserving naming encoding scheme. Based on this observation, we propose the following naming encoding scheme. We use a bit string with fixed length to represent ClusterIDs. We call the virtual binary tree depicting the partition of the data space as *global partition tree (GPT)*. Note that GPT is not maintained anywhere in the network. We simply use it for the explanation of the partitioned space and the naming encoding scheme. The root node of the GPT represents the initial data space. Each partition event generates two subtrees where the left (right) subtree corresponds to the subspace with smaller (larger) coordinate along the partition dimension. The subspace corresponding to a leaf node in the GPT is called *leaf subspace*, which peers are residing in (and responsible for managing) . We associate each bit in the ClusterID (starting from the most significant one) with a level in the GPT (starting from the root level). Therefore, level i at the

GPT corresponds to the i^{th} most significant bit in ClusterID. These bits are set to 0 by the left subtrees and "1" by the right subtrees. Any tree node obtains a *tree label* by concatenating the bits along the path from the root to itself and padding the remaining bits in its ClusterID with 0 (note that the combination of the depth and the tree label uniquely identifies a node in the GPT). The tree label of a leaf node is the ClusterID of the peer cluster in charge of the corresponding leaf subspace.

Figure 3.3(a) shows the same snapshot of the system as Figure 3.2 where the whole semantic space is partitioned into 11 clusters with the ClusterIDs indicated in the figure. Figure 3.3(c) depicts the corresponding GPT (with tree labels). We illustrate the process in a 2-dimensional space with a naming space of 4-bit long. In this example, the semantic space is first partitioned along the vertical line as indicated by "p = 1" in the figure. Peers at the left side and right side of this line obtain ID "0000" and "1000", respectively. The left side is then partitioned along the horizontal line as indicated by "p = 2". At this point, peers at the lower left side and top left side obtain ID "0000" and "0100", respectively. The data space is eventually partitioned as shown in the figure. We use the solid line to illustrate the order of the assigned ClusterIDs. The dashed portion of the line, naturally created since SSW-1D is a double linked list, indicates that a search can be performed bi-directionally.

3.1.4.2 Naming Embedding

We now proceed to the second issue - how the mapping from the high dimensional semantic space to the one dimensional naming space is embedded in the overlay network so that a peer can determine the ClusterID of the peer cluster responsible for a given

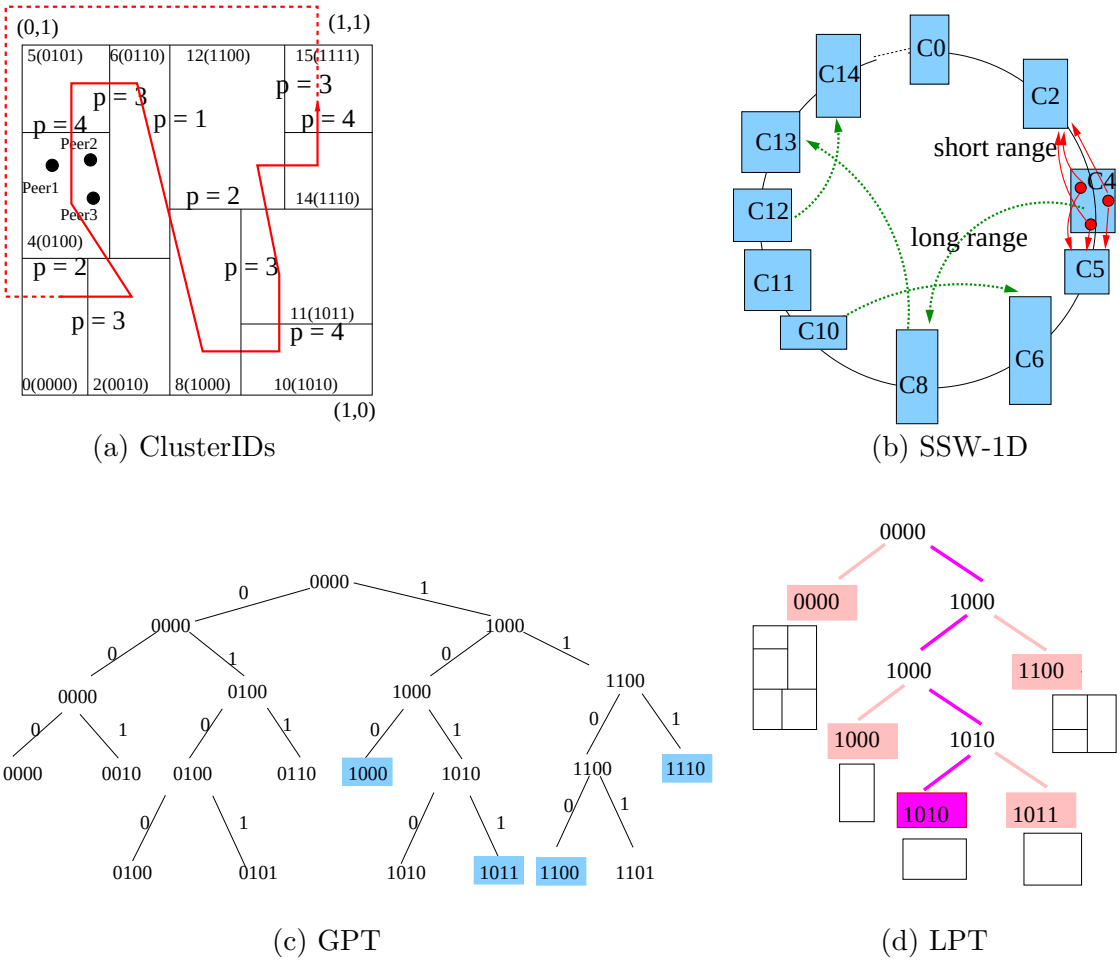


Fig. 3.3. An illustrative example of ASL.

data object. A simple scheme is to replicate the complete partition history, i.e., the entire GPT tree structure, at each peer. However, this scheme is not scalable since any subspace partition needs to be propagated to the whole network.

Here we propose to only store the history of the partitions (in terms of the partition dimension and partition point) that a peer has been involved with at this peer. Note that this partial partition history corresponds to the path in the GPT starting from the root to the leaf node corresponding to this peer's subspace. Thus, we call this partial partition history as *local partition tree (LPT)*. In the LPT, each node (except the leaf node) has two children representing the two subspaces generated in a space partitioning event. Between the two subspaces, the one that the peer does not reside in may undergo further partitioning that is not known to the peer, and thus it is called *old neighbor subspace*. The readers should not confuse the old neighbor subspaces with short range contacts or long range contacts. While a peer has maintained the history of the data space partition events generating these subspaces in its LPT, it doesn't necessarily have contacts in each of these old neighbor subspaces.

Figure 3.3(d) illustrates the LPT for a peer residing in Cluster 10 (marked by dark rectangle). Cluster 10 has been involved in four partitioning events, generating four old neighbor subspaces (marked by light rectangles). In this figure, we also show how the subspace corresponding to a leaf node in the LPT is partitioned further. From this figure we can see the two old neighbor subspaces (i.e., the tree node with label 0000 and 1100) are partitioned further and consist of a collection of smaller subspaces, respectively.

Figure 3.3(b) illustrates SSW-1D built upon the ClusterIDs. A peer in Cluster 4 maintains short range contacts to the neighboring clusters 2 and 5. It also maintains a

long range contact to a distant cluster 8. A peer in Cluster 8, 10 and 12 maintains a long range contact to Cluster 13, 6 and 14, respectively. The contacts of other peers are not shown here for readability.

3.1.5 Search

In this section, we explain the search process in SSW-1D. For simplicity, we refer SSW-1D as SSW in the rest of this paper where the context is clear. We present the algorithms for semantic bases searches, i.e., *approximate point query* (denoted as *point* query*), which is specified by a query semantic vector or query point q. Point* query returns the data objects matching q if there is such a data object in the system (similar to a regular point query); otherwise, it returns a data object similar to q (different from a regular point query). Note that a point query can be viewed as a special case of range query with the query range as 0.

To process point* queries, we need to address the following issue:

- *Search space resolution (SSR).* To process a query, a peer first needs to determine which portions of the overlay (peers, data subspaces) host the requested data (or the index information for the data). Since the overlay (index) structure of SSW is constructed over the linearized naming space, and each peer only has partial knowledge of the mapping from the high dimensional data space to the one dimensional overlay, SSR is a non-trivial issue.

The solution for SSR is presented in Section 3.1.5.1, followed by the algorithms for point* query in Section 3.1.5.2.

3.1.5.1 Search Space Resolution

The issue of SSR is not faced by point* query alone. Instead, it is an issue faced by other queries, such as range query and KNN query (to be addressed in details in next section). For presentation brevity, we address the issue of SSR in the context of range query, which is specified by a query point q and a query range r. As mentioned previously, a point query is a special case of range query with the query range r as 0.

Since SSW is constructed over the one dimensional naming space captured by ClusterIDs, SSR basically is to figure out the ClusterIDs of the peer clusters intersecting with the search space (or query region) associated with a given query. We propose a localized algorithm for SSR based on the concept of LPT (local partition tree) as follows. A peer examines its LPT starting from the root node and decides whether a subtree in the LPT should be pruned or not by examining whether the corresponding subspace intersects with the search space or not. To determine whether a subspace intersects with the search space, the peer first computes the minimum distance, *mindist*, from the query point to the subspace. If the computed *mindist* is greater than r (the range of the search space), the subspace does not intersect with the search space, and thus the corresponding subtree is pruned. Otherwise, the corresponding subtree is examined further. Both subtrees may be examined if both corresponding subspaces intersect with the query region. This process continues till the leaf nodes in the LPT, called as *terminal nodes*, are reached. We assign a *pseudo-cluster-name (PCN)* to the corresponding subspace of a terminal node. PCN is a tuple of $\langle a, h \rangle$, denoted as a_h, where a and h are set to the tree label and depth of the corresponding terminal node in the LPT. Here a and h correspond

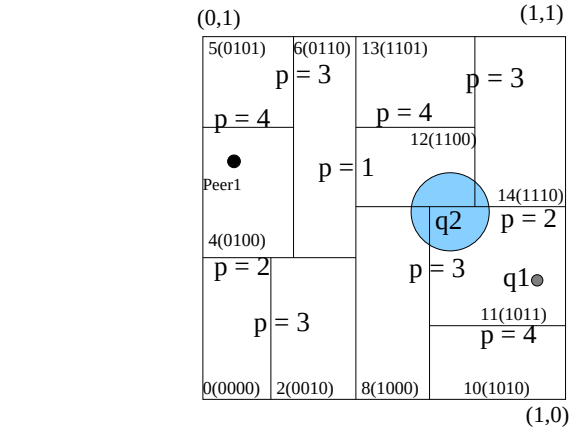
to the value and the number of bits resolved in the PCN respectively. If a PCN's value is the same as the ClusterID of current peer, this PCN is completely resolved. Otherwise, the PCN may be refined further by invoking SSR at the peers managing the old neighbor subspace corresponding to the terminal node. Algorithm 1 shows the pseudo code for SSR at a peer.

Algorithm 1 Algorithm for SSR.

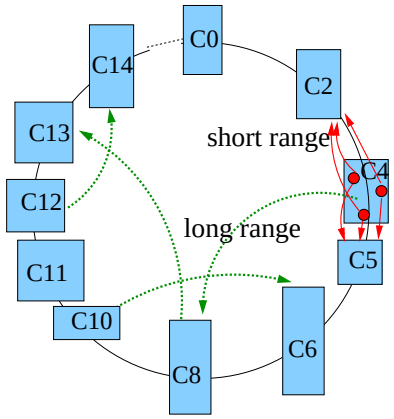
Peer *i* issue ssr(x, q, r, a, h): $i.ssr(x, q, r, a, h)$ **(x, set to the root node initially, is the current tree node to be examined in i's LPT. q and r are the center and range of the search space. a and h, set to 0 initially, indicate the value and the number of bits resolved in the PCN.)**

```
1: if x is the leaf node in i.LPT then
2:    Return a_h.
3: else
4:    h = h + 1;
5:    if mindist(q, x.left) ≤ r then
6:       Set the i^th most significant bit of a to 0.
7:       i.ssr(x.left, q, r, a, h).
8:    end if
9:    if mindist(q, x.right) ≤ r then
10:      Set the i^th most significant bit of a to 1.
11:      i.ssr(x.right, q, r, a, h).
12:   end if
13: end if
```

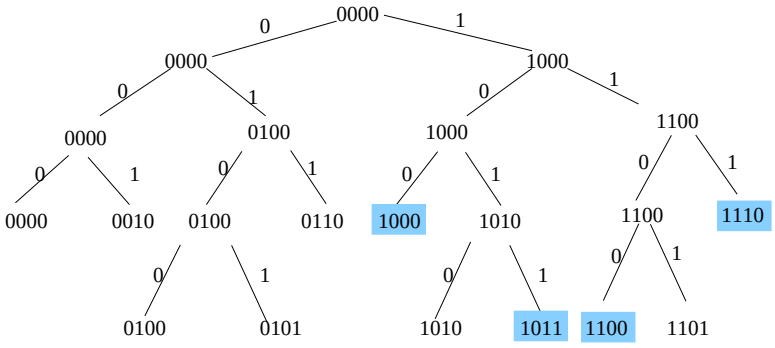
Figure 3.4(a) illustrates an example for SSR. The large circle ($q2$) intersecting with Clusters 8, 11, 12 and 14 depicts the search space (also marked on the GPT in Figure 3.4(b)). A peer in Cluster 10 resolves the search space and obtains the corresponding PCNs as 8_3, 11_4 and 12_2. The search space corresponding to PCN 12_2 is later resolved as 12_4 and 14_3 by a peer residing in the old neighbor subspace corresponding to the tree node with label 1100 in Cluster 10's LPT.



(a) search space



(b) SSW-1D



(c) GPT

Fig. 3.4. An illustrative example of SSR.

3.1.5.2 Query Algorithm

The algorithm to process an approximate point query consists of two stages: *navigation-across-clusters*, which navigates to the *destination peer cluster* covering the query point, and *flooding-search-within-cluster*, which floods the query within the destination peer cluster for query results. When a message is received, a peer first checks whether q falls within the range of its peer cluster. If that is not the case, SSR is invoked to obtain the PCN (pseudo-cluster-name) for q based on the partition history (LPT) stored at this peer. The query message is then forwarded to the contact with the shortest distance to the PCN in the one-dimensional naming space. The above process is repeated until the cluster whose semantic subspace covering q is reached. At this point, flooding-search-within-cluster is invoked by flooding the message to other peers in the same peer cluster. Then the data object with the highest similarity to the query is returned as the result.

Here, we show an example to illustrate point* query processing in SSW-1D. Let's go back to Figure 3.4(a). Assume Peer 1 in Cluster 4 issues a point* query $q1$ [0.9,0.3] indicated by the small grey circle in Cluster 11. Peer 1 first checks its own cluster range. Since [0.9,0.3] is not within the subspace of its peer cluster, Peer 1 invokes SSR and estimates the PCN for the query as 8_1. Peer 1 then forwards the message to Cluster 8, which invokes SSR and refines the PCN as 10_3. The message is then forwarded to Cluster 10, which refines the PCN further as 11_4. The query is finally forwarded to a peer in Cluster 11, which finds $q1$ is within its own cluster range and thus floods the message within its peer cluster to obtain the query results.

Despite the need for SSR, navigation in SSW-1D is efficient. We group the routing hop(s) to resolve one bit of PCN as a *PCN resolving phase*. A query may need to go through multiple PCN resolving phases, where each phase brings the query message half-way closer to the target. The following theorem obtains the search path length for SSW-1D (different from Theorem 1 where a peer has short range contacts along all the dimensions, in SSW-1D, a peer has only 2 short range contacts in a k-dimensional space ($k >> 1$) and thus it only has a partial knowledge of its neighborhood.).

THEOREM 3.2. *For a k dimensional space, given a SSW-1D of N peers, with maximum cluster size M and number of long range contacts l, the average search path length for navigation across clusters is $O(\frac{log^3(2N/M)}{l})$.*

Proof: The average number of peer clusters in the system is $\frac{2N}{M}$, which implies total $log\frac{2N}{M}$ bits in PCN need to be resolved, requiring $log\frac{2N}{M}$ PCN resolving phases. Starting from the most significant bit, each phase resolves one bit in PCN, thereby reducing the distance to the target cluster by half. S denotes the initial distance to the target cluster, i.e., $S = \frac{2N}{M}$. According to Theorem 3.1 (dimensionality k is 1 here), the path lengths for these $log\frac{2N}{M}$ phases are $\frac{log^2 S}{l}$, $\frac{log^2(S/2)}{l}$, $\frac{log^2(S/4)}{l}$, ..., $\frac{log^2(S/2^{logS-1})}{l}$, respectively. The search path length in SSW-1D is the summation of all these path lengths incurred by different phases, which is $O(\frac{log^3 S}{l})$. Hence, the above theorem is proved. □

During the query process, SSW adapts to locality of users interests as follows. Each peer maintains a *query-hit* list, which consists of the peers who have query hits (provide query results) in the past X queries issued by this peer. For every X queries, a node replaces the long range contact having the lowest hit rate with the entry having the

highest hit rate in the query-hit list with probability of $\frac{d_i}{d_i+d_j}$, where d_i and d_j represent the naming distance of the old long range contact and the candidate long range contact to current peer, respectively. We prove this update strategy maintains the properties of small world networks, i.e., the distribution of long range contacts is still proportional to $\frac{1}{d}$ where d is the distance of a long range contact.

THEOREM 3.3. *By replacing a long range contact at distance d_i by another peer at distance d_j with probability $\frac{d_i}{d_i+d_j}$, the distribution of long range contacts is proportional to $\frac{1}{d}$.*

Proof: Basically if we can prove at steady state the probability that a peer at distance d becomes a long range contact, denoted by p_{in}, equals to the probability that a long range contact at distance d is replaced by other peers, denoted by p_{out}, we prove the above theorem.

p_i denotes the probability that Peer i is the long range contact of current peer, which equals to $\frac{C}{d_i}$ where C is the normalization constant that brings the total probability to 1. $p_{i\to j}$ donates the probability that a long range contact i is replaced by another peer j, which equals to $\frac{d_i}{d_i+d_j}$. We can then obtain $p_{out} = \sum_{j=1}^{N} p_i \cdot p_{i\to j} = \sum_{j=1}^{N} \frac{C}{d_i} \cdot \frac{d_i}{d_i+d_j} = \sum_{j=1}^{N} \frac{C}{d_i+d_j}$, and $p_{in} = \sum_{j=1}^{N} p_j \cdot p_{j\to i} = \sum_{j=1}^{N} \frac{C}{d_j} \cdot \frac{d_j}{d_i+d_j} = \sum_{j=1}^{N} \frac{C}{d_i+d_j}$. Since $p_{in} = p_{out}$, this proves the above theorem. □

3.1.6 Simulation Setup

A random mixture of operations including peer join, peer leave and search (based on certain ratios) are injected into the network. During each run of the simulation, we

issue $10 \cdot N$ random queries into the system. The proportion of peer join to peer leave operations is kept the same to maintain the network at approximately the same size. The simulation parameters, their values and the defaults (unless otherwise stated) are given in Table 3.1. Most of these parameters are self-explanatory. More details for some of the system parameters, synthetic data set, real data set, and query workload are given below.

Table 3.1. Parameters used in the simulations of SSW

	Descriptions	**Values, default**
N	Number of peers in the network	256 - 16K, $\underline{1K}$
l	Number of long range contacts	$\underline{4}$
M	Size of peer clusters	1 - 1024, $\underline{8}$
n	Number of data objects per peer	1 - 100, $\underline{100}$
α_{d1}	Skewness of Dataseed-Zipf	0 - 1.0, $\underline{0}$
α_{d2}	Skewness of Data-Zipf	0 - 1.0, $\underline{0}$
γ	Percentage of peer join and peer leave operations	0% - 50%, $\underline{20\%}$
α_{q1}	Skewness of Queryseed-Zipf	0 - 1.0, $\underline{0}$
α_{q2}	Skewness of Query-Zipf	0 - 1.0, $\underline{0}$
W	SV dimension weight assignment	uniform, $\underline{linear}$, zipf
k	Dimensionality of semantic space	10 - 100, $\underline{100}$
R	Maximum query radius	0 - 1, $\underline{0.5}$

Synthetic Data Set. The data set is defined by the dimensionality of SV (and the semantic space), the weight of each dimension in SV, the number of data objects per peer (n), and the data distribution in the semantic space. The default setting for the dimensionality of SV (k) is 100. We use three different weight assignments for SV,

namely, uniform, linear, and Zipf dimension weight assignments. In the uniform dimension weight assignment, each dimension has the same weight. In the linear dimension weight assignment, Dimension i has weight $\frac{k+1-i}{k}$ $(1 \leq i \leq 100)$. Zipf-distribution is captured by the distribution function $\frac{1}{i^{\alpha}}$ where α is the skewness parameter. In our Zipf dimension weight assignment, α is set to 1, i.e., Dimension i has weight $\frac{1}{i}$. If unspecified, linear dimension weight assignment is the default setting.

Data distribution in the semantic space is determined by two factors: 1) semantic distribution of data objects among different peers; and 2) semantic distribution of data objects at a single peer. The former controls the data hot spots in the system and the latter controls the semantic similarity between data objects at a single peer, namely *semantic closeness*. To model both factors, we associate a Zipf-distribution each, *Dataseed-Zipf* for the former and *Data-Zipf* for the latter, with their skewness parameters as α_{d1} and α_{d2}, respectively. We first draw a seed for each peer following the Zipf distribution controlled by α_{d1}. This serves as the centroid around which the actual data objects for that peer are composed following α_{d2}.

Real Data Set. We derive the latent semantic indexes for 527000 documents in TREC version 4 and version 5 [4]. We distribute these documents to peers in the following two ways: 1) random data distribution, where documents are randomly distributed among peers, and 2) clustered data distribution, where documents are first assigned into N clusters using k mean algorithm and each cluster is then randomly assigned to a peer.

Workload Parameters. A point* query is specified by the query point. Similar to data parameters, we consider two factors in generating query points: distribution of query points emanating across the peers in the system and the skewness of the query

points emanating from a single peer. The former controls query hot spots (i.e. more users are interested in a few data items) in the system and the latter controls the locality of interest for a single peer, namely *query locality* (i.e. a user is more interested in one part of the semantic space). We use two Zipf distributions with parameters α_{q1} (for *Queryseed-Zipf*) and α_{q2} (for *Query-Zipf*) to control the skewness, i.e., α_{q2} captures the skewness of the queries around the centroid generated by α_{q1}.

While the main focus of this paper is to improve search performance with minimum overhead, we also try to explore the strengths and weaknesses of SSW in other aspects, such as fault tolerance and load balance. In this paper, we use the following metrics for our evaluations.

- **Search path length** is the average number of logical hops traversed by search messages to reach the destination.
- **Search cost** is the average number of messages incurred per search.
- **Maintenance cost** is the number of messages incurred per membership change, consisting of **overlay maintenance cost** and **foreign index publishing cost**. Since the size of different messages (join, query, index publishing, cluster split, cluster merge) is more or less the same (dominated by the size of one SV (400 bytes)), we focus on the number of messages in the paper for clarity.
- **Search failure ratio** is the percentage of unsuccessful searches that fail to locate existing data objects in the system.
- **Index load** is the number of foreign index entries maintained at a peer.

- **Routing load** is the number of search messages that a peer processed.

- **Result quality** is to measure the quality of the returned data object for a point* query. To calculate the result quality, we first calculate the normalized dissimilarity (Euclidean distance)[4] between the query and the result returned by SSW (or pSearch/SWRI), denoted as d_{real}, and the normalized dissimilarity between the query and the data object most similar to the query in the system, denoted as d_{ideal}. Then we use $1-(d_{real}-d_{ideal})$ to represent the result quality. When the difference between d_{ideal} and d_{real} is very small, the result quality is high.

3.1.7 Simulation Results

We have conducted extensive experiments under both synthetic data set and real data set to demonstrate SSW's strength on various aspects. In this section, we first demonstrate the scalability of SSW in terms of the size of the network and the number of data objects in the system. This is followed by an examination of the effect of peer cluster sizes on SSW. The benefits of constructing overlay based on semantics and updating long range contacts is subsequently illustrated with different workload behaviors. We then show the strength of SSW in tolerating peer failures and balancing the load. Lastly, we present the result quality of point* query. We present the results under different data sets when they display different trends. Otherwise, we present the results under one specific data set for presentation brevity and the interest of readers.

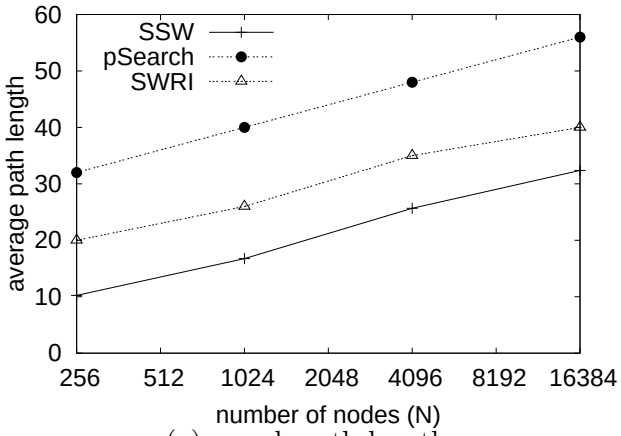
[4]To perform the normalization, we divide the Euclidean distance between two vectors by $\sqrt{k}$, which is the maximum Euclidean distance between two vectors in the semantic space.

3.1.7.1 Scalability

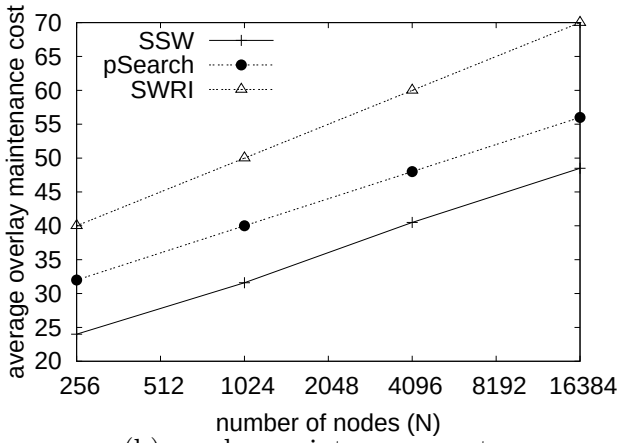
We vary the number of peers from 2^8 to 2^{14} to evaluate the search efficiency and maintenance cost of SSW. According to our preliminary simulation results ([87]), we find SSW with 4 long range contacts has reasonable trade-off between search efficiency and maintenance overhead for most of the γ settings, and we use this value in the experiments. Since pSearch does not use any clustering, we disable the clustering feature of SSW (i.e., cluster size M is set to 1) in this set of experiments. We have demonstrated that SSW can perform even better with appropriate cluster sizes (as detailed in section 3.1.7.2).

Figure 3.5(a) shows the average path length under uniformly distributed synthetic data set ($\alpha_{d1} = \alpha_{d2} = 0$). The results under other data sets are almost the same and are omitted. Since the size of peer clusters is set to 1 in this experiment, there is no flooding within a cluster and the average search path length for SSW represents the search cost as well. The search path length for SSW increases slowly with the size of network, confirming search path length bound in Theorem 3.2. In addition, the constant hidden in the big-O notation is smaller than 1 as shown in the figure. The plot of pSearch's path length has a slope close to SSW (with 4 long range contacts) but has a much higher offset. In fact, the search path length of SSW is about 40% shorter than that of pSearch at network size 16K. The search path length of SWRI is between pSearch and SSW. This confirms the effectiveness of ASL vs. rolling index in terms of search path length.

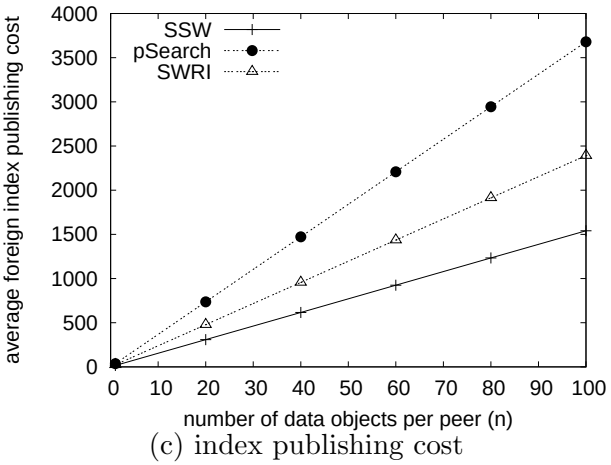
Overlay maintenance cost is proportional to the number of states maintained at each peer, which are 20, 24 (20 short range contacts and 4 long range contacts), 6 (2 short range contacts and 4 long range contacts) for pSearch, SWRI and SSW



(a) search path length



(b) overlay maintenance cost



(c) index publishing cost

Fig. 3.5. Comparing the scalability of SSW and other schemes.

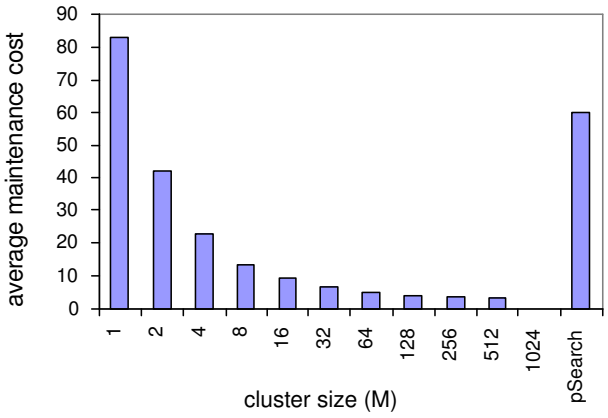
respectively. Figure 3.5(b) shows the overlay maintenance cost for the same experiments as Figure 3.5(a). These two figures confirm our expectation that compared to pSearch, SSW can achieve much better search performance with smaller overlay maintenance overhead.

The other maintenance cost to consider is the overhead of publishing foreign index upon peer joins (apart from the cost shown in Figure 3.5(b)). This cost is proportional to the number of data objects that need to be published, and the corresponding relationship is shown in Figure 3.5(c). Due to the fact that pSearch/SWRI have to publish a data object multiple times, the index publishing costs for pSearch/SWRI are much higher than that for SSW. In addition, the index publishing cost for SWRI is lower than that for pSearch. Note that these results for SSW are conservative since $\alpha_{d2} = 0$ (uniform data distribution) and the overhead is likely to be lower with any skewness (as will be shown later).

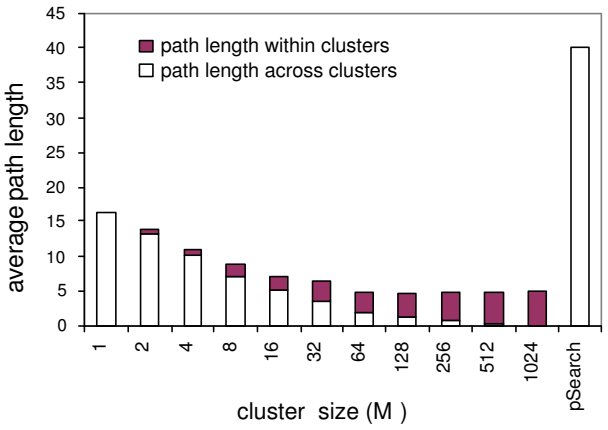
This set of experiments confirms the scalability of SSW. It also confirms our expectations that ASL is a better dimension reduction method than rolling index in terms of various aspects, including search cost and index publishing cost (later, we will show ASL is better than rolling index in terms of the result quality for point* query as well). In the remaining experiments, we only compare with pSearch for presentation clarity.

3.1.7.2 Peer Clustering Effects

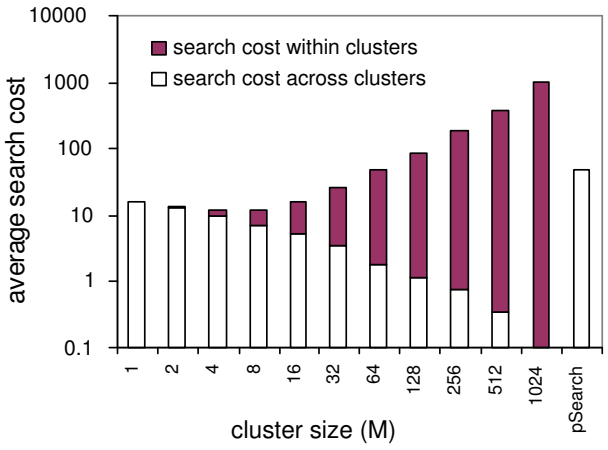
Until now the size of the peer cluster (M) has been set at 1. When M is larger, cluster splits or merges occur less frequently, resulting in lower overlay maintenance



(a) maintenance cost



(b) path length



(c) search cost

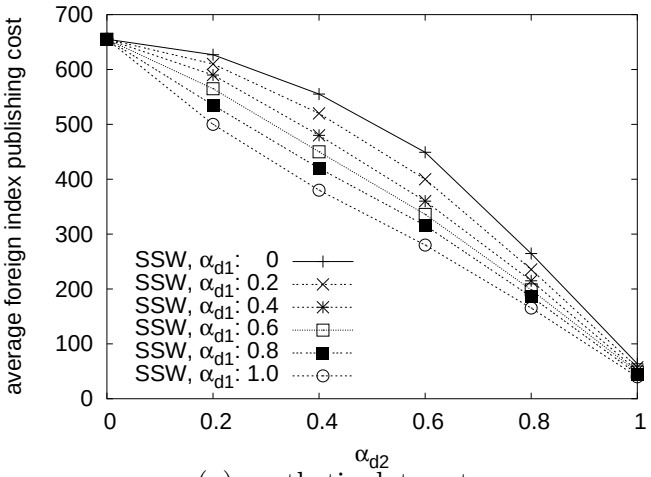
Fig. 3.6. Studying the effect of cluster size (M) for SSW.

costs. Further, the total number of clusters in the system decreases with larger cluster sizes, thereby reducing searches across clusters. The down-side of large sized clusters is the higher search cost within a cluster (due to flooding).

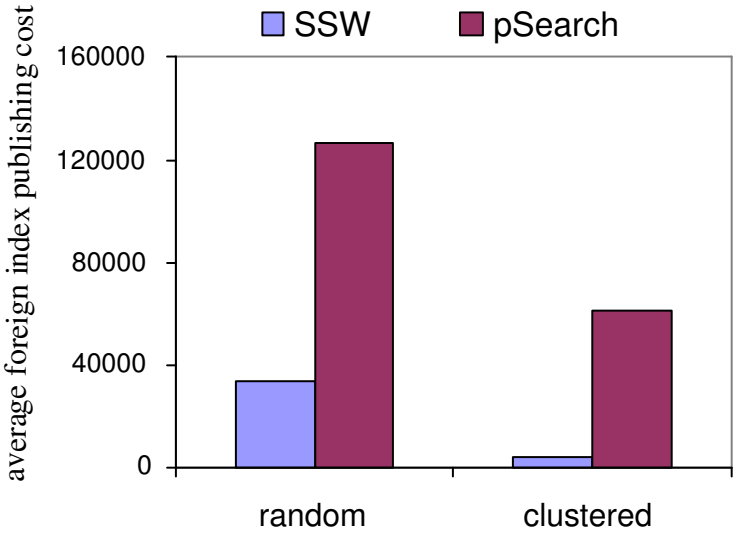
The effect of the cluster size on the maintenance cost, overlay navigation path length/cost (across clusters), and flooding search path length/cost (within clusters) are given in Figure 3.6. The cluster size is varied between 1 and 1024 (the whole network is one big cluster). In these graphs, the bars for pSearch (the last bar) is also given for easier comparison. As expected, the maintenance cost decreases when the cluster size increases (drops by 75% when the size increases from 1 to 4). The path length, though decreases slightly (because of the steeper drop in path length across clusters), is not as sensitive to the cluster size compared to the overall search cost (note that the third graph has y-axis in log scale). This is because the effect of the flooding within the cluster dominates for larger clusters. Within a spectrum of cluster sizes between 2 and 16, SSW does better than the size of 1 (whose results were presented in the previous section) in terms of all maintenance cost, path length and search cost. We have also conducted simulations by considering different mixes of the join/leave and search operations. Based on the results, we set the cluster size to 8 for the rest of the simulations.

3.1.7.3 Adaptivity to Data Distribution and Query Locality

In SSW, a peer selects the semantic centroid of its largest local data cluster as the join point (semantic position) when it joins the network. The rationale is that when data is more skewed around the centroid, fewer foreign indexes for data objects need to be published outside the cluster, thereby reducing the foreign index publishing cost. To



(a) synthetic data set



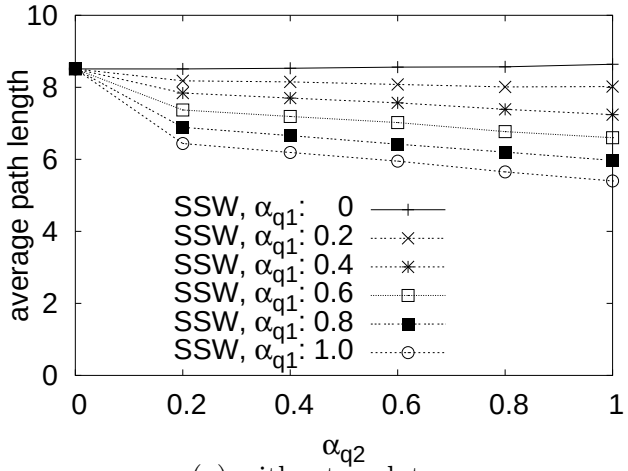
(b) real data set

Fig. 3.7. Effect of data distributions on SSW.

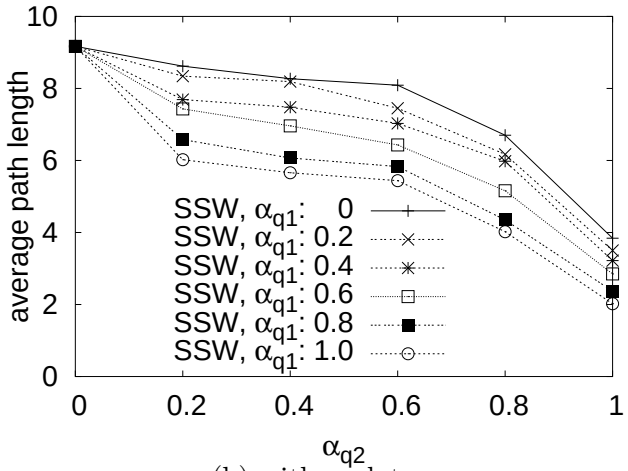
better understand the effect of semantic closeness on the foreign index publishing cost, we vary α_{d2}, the skewness for Data-Zipf, from 0 to 1. In addition, we also vary α_{d1}, the skewness for Dataseed-Zipf, from 0 to 1 to observe the effect of data hot spots. The results are shown in Figure 3.7(a). As pointed out, a higher skewness lowers the foreign index publishing cost of SSW significantly. pSearch's foreign index publishing costs are in the range of 3500 and only decrease slightly under skewed data distribution. We omit the plot for pSearch from this figure to avoid distorting the graph.

Figure 3.7(b) compares the foreign index publishing cost under real data set with random data distribution and clustered data distribution. Similar to the trends observed under synthetic data sets, if data objects are rather clustered instead of randomly distributed, SSW reduces the foreign index publishing cost significantly. pSearch's foreign index publishing cost is also reduced under clustered data distribution. However, the reduction is not as significant as in SSW.

In SSW, long range contacts can be updated based on query history to exploit query locality. To study this improvement, we vary α_{q2}, the skewness for Query-Zipf, from 0 to 1. We also vary α_{q1}, the skewness for Queryseed-Zipf, to observe the effect of query hot spots. Figure 3.8 compares the search path length of SSW (a) without updates and (b) with updates of long range contacts (based on what described in Section 3.1.5). Without any updates, query locality has little impacts on the results. With long range contact updates, however, query locality significantly enhances the search performance. For instance, we see nearly a 78% reduction in path length when α_{q2} increases from 0 to 1.0 with α_{q1} set to 1.0. pSearch's result (plot is not shown here) is similar to the one without update except that pSearch's path length is much higher (in the range of 40).



(a) without updates



(b) with updates

Fig. 3.8. Effect of query distributions on SSW. Cost for pSearch is much higher (not shown for clarity).

3.1.7.4 Tolerance to Peer Failures

Peer failure is a common event in P2P systems. Thus, a robust system needs to be resilient to these failures. To evaluate the tolerance of SSW to peer failures, a specified percentage of peers are made to fail after the network is built up. We then measure the ratio of searches that fail to find data objects existing in the network (we do not consider failures due to the data residing on the failed peers).

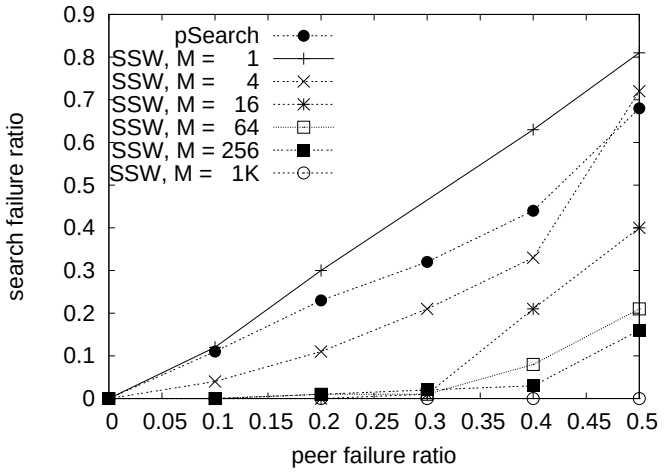


Fig. 3.9. Effect of peer failure ratio on the failure of search operations of SSW ($\gamma = 0\%$).

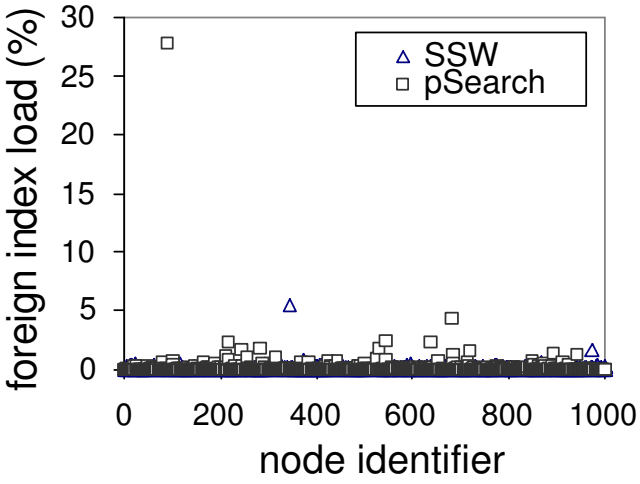
Figure 3.9 shows the fraction of searches that fail as a function of the ratio of induced peer failures (from 0% to 50%). Since the fault tolerance of SSW is largely dependent on the cluster size, we also consider different values for M (the cluster size) in these experiments. Even though each peer in pSearch maintains a large number of

states (20), the search failure ratio grows rapidly with the ratio of peer failures. On the other hand, at cluster size of 1, SSW with much smaller number of states maintained per peer (2 short range contacts and 4 long range contacts) has similar search failure rate as pSearch. Increasing the cluster size to 4 substantially improves SSW's fault tolerance. Beyond sizes of 4, the search failure ratio, even with as high as 30% peer failure, is very close to 0. These results reiterate the benefits of forming peer clusters.

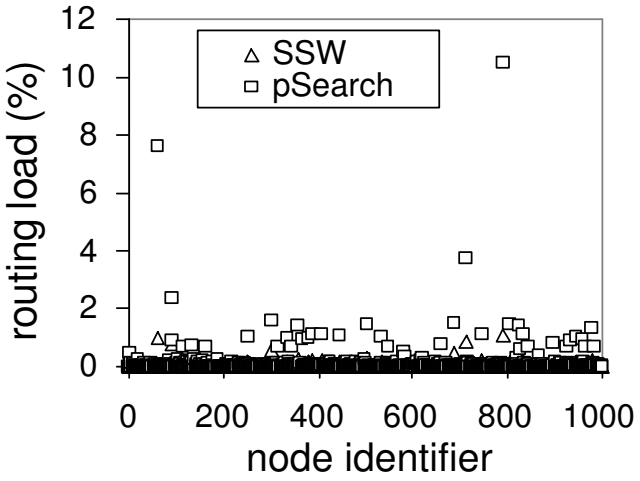
3.1.7.5 Load Balancing

We evaluate the load balance of SSW from two aspects: index load and routing load. For the index load, we evaluate the distribution of the foreign index maintained at each peer under different data distribution patterns. Since the load is evenly balanced under the uniform distribution for both pSearch and SSW, we present only the index load distribution for the skewed distribution in Figure 3.10(a). As we expected, pSearch has a much more uneven index load distribution compared to SSW (more rectangles with a higher load than triangles). In fact, Peer 87 in charge of a hot data region in pSearch stores about 28% of the index load of the whole system. In contrast, SSW displays a relatively even distribution of index load even under this skewed data set. This confirms our intuition that placing peers in the semantic space in accordance with their local data objects can effectively partition the search space according to data distribution.

Similarly, we have varied the query distribution in order to study the routing load distribution across the nodes, again with uniform and skewed distributions (this time with queries). We present the results for the skewed query distribution (α_{q1} and α_{q2} set to 1) in Figure 3.10(b). We find that the routing load is more evenly distributed in



(a) index load



(b) routing load

Fig. 3.10. Distribution of foreign index load and routing load among the peers in SSW.

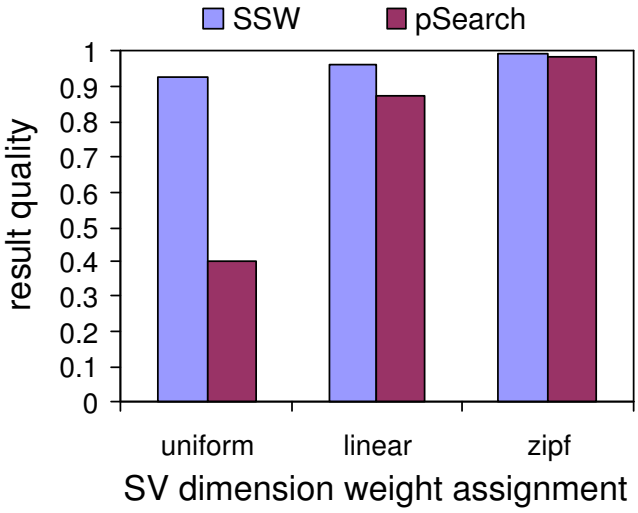
SSW compared to pSearch. This is due to the introduction of randomness through the long range contacts and the good balance within a peer cluster itself.

3.1.7.6 Result Quality

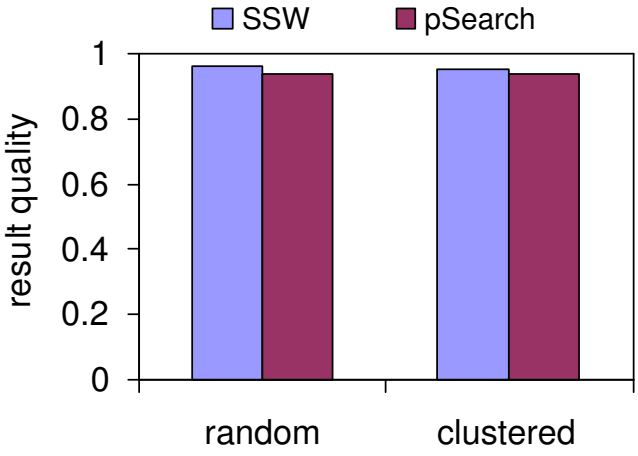
While the results on the search path length and search cost of point* query have been presented earlier, here we present the result quality of point* query under different data sets. Since the result quality highly depends on the SV dimension weight assignments, we vary SV dimension weight assignments and the results are presented in Figure 3.11(a). SSW partitions the data space in a way adaptive to data distribution in the semantic space, while pSearch partitions the data space uniformly (into two equally-sized subspaces). We expect that our partition scheme adapts to different SV dimension weight assignments automatically while pSearch does not. This is confirmed by Figure 3.11(a). Different SV dimension weight assignments have little effect on the result quality of SSW (and search path length, search cost, and maintenance cost as well) . On the other hand, pSearch is very sensitive to different dimension weight assignments. In all three settings, the result quality of SSW is higher than pSearch.

Figure 3.11(b) shows the result quality of point* query under real data set with random data distribution and clustered data distribution. From this figure, we can see that the result quality obtained under real data set is similar to that under synthetic data set with Zipf dimension weight assignment. This is not surprising since the vector elements in the document SVs have weights following Zipf-distribution.

In addition, we vary the dimension of semantic space (SVs) from 10 to 100 and evaluate its effect. The search path length is more or less the same under different



(a) synthetic data set



(b) real data set

Fig. 3.11. Result quality of point* query under different data sets.

dimensions (the results are omitted). This is not out of expectation since the underlying routing structure of SSW-1D is based on the linearized naming space, which is not tied to the data dimensionality. On the other hand, there are some changes on the result quality. Figure 3.12 shows the result quality of point* query under uniformly distributed synthetic data set (the trends observed under other data sets are similar and omitted). From this figure, we can see when the dimension of data objects increases, the result quality using SSW decreases by a very small amount. On the other hand, the result quality using pSearch is more sensitive to the dimensionality of data objects. This is expected since only partial subvectors are considered in pSearch, which degrades its result quality when the dimension is high.

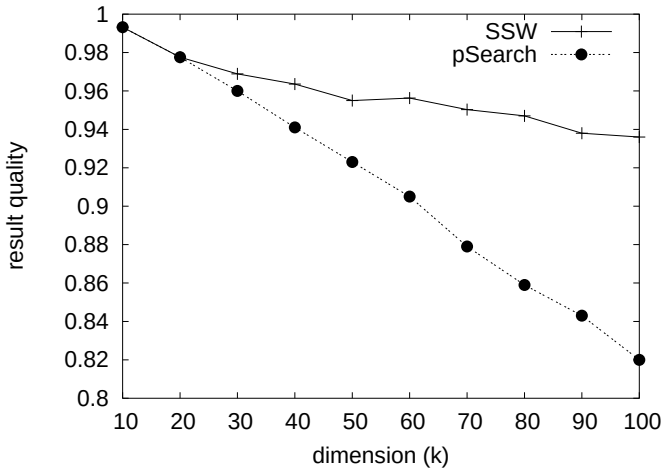


Fig. 3.12. Effect of dimensionality on point* query in SSW.

3.1.8 Summary

The voluminous information shared in peer-to-peer (P2P) systems mandates efficient semantic based search for data objects, which has not been adequately addressed in previous works. In this study, we present the design of a new P2P overlay network, semantic small world (SSW), that supports efficient semantic based search. SSW integrates four ideas, i.e., semantic clustering, dimension reduction, small world network, and efficient search algorithm, into an overlay structure with following desirable features. It facilitates efficient search without incurring high maintenance overhead. By placing and clustering peers in the semantic space based on the semantics of their data objects, SSW adapts to distribution of data automatically, gains high resilience to peer failures and balances index and routing load nicely. All of the above advantages of SSW are verified through extensive experiments.

There are some future works that can be explored in SSW. First, we can take advantage of the heterogeneity existing among peers to adjust the number of join points and contacts dynamically. Second, while SSW is designed in the context of P2P systems, whether similar structures can be deployed in the context of other LSDNs, such as ad hoc network, and sensor networks, is an potential issue to look into.

3.2 Processing Complex Queries

In this section, we present our study in range query and k nearest neighbors query (KNN) on high dimensional data objects in P2P systems. We study this problem in the framework of Semantic Small World (SSW), which is an efficient peer-to-peer

overlay developed for high dimensional data. Efficient query algorithms and solutions that address various technical challenges, such as multiple-destination query propagation, and KNN search space refinement, are proposed. An extensive simulation using both synthetic and real data sets demonstrates that our proposal efficiently supports range query and KNN query on high dimensional data in P2P systems.

3.2.1 Introduction

In Chapter 2.2, we review the relevant studies in query processing in P2P systems. These work mainly focus on low-dimensional data. In this study, we develop algorithms in support of range and KNN queries on high-dimensional data in dynamic P2P systems. Specifically, we develop our query processing algorithms based on semantic small world (SSW) (as presented in the previous section), a distributed index overlay designed for high dimensional data.

In SSW, a data object is treated as a point in a high-dimensional data space. This data space is dynamically partitioned into data subspaces, which are taken charge of by different peers. To deal with high dimensionality and dynamic membership changes, the data subspaces are linearized into a ring on which small world overlay is constructed. Note that in contrast to CAN [106], where each peer maintains pointers (routing information) for neighbors in all data dimensions, a peer in SSW only maintains pointers for a small number of peers which are nearby in the *linearized* space. While linearization is a rather common practice to address high dimensionality in general, we expect that SSW represents a generic class of P2P overlays supporting high dimensional data. Therefore, the challenges identified in this study are expected to be commonly faced by any other

P2P systems (e.g., [49]) that adopt space linearization to support high dimensional data, and the proposed solutions are expected to be generic enough to be applicable on these systems.

Several challenges are identified through our design of range and KNN query processing algorithms on high dimensional data. First of all, by the nature of high dimensional data and range/KNN queries, it is easy to observe that the searched data may be located in multiple peer nodes (possibly spread out in different data subspaces). Thus, one essential issues is *multi-destination query propagation* - efficiently routing and propagating a query to peer nodes which may have qualified data. The lack of complete neighborhood information in all data dimensions makes this very difficult to solve. We propose two efficient routing techniques, namely *single-path propagation (SPP)* and *multiple-path propagation (MPP)* to address this issue. Additionally, due to the non-deterministic search space of KNN queries, our KNN algorithm is based on an idea of *incremental search space refinement*, which assumes a large enough data space as initial search space and gradually prunes the search space as more data and/or index information are obtained. As the data dimensionality increases, the data space becomes very sparse and the distance to the nearest neighbors becomes large. Therefore, how to reduce KNN search space during query propagation and processing to avoid excessive examination of a large search space is critical for efficient processing of KNN query on high dimensional data. Three different KNN search space refinement strategies, namely, *minimum ID distance (MinIDD)*, *minimum mindist (MinMD)*, and *minimum centroid distance (MinCD)* are proposed. Our KNN algorithm is designed such that it can be

easily adapted to approximate KNN queries based on relaxed constraints on result quality.

While the proposed techniques are developed for query processing, they can be applied in a broad context. The proposed multi-destination query propagating methods can be applied for overlay multicast. The proposed KNN search space refinement methods and approximation techniques can be applied to address the issues related to query on high dimensional data (i.e., the well-recognized curse of dimensionality in the literature) in general.

To the best of our knowledge, this is the first study that provides an in-depth treatment of processing complex queries on high dimensional data in P2P systems. The contributions of this study are three-fold:

- Efficient algorithms for processing range and KNN queries on high dimensional data in a P2P system are proposed.
- A number of critical research issues raised due to high dimensionality in P2P systems, i.e., multi-destination query propagation and KNN search space refinement, are investigated. Solutions to address those issues are developed.
- An extensive performance evaluation using both synthetic and real data sets is conducted to validate our proposal. The result provides a number of insightful observations and shows that our algorithms efficiently support range and KNN queries on high dimensional data in P2P systems.

The rest of this section is organized as follows. The background is introduced in next section. The algorithms to process range query and KNN query as well as the solutions to relevant issues (i.e., multi-destination query propagation and KNN search space refinement) are given in Section 3.2.3 and Section 3.2.4, respectively. The experiment set up and results for the performance evaluation are detailed in Section 3.2.5 and 3.2.6. Finally, we summarize this study and point out the relevant issues that can be explored further in Section 3.2.7.

3.2.2 Research Formulation

Assume that there are N peer nodes in the system, where each peer node i has a dataset U_i $(i = 1, 2, ..., N)$ consisting of n_i data objects with k search attributes (i.e., dimensions). The union of the dataset U_i is U and the domain of U is $D \in R^k$, where R is the set of real numbers in the range of $\{0, 1\}$[5]. Each data object $x \in D$ is represented by a vector $x = \{x_1, x_2, ..., x_k\}$. Without loss of generality, we again assume the distance (dissimilarity) between two data objects, $d(x, y)$ where $x, y \in D$, is their Euclidean distance, i.e., $\sqrt{\sum_{j=1}^{k}(x_j - y_j)^2}$.

In this study, we focus on two representative multi-dimensional queries, i.e., range query and K nearest neighbors (KNN) query:

- **Range query:** given a query point $q \in D$ and real value $r \geq 0$, return the set of data objects $Q = \{o \in U | d(o, q) \leq r\}$.

[5]We can always normalize the attributes with domain not in the range of $\{0, 1\}$.

- **KNN query:** given a query point $q \in D$ and an integer $k \geq 1$, return the set of data objects Q with $|Q| = k$ and $Q = \{o \in U | d(o,q) \leq d(m,q)\}$ where $m \in (U - Q)$.

The goal of this study is *to develop efficient algorithms in support of range and KNN queries on high dimensional data in P2P systems.* Thus, we use **message overhead** and **query latency** to measure the performance of both range query and KNN query. In addition, we use **result quality** to measure the quality of the results returned by approximate KNN query. These three metrics are described in the following:

- **Message overhead:** the amortized number of messages incurred per query.
- **Query latency:** the time elapsed (measured in terms of number of hops, i.e., propagation steps, traversed) from the moment a query is issued till the query result is obtained.
- **Result quality:** $\frac{|Returned\ Result \bigcap Ideal\ Result|}{|Ideal\ Result|}$, where *Returned Result* and *Ideal Result* denote the returned result set and the ideal result set[6].

3.2.3 Range Query Processing

The query region of a range query may intersect with multiple subspaces, called as *candidate subspaces*, managed by different peer clusters (called as *candidate clusters*, accordingly). In other words, multiple peer nodes need to be searched in order to obtain the final answer set. An idea to answer range query is to greedily route a query, based on the shortest distance to the specified query region, towards a peer node located

[6]While there are various metrics to measure result quality in the literature, we choose the above metric since it is a practical metric to measure the result quality for approximate query in high dimensional space [56]

in a candidate subspace intersecting with the query region. Once the query message reaches such a subspace, in addition to being propagated to other peers located in the same cluster/subspace, the query message is propagated further (as needed) to other candidate subspaces.

In the following, we first describe a general algorithm for processing range queries and then present two strategies to propagate a query to multiple candidate subspaces.

3.2.3.1 Range Query Algorithm

Based on the above idea, we develop an algorithm for processing range query. Given that a peer issues or receives a range query, if its subspace intersects with the specified query region, the peer propagates the query message to all other members in its peer cluster. Meanwhile, by invoking SSR as described earlier, it determines whether there exist other subspaces intersecting with the remaining part of the query region and obtains their PCNs. Next, the peer forwards the query message greedily towards the candidate subspace(s) using a *multi-destination query propagation (MDQP)* algorithm (either sequentially using SPP or in parallel using MPP - to be discussed in Section 3.2.3.2). Algorithm 2 shows the pseudo code for range query processing in SSW.

Algorithm 2 Algorithm for Range Query.

Peer *i* issues range(q, r): $i.range(q, r)$

1: **if** $range(q, r)$ intersects with i.subspace **then**
2: Invoke search-within-cluster stage.
3: **end if**
4: Invoke SSR and obtain the PCN(s) for the candidate subspace(s).
5: Invoke MDQP and propagate range(q, r) to the candidate subspace(s).

3.2.3.2 Multi-Destination Query Propagation

If a peer maintains neighborhood information of all data dimensions as in CAN, MDQP can be solved easily by propagating the message to the neighbors recursively. However, as mentioned previously in Section 3.2.1, to address the high dimensionality and dynamic membership changes, in SSW's design, a peer does not maintain the complete neighborhood information of all data dimensions. Therefore, MDQP is a nontrivial issue. In this section, we present two different MDQP techniques, namely *single-path propagation* and *multiple-path propagation*, which propagate a query message to multiple subspaces sequentially or in parallel.

Single-Path Propagation. The first technique, called *single-path propagation (SPP)*, is to propagate the query message from query issuer to the candidate subspaces sequentially along a single path. The benefit of SPP is that the single-destination routing protocol (as described in Section 3.1.5) can be directly applied here to solve the multi-destination query propagation problem. However, one tricky issue is to decide in which order the candidate subspaces should be visited to minimize the message overhead as well as query latency. We observe that the routing path length between two subspaces is statistically proportional to their distances (d) in the one-dimensional ID space (see Theorem 3.2). To minimize the query latency, the optimal order to visit the candidate subspaces is the increasing order of their distances to the query issuer in the ID space (see Theorem 3.4), where the distance (called *ID distance*) basically is the difference between the ClusterID of the query issuer and the PCN of a candidate subspace.

THEOREM 3.4. *Given a one-dimensional SSW of N nodes and a set of m candidate subspaces to be visited, with their ID distances to the query issuer q as* $d(q,1), d(q,2), ..., d(q,m)$ *(*$d(q,1) < d(q,2) < ... < d(q,m)$*), the optimal latency using SPP is achieved when these m candidates are visited in the order of* $1, 2, ..., m$.

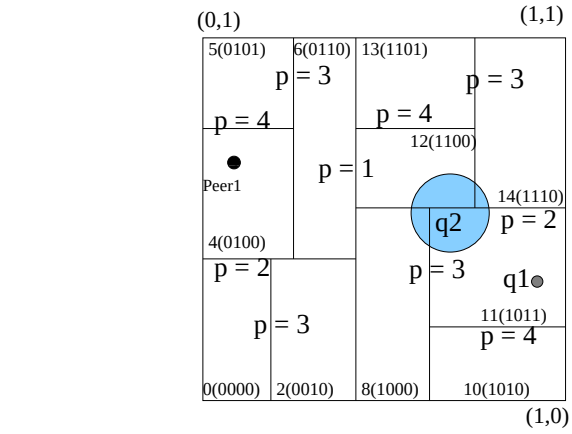
Proof: Before we prove the theorem, we first define some terms. Assume that the search path length between Cluster i and j ($i < j$) is $F(d(i,j))$, where $F(x)$ is positively proportional to x. For presentation clarity, we denote the subspace where query issuer resides as Candidate 0. We collectively call the routing hops to reach one of the m candidates as a *routing step*, with L_i denoting the search path length of the i^{th} routing step. Thus, propagating a message to all of m candidate subspaces incurs m routing steps. If the m candidates are visited in the order of $1, 2, ..., m$, $L_i = F(d(i-1,i))$ with $0 < i \leq m$. Therefore, the (optimal) total search path length, denoted as L, is obtained as $\sum_{i=1}^{m} L_i$.

We prove the above theorem by contradiction. Assume that when Candidate $(i+1)$ is visited immediately before Candidate i with $0 < i < m$, the incurred total search path length, denotes as L', is optimal. If Candidate $(i+1)$ is visited right before Candidate i, $L'_i = F(d(i-1,i+1))$, $L'_{i+2} = F(d(i,i+2))$ and $L'_j = L_j = F(d(j-1,j)$ with $0 < j \leq m$ and $j \neq i, j \neq (i+2)$. Since $d(i-1,i+1)$ is the sum of $d(i-1,i)$ and $d(i,i+1)$, which is greater than $d(i-1,i)$. Thus, $L'_i > L_i$. Similarly, $L'_{i+2} > L_{i+2}$. Therefore, $L' > L$. □

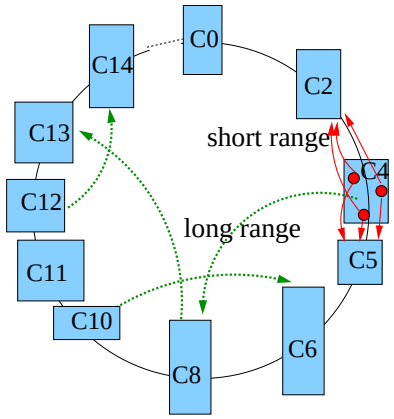
Multiple-Path Propagation. The query latency incurred by SPP may be high since the candidate subspaces are visited sequentially. Thus, we propose an alternative technique, called *multiple-path propagation (MPP)*, to propagate the query message in parallel along multiple paths. An important issue in MPP is to avoid redundant query message propagation and processing. Thus, a peer node, upon receiving a query message, needs to know which portion of the query region has been processed or to be processed by other peers. We observe that a PCN (e.g., a_h) corresponds to a GPT subtree rooted at the tree node with label as a and depth as h. This information can be utilized to propagate the query message systematically from the virtual tree downwards along different paths. In other words, each forked query message is attached with the PCN of a candidate subspace. The receivers of the query message will only process the portion of the query region that intersects with the subspaces indicated by the PCN. This process is repeated recursively until there is no more subtrees to be examined.

MPP is expected to incur lower query latency than SPP at the expense of increased message overhead. Nevertheless, MPP avoids redundant query message propagation. Hence, the increase of message overhead is expected to be low. The number of query messages and query latency in SPP and MPP is bounded by $O(xlog^2N)$ where x is the number of data subspaces intersecting with the query region.

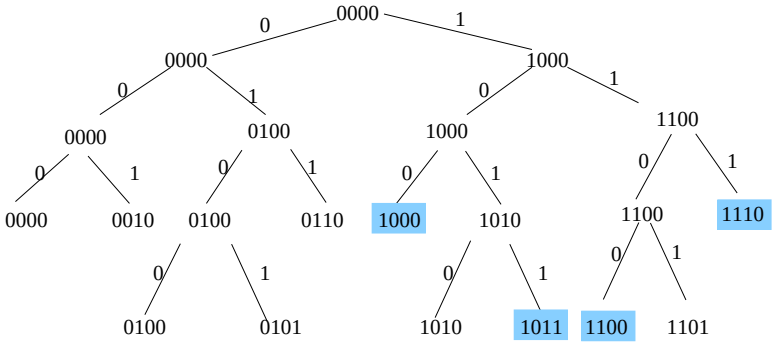
Figure 3.14 illustrates the difference between SPP and MPP on processing the range query depicted by the grey circle in Figure 3.13(a), issued by Peer 1 in Cluster 4. For readability, we repeat the GPT and structure of SSW-1D presented in previous chapter. In SPP, Peer 1 first obtains the PCN of the only candidate cluster/subspace (i.e., 8_1) by invoking SSR. As a result, the query is forwarded to Cluster 8 (please refer to



(a) search space



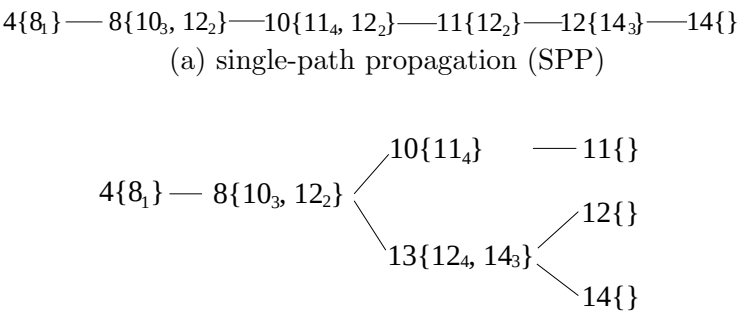
(b) SSW-1D



(c) GPT

Fig. 3.13. An illustrative example of SSW-1D.

Figure 3.13(b) for the long range contacts maintained at some peers). The peer receiving the message in Cluster 8 further obtains the PCNs for the query range via SSR as 10_3 and 12_2. Since the former is closer, the query message is forwarded to Cluster 10. This process continues and the message is subsequently forwarded to Cluster 11, 12 and 14. On the other hand, if based on MPP, Cluster 8 routes the query message along two paths in accordance with the obtained PCNs 10_3 and 12_2, respectively. The message corresponding to 10_3 reaches Cluster 10 and then is further forwarded to Cluster 11, while the message corresponding to 12_2 reaches Cluster 13 first and then is forwarded to Cluster 12 and 14 simultaneously. While SPP incurs a slightly lower number of messages (5 in SPP vs. 6 in MPP), MPP incurs lower query latency (3 hops in MPP vs. 5 hops in SPP).



(a) single-path propagation (SPP)

(b) Multi-path propagation (MPP)

Fig. 3.14. An illustrative example of multi-destination query propagation using SPP and MPP.

3.2.4 KNN Query Processing

To process a KNN query, the query message is first sent to a peer node (called *coordinator*) located in the cluster covering the query point and then propagated to all the peers within the cluster. Thus, the K data objects (within the cluster) located nearest to the query point can be obtained[7]. However, these K data objects may not be the correct (final) answer since some data objects located in neighbor clusters could actually be closer to the query point than those K data objects found so far. Thus, the distance of the current K^{th} nearest data object to the query point is used as the radius to obtain a multi-dimensional query region (sphere), which serves as the initial KNN search space. In order to avoid excessive examination of a large search space, the candidate clusters intersecting with the KNN search space are examined one by one and the KNN search space is refined (shrinks) incrementally whenever closer data objects are found. This process continues until the K nearest data objects are obtained.

In the following, we first describe a general algorithm for processing KNN queries and then discuss different strategies for determining the order to visit candidate clusters and incrementally refining the search space accordingly. Finally, we discuss how to employ our algorithm on approximate KNN query processing.

[7] Without loss of generality, we assume there are more than K data objects located in a cluster. If the cluster has less than K data objects, other nearby clusters can be visited to obtain K data objects.

3.2.4.1 Incremental KNN Query Algorithm

After receiving a query message, the coordinator first obtains K nearest data objects within its own cluster. The initial KNN search space is then obtained as a multi-dimensional query region (sphere) centered at the query point with the distance of the current K^{th} nearest data object to the query point as the radius. By invoking SSR, the PCNs of candidate clusters that intersect with the KNN search space are obtained. Based on different strategies (to be discussed in Section 3.2.4.2), we determine the next candidate cluster to search and proceed sequentially. In the new cluster, a new (and refined) set of K nearest data objects will be obtained and used to estimate a refined KNN search space. SSR is then invoked to obtain a new and refined list of candidate clusters. As such, the KNN search space continuously shrinks (whenever data objects nearer to the query point are found) and eventually converges. The query processing stops when there is no more unexamined clusters intersecting with the converged KNN search space. Algorithm 3 shows the pseudo code for KNN query processing in SSW.

3.2.4.2 KNN Search Space Refinement

Intuitively, the order to visit different candidate subspaces plays an important role in refining the KNN search space. Thus, we propose a number of strategies for the KNN algorithm to incrementally refine the search space.

Pessimistic Refinement: A pessimistic strategy assumes that all clusters/subspaces intersecting with the potential KNN search space are likely to contain the requested data objects. Thus, intuitively, a peer can randomly pick a candidate subspace to examine at one time, and refine the answer set and KNN search space accordingly. However, this

Algorithm 3 Algorithm for KNN Query.

Peer *i* issues knn(q, K): $i.knn(q, K, A)$ **(A records the** K **nearest neighbors encountered so far in the increasing order of their distances to** q**.)**

1: **if** $q \in i.subspace$ **then**
2: Invoke knn_range(q, K, A, r, c, Q).
3: **else**
4: Invoke SSR and obtain PCN for q as a_h.
5: forward $knn(q, K, A)$ towards a_h.
6: **end if**

Peer *i* issue knn_range(q, K, A, r, c, Q): $i.knn\_range(q, K, A, r, c, Q)$ **(**r **is the distance from the current** K^{th} **nearest neighbor to** q**.** c**, set to 0 initially, records the number of candidate subspaces examined so far.** Q **is a priority queue recording the candidate subspaces to be examined.)**

1: Process the query locally and initialize or update A.
2: $r' = d(A[K], q)$.
3: **if** $r' < r$ **then**
4: $r = r'$
5: Remove from Q these entries that do not intersect with the new query range(q, r).
6: **end if**
7: Invoke SSR and enqueue the obtained PCNs.
8: c = c+1.
9: **if** $Q = \emptyset$ **then**
10: stop.
11: **end if**
12: x = dequeue(Q).
13: Forward the query towards x.

may incur high message overhead as well as long query latency. Based on the same design principle for SPP of range algorithm, we propose one method to visit these subspaces by a more intelligent order in pessimistic strategy.

Minimum ID Distance (MinIDD). The candidate subspaces are visited sequentially according to their ID distances to the query point, i.e., the subspace with minimum ID distance to query point is visited first. Thus we call this method as *Minimum ID Distance (MinIDD).* During the process, whenever closer data objects are found, the KNN search space shrinks, and the candidate subspaces that do not intersect with the new KNN search space are removed from the candidate subspace set.

Optimistic Refinement: Pessimistic strategy is simple. However, the search space is expected to converge slowly using pessimistic strategy. Thus, an alternative strategy is to have an optimistic assumption that some candidate subspaces are "better" than others and contain closer data objects, and thus should be visited first such that the KNN search space will converge soon. Based on this assumption, we propose two optimistic methods for KNN search space refinements.

Minimum Mindist (MinMD). Generally speaking, the subspaces closer to the query point are likely to have more data objects close to the query point. Based on this observation, we propose to visit candidate subspaces according to their minimum distance (*mindist*, computed on the fly using a peer's LPT) to the query point, i.e., the subspace which has minimum *mindist* to the query point is visited first.

Minimum Centroid Distance (MinCD). If the data objects are randomly distributed within a data subspace, *mindist* is expected to be a good heuristic for estimating the likelihood of a subspace containing nearest data objects. However, MinMD may not work

well if the data is not uniformly distributed in the data space. In some pathological cases, a subspace with the smallest *mindist* to the query point might have all data objects residing in the corner furthest away from the query point, resulting in a very poor estimation.

To avoid the above problem associated with MinMD, some knowledge about distribution of data objects in candidate subspaces may help. Thus, we propose to use the centroid of data objects in a subspace to summarize the data distribution in a subspace. This information is very compact and thus can be exchanged among peers without incurring much overhead. As a result, the next candidate subspace to be visited is determined based on the distance from the centroids of candidate subspaces to the query point. We name this strategy as *Minimum Centroid Distance (MinCD)*.

To facilitate MinCD, we make the following extension on SSW in this paper. Each subspace caches state information of C subspaces (encountered during query processing) with closest centroids to its own centroid, including the feature vectors of their centroids, partition ranges, and the identifiers of some peers in these subspaces. The cached state information is updated during query processing when some new subspaces with closer centroids are found (due to peer join) or some subspaces maintained in the centroid cache become unreachable (due to peer leave).

Figure 3.15 shows an example for MinMD and MinCD where query region intersects with Cluster A, B, C and D and the query point resides in Cluster A. Majority of data objects (depicted by black dots) in Cluster D resides in the lower right corner far away from the query point, while majority of data objects in Cluster B and C resides in the corner close to the query point. In this case, the *mindist* of cluster D is smaller

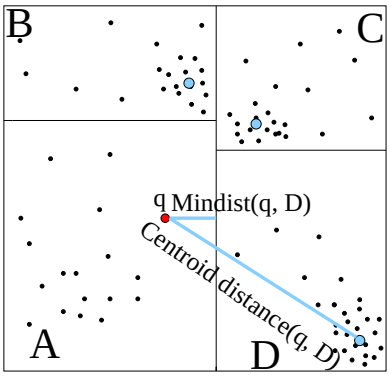


Fig. 3.15. An illustrative example of MinMD and MinCD.

than the *mindist* of Cluster B and C. However its centroid distance to the query point is much larger than the centroid distance of Cluster B and C. Thus, MinCD is a better heuristic than MinMD for refining search space under this situation.

Discussion. Among the three proposed methods, we expect that the KNN search space converges faster by using an optimistic strategy such as MinMD or MinCD. Between them, MinCD is expected to incur a smaller number of query messages than MinMD at the expense of caching some extra state information. On the other hand, although the KNN search space may shrink more rapidly using an optimistic strategy than using pessimistic strategy, the message overhead incurred by optimistic strategy is not necessarily lower than that incurred by pessimistic strategy (because the visiting ordering based on mindist or centroid distance is not exactly aligned to the order in the ID space and thus could result in some back and forth traversals).

3.2.4.3 Approximate KNN Query

Approximation, by trading off a slightly lower result quality, is frequently used to address the high cost of KNN query processing. These techniques more or less rely on certain types of global information, such as clusterings among data objects [78], random samples [56], sample dimensions [51], or error bound [7]. The large scale and dynamics of P2P systems make these query algorithms and approximation techniques unsuitable. While the higher levels of the tree-structured indexes can easily become the performance bottlenecks and single points of failure, obtaining and maintaining the global information for the above approximation techniques in P2P systems is very expensive.

Our KNN query algorithms can be naturally adopted to facilitate approximate query processing. The basic idea is to examine only the first t, a threshold value predefined either by the system or by the users, candidate subspaces in accordance with the three strategies proposed for search space refinement.

3.2.5 Simulation Setup

The simulation is set up similarly as in the performance evaluation for SSW (Chapter 3.1.6). We also adopt the data sets used in the evaluation for SSW. In the following, we explain the system parameter settings and query workload.

System Parameters: The number of long range contacts and the size of peer clusters are set to 4 and 8, respectively (similar to the settings used in Chapter 3.1.6). The number of cached centroids (C) for MinCD is set to 10. We also vary C in one set of the experiments to evaluate its impact on the performance of MinCD.

Workload Parameters: A range query is specified by two parameters, query radius and query point, and a KNN query is specified by two parameters as well, the value of K and query point. The query radius for range query are randomly drawn from the range $[0, m]$ where m, whose default value is set to 0.5, indicates the maximum query radius. The K value for KNN query is set to 10. The query points for both range query and KNN query are randomly drawn from the data space.

3.2.6 Simulation Results

We have conducted an extensive simulation to evaluate the performance of our proposed algorithms for range query and KNN query[8].

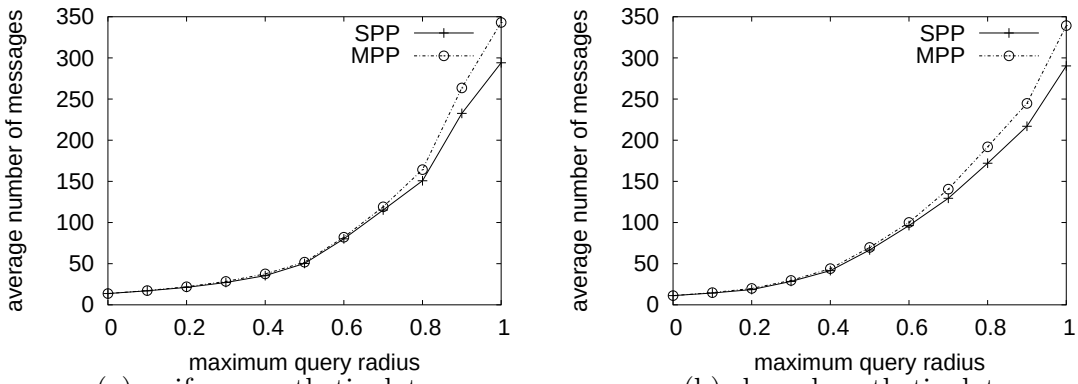
In this section, we first present the simulation results for range query, followed by the simulation results for KNN query. We conduct all sets of simulations under different data sets as explained above, and the general trends observed under different data sets are consistent. In the following, we present the results under different data sets when necessary. Otherwise, for the interest of readers and presentation clarity, we only present simulation results under one data set.

3.2.6.1 Range Query

In the following, we present the effect of *query radius* and *data dimensionality* on the performance of range query.

Effect of Query Radius. We vary the maximum query radius m from 0 to 1 and evaluate the performance of range query under different data sets. Figure 3.16 shows

[8]To the best of our knowledge, there is no comparable existing work in the literature as discussed in 2. Thus, we compare the performance of different strategies proposed in this paper.



(a) uniform synthetic data (b) skewed synthetic data

(c) uniform real data (d) skewed real data

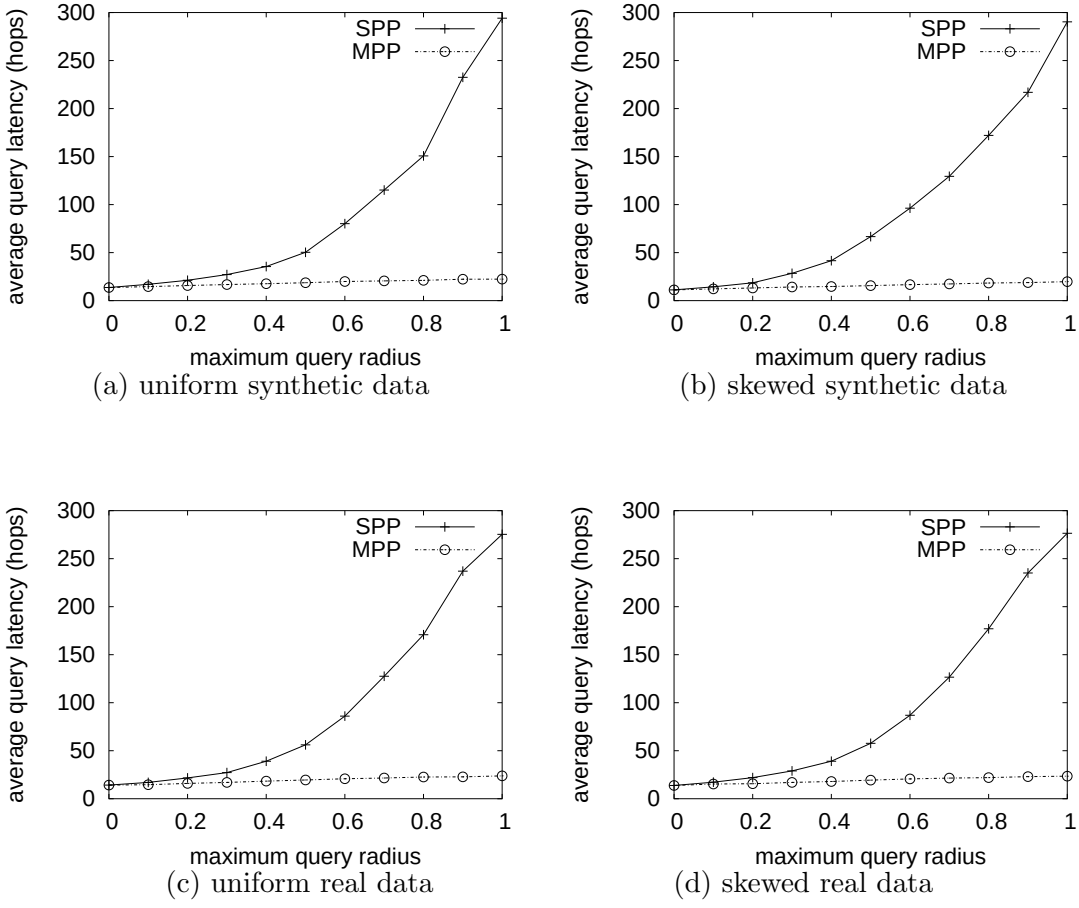
Fig. 3.16. Maximum query radius vs. message overhead (range query).

message overheads under (a) uniformly distributed synthetic data (the skewness parameters set to 0), (b) skewly distributed synthetic data (the skewness parameters set to 1), (c) uniformly distributed real data and (d) skewly distributed real data. From this figure, we can see that the message overhead under MPP is slightly higher than the one under SPP when the query radius is large.

Figure 3.17 shows the query latency incurred by SPP and MPP. From this figure, we can see that when the query radius is small, the difference in the query latency incurred by SPP and MPP is very small. However, when the query radius increases, the query latency incurred by MPP is much smaller than the query latency incurred by SPP. This confirms our expectation (discussed in Section 3.2.3.2) that lower query latency is incurred when query message is propagated to multiple destinations in parallel.

Combining these two sets of results, we can conclude that under small query radius, SPP and MPP have similar performance in terms of both message overhead and query latency, while under large query radius MPP incurs a slightly higher message overhead but much lower query latency compared to SPP. In addition, we observe that the simulation results under the four different data sets are very similar to each other.

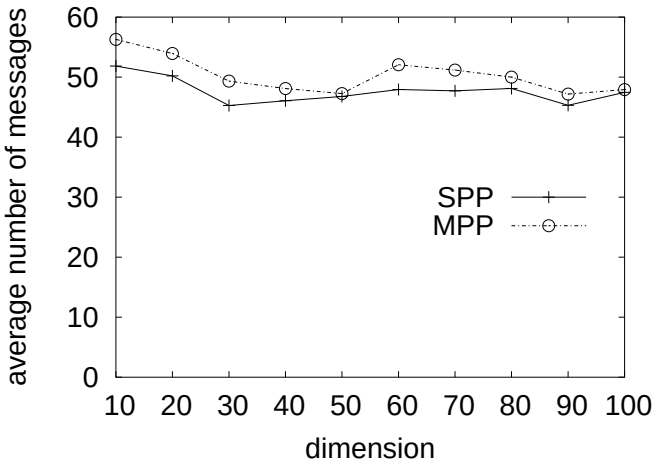
Effect of Data Dimensionality. We vary the dimensions of data objects and observe its effect on message overhead and query latency for range query. Figure 3.18 shows the results for SPP and MPP under uniformly distributed synthetic data (the results under other data set are similar) and it demonstrates that the message overheads and query latency are more or less the same when the dimension is varied from 10 to 100. This is not out of expectation since the underlying routing structure is based on a linearized ID space, which is not tied to the data dimensionality. The figure also shows that with a



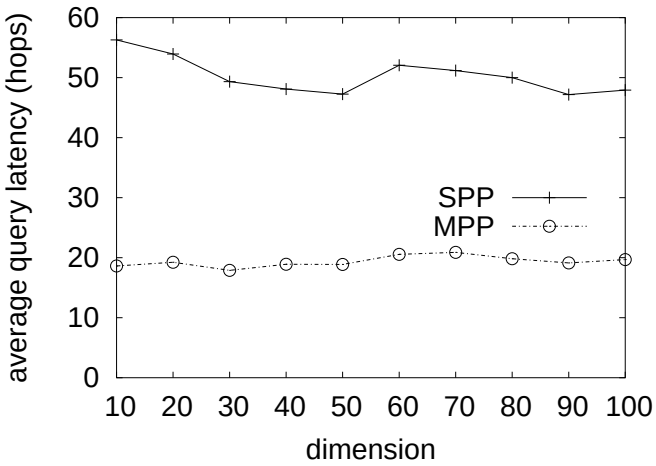
(a) uniform synthetic data (b) skewed synthetic data

(c) uniform real data (d) skewed real data

Fig. 3.17. Maximum query radius vs. query latency (range query).



(a) message overheads



(b) query latency

Fig. 3.18. Effect of dimensionality under uniformly distributed synthetic data (range query).

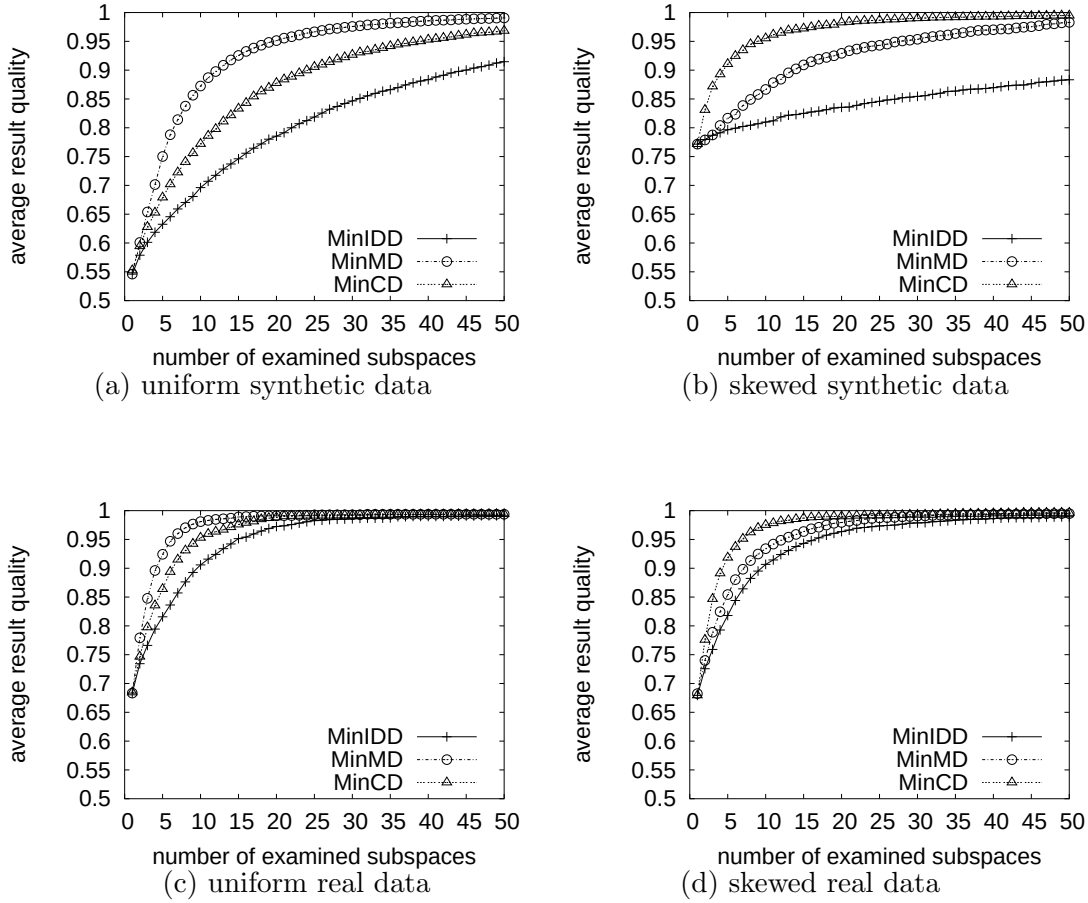
small amount of extra message overhead, the MPP significantly outperforms the SPP in query latency.

3.2.6.2 KNN Query

In this section, we first present the incremental changes of result quality and message overhead during the query process using the three KNN search space refinement methods. We then examine the message overheads incurred by the three methods to reach the same level of result quality. Lastly, we present the effect of dimensionality. Since the candidate subspaces are visited sequentially in our proposed KNN query algorithm, the query latency is proportional to message overheads. Thus, in the following, we only present the plots on message overheads and result quality.

Incremental Changes of Result Quality and Message Overhead. We first examine the performance of our proposals during query processing (i,e., when more and more subspaces are visited). This experiment also evaluates the performance of our proposed methods under various threshold values of t for approximate KNN queries.

Figure 3.19 shows the progress of result quality. We only show the result for the first 50 examined subspaces for readability. From this figure, we have the following three observations. First of all, the result quality under optimistic refinement methods (MinMD and MinCD) climbs faster than under pessimistic refinement method (MinIDD). This confirms our expectation since MinMD and MinCD choose to first visit the "promising" subspaces which are likely to contain qualified data objects (thus result in quick increase of result quality), while MinIDD does not make this optimization. Secondly, under skewly distributed data set ((b) and (d)), the result quality using MinCD climbs



(a) uniform synthetic data (b) skewed synthetic data

(c) uniform real data (d) skewed real data

Fig. 3.19. Number of examined subspaces vs. result quality (KNN query).

faster than the one using MinMD, while under uniformly distributed data set ((a) and (c)), the result quality using MinMD climbs faster than the one using MinCD. This again confirms our discussion in Section 3.2.4.2. Under skewly distributed data set, the distance between the query point and the subspace centroid is a better heuristic than mindist for estimating the likelihood of a subspace containing data objects closer to the query point. Lastly, under all four data sets, the result quality for the optimistic refinement methods (MinMD and MinCD) initially increases quickly with the number of examined subspaces, followed by a much slower increase. This suggests that the proposed refinement methods can be utilized for approximate KNN query effectively.

Figure 3.20 shows the message overhead during query processing. We have two observations from this figure. First of all, when only a few subspaces are examined, the number of messages incurred by MinCD is a little bit higher than the ones incurred by MinMD or MinIDD. When more subspaces are examined, the number of messages incurred by MinCD increases much slower than the ones incurred by MinMD or MinIDD, thus quickly falls below the plots for MinMD or MinIDD. The reason is exactly as explained in Section 3.2.4.2. Compared to MinMD or MinIDD, MinCD maintains some extra state information, i.e, the cached centroids. On one hand, these extra states incur maintenance overheads. On the other hand, they provide direct contacts to the relevant subspaces, thus propagating to these subspaces is quite lightweighted. When only a few subspaces are visited, the maintenance overheads for MinCD dominate the message overheads. However, when more subspaces are visited, the routing benefit weighs more and puts MinCD on the winning side of message overheads. Our second observation

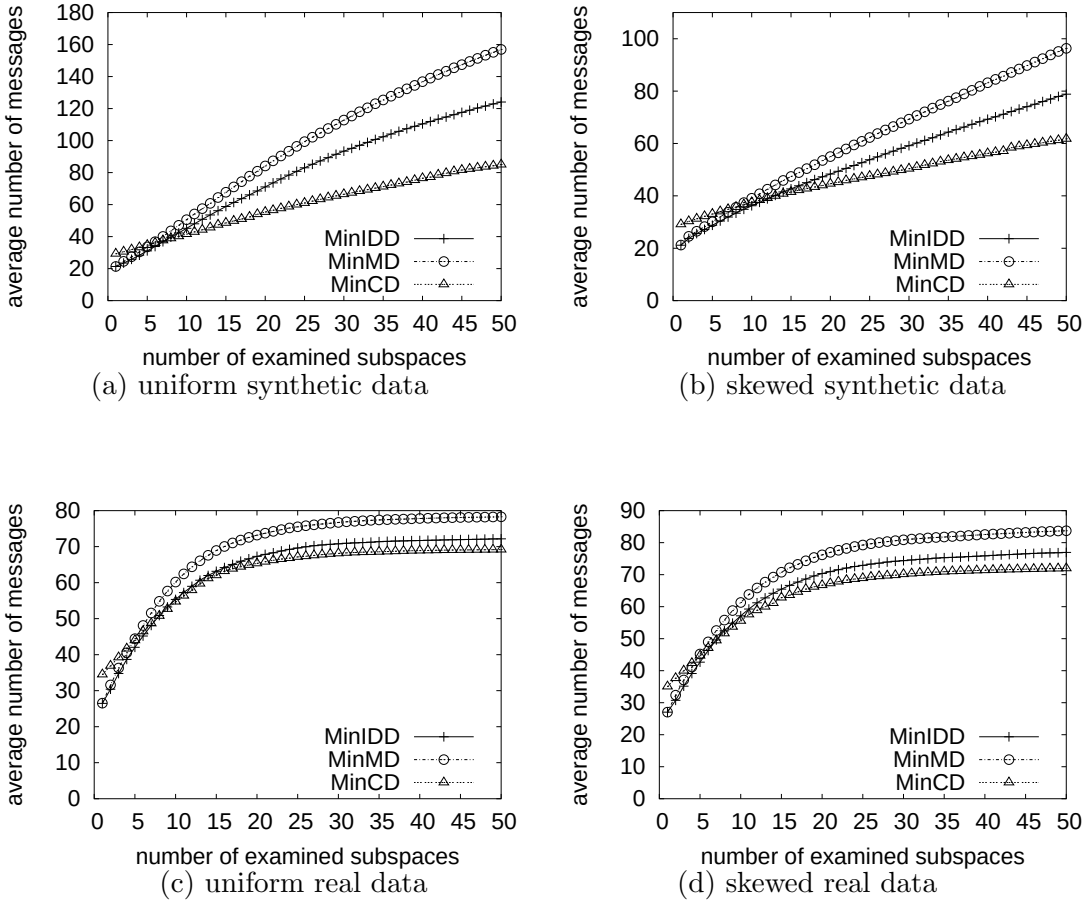


Fig. 3.20. Number of examined subspaces vs. message overhead (KNN query).

is that to visit the same number of candidate subspaces, MinMD always incurs higher message overheads than MinIDD. This confirms Theorem 3.4.

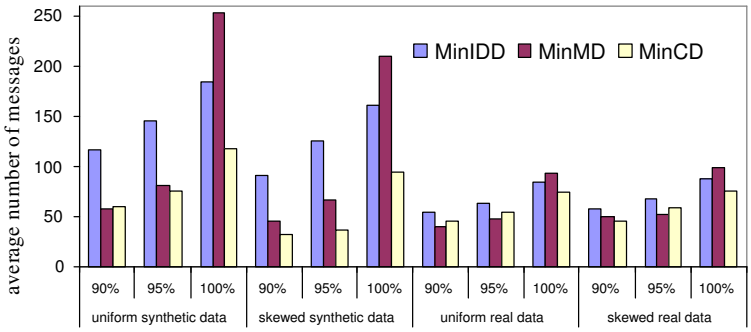


Fig. 3.21. The number of messages incurred to reach 90%, 95% and 100% result quality under skewly distributed synthetic data (KNN query).

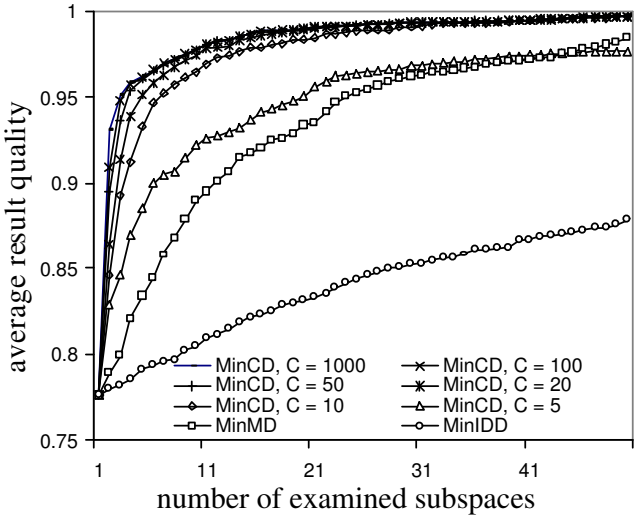
Message Overheads vs. Result Quality. Having observed the incremental changes of result quality and message overheads using the three refinement methods during query processing, we now examine the actual number of messages incurred by the three methods to reach certain level of result quality. Figure 3.21 shows the number of messages incurred per query to reach 90%, 95% and 100% result quality under skewly distributed synthetic data (the results under other data set are similar). From this figure, we can see that in most cases, MinCD incurs the lowest message overhead. We also observe that MinMD incurs lower number of messages than MinIDD to reach result quality 90% and 95%. However, to reach 100% result quality, the message overhead incurred by MinMD is

a little bit higher than MinIDD. This is not surprising since MinIDD optimizes the propagation (i.e., routing) of query messages to a group of destinations. Though MinIDD might need to examine more subspaces to reach the same level of result quality as MinMD or MinCD, the number of messages incurred by MinIDD is not necessarily higher.

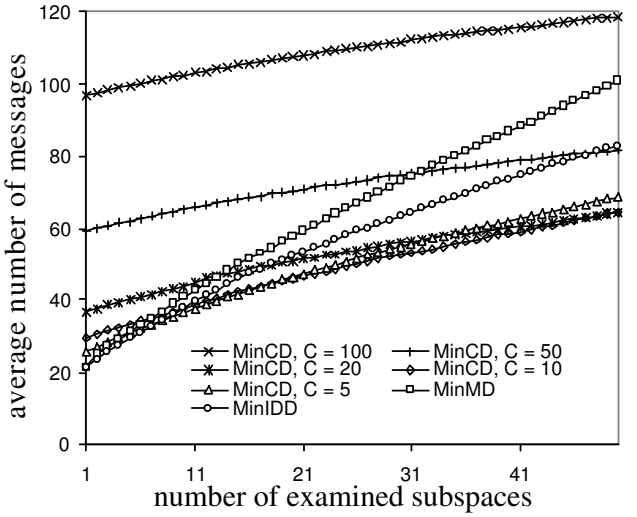
In summary, comparing the three KNN search space refinement methods, MinCD has the best performance in most tested cases. Between MinMD and MinIDD, MinMD is better for approximate KNN query (with a result quality smaller than 100%) while MinIDD is better for exact KNN query (i.e., result quality equals 100%).

Effect of Number of Cached Centroids on MinCD. Up to now, the number of cached centroids, C, for MinCD method is fixed at 10. In this set of experiments, we vary C from 5 to the maximum value 1000 (the network size). Figure 3.22 shows the result quality and number of messages with different sizes of centroid caches under skewly distributed synthetic data. We also re-display the plots for MinMD and MinIDD for comparison in this figure. From Figure 3.22(a), we observe that the climbing speed of result quality doesn't decrease much when C varies from 1000 to 10. Yet we observe a significant difference when C decreases from 10 to 5. At this point, the result quality for MinCD is close to the one for MinMD. From Figure 3.22(b), we see that the number of messages incurred by MinCD decreases when C decreases. This is obvious since less messages are incurred for maintaining less number of centroid information.

Effect of Data Dimensionality. In previous experiments, we fix the dimensionality of data objects as 10. In this set of experiment, we vary the number of dimensions from 10 to 100 and evaluate the effect on the three methods. We observe that the number of messages incurred to visit the same number of subspaces does not change

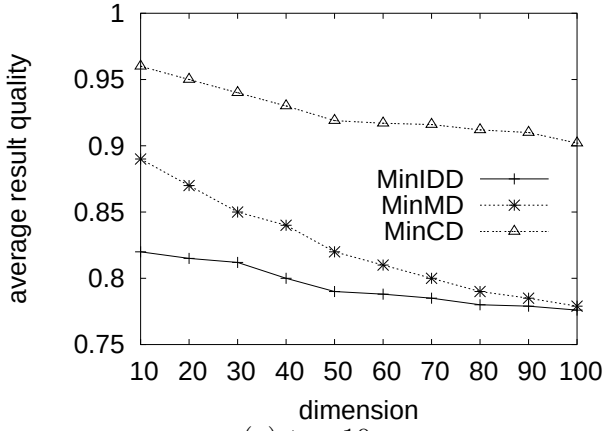


(a) result quality

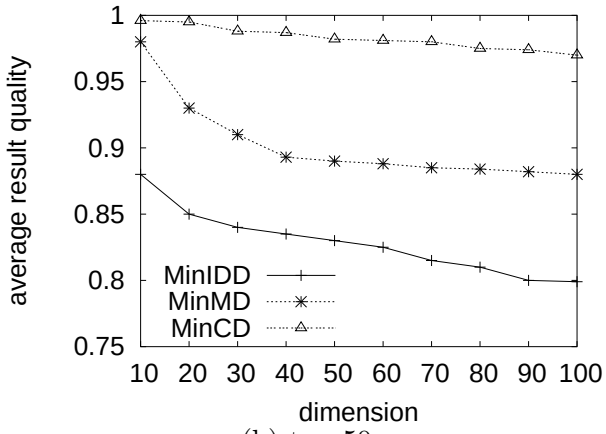


(b) number of messages

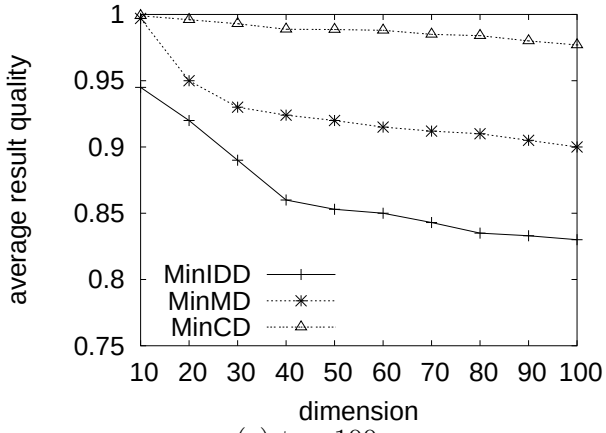
Fig. 3.22. Effect of number of cached centroid on MinCD under skewly distributed synthetic data (KNN query).



(a) t = 10



(b) t = 50



(c) t = 100

Fig. 3.23. Effect of dimensionality under skewly distributed synthetic data (KNN query).

significantly with the increase of dimensionality. This is expected since the underlying 1-dimensional routing structure is not tied to the data dimensionality. The plots on number of messages are not shown here due to space constraint. Figure 3.23(a–c) shows the result quality under skewly distributed synthetic data (the results under other data set are similar) when the first 10, 50, and 100 subspaces are examined (i.e., t = 10, 50, 100), respectively. From this figure, we can see with the increase of dimensionality, the result quality decreases. However, the decrease is not dramatic, especially for MinCD. For instance, the result quality using MinCD decreases around 5%, 3% and 2% when t = 10, 50, and 100, respectively. This indicates our KNN (and approximate KNN) query algorithm works well under high dimensionality.

3.2.7 Summary

While more and more applications are starting to leverage P2P technology for sharing and exchanging resources among numerous users, efficient algorithms in support of complex queries are needed. In this study, we investigate range and KNN queries on high dimensional data in P2P systems.

The high dimensionality of data objects raises many challenging issues in range and KNN query processing, such as multi-destination query propagation and KNN search space refinement. We propose two routing methods, namely, SPP (single-path propagation) and MPP (multiple-path propagation), for multi-destination query propagation. To tackle the challenge of KNN search space refinement, we develop three schemes, namely MinIDD (minimum ID distance), MinMD (minimum mindist) and MinCD (minimum centroid distance), to incrementally refine the KNN search space. These schemes can be

easily adopted for approximate KNN queries. We conduct an extensive simulation using both synthetic and real data set to evaluate our proposal. The simulation results show that our proposal efficiently supports range and KNN query on high dimensional data in P2P systems.

We are exploring other types of complex queries, such as Top-K query and joins, in P2P systems. In addition, we plan to extend the proposed ideas to other overlay structures.

Chapter 4

Managing Multi-Dimensional Data Objects

While SSW efficiently supports high dimensional data objects, the design principle of SSW is based on space partition based indexes. As discussed in Chapter 2.1.1, space partition based indexes is better suited for high dimensional data objects. Nevertheless, when the dimensionality is not high (in the range of tens), object grouping based indexes are better compared to space partitioned based indexes.

In this study, we propose a framework, called *distributed peer tree (DPTree)*, which, based on object grouped based indexes (balanced tree indexes), efficiently supports various types of queries on multi-dimensional data in P2P systems with reasonable maintenance overheads. In addition, DPTree adapts to data distribution and access pattern by effectively balancing the access load among peers. DPTree achieves the efficiency and effectiveness through the following designs: 1) distributing the tree structure among peers in a way preserving the nice properties of balanced tree structures yet avoiding single points of failure and performance bottlenecks; 2) organizing peers into an overlay structure that enables efficient navigation yet is easy to maintain; 3) an efficient navigation algorithm; 4) an innovative wavelet-based load balancing mechanism. Through extensive performance evaluation, we verify the superiority of DPTree over existing works on various aspects, including load balancing, routing, query processing, and maintenance.

4.1 Introduction

Many applications deployed or to be deployed on P2P systems involve data objects with a small to moderate number of attributes, which can be viewed as points in a multi-dimensional space. For instance, mapping or location service involve data objects with two attributes (longitude and latitude). Grid information service involves data objects with multiple attributes including types of operating systems, CPU speed, network address, storage capacity, and etc. Similarly, relational data objects have multiple attributes, such as product ID, quantity, price, and manufacturer.

In this study, we propose an indexing framework, called distributed peer tree (DPTree), which efficiently supports various types of queries on multi-dimensional data in P2P systems at reasonable maintenance overheads. In addition, DPTree automatically adapts to data distribution and access pattern. DPTree is inspired by balanced tree indexes (R-tree [54] and a series of variants), which have been intensively studied and become well-accepted multi-dimensional index structures in database community over the years. They possess some nice features including the ability to efficiently support a rich set of queries and adaptivity to data distribution. However, it is challenging to support balanced tree indexes in P2P systems. Simply mapping each tree node to a peer would result in performance bottlenecks and single points of failure at the peers taking charge of the tree nodes at higher level. In addition, coupling a tree node with a peer (and coupling the tree data structure with the overlay structure) would make the maintenance complex and costly. DPTree overcomes this challenge by decoupling a tree

node from a peer (and decoupling the tree data structure from the overlay structure) and assigning (replicating) tree nodes to peers in accordance with their access frequencies.

While the above basic idea of DPTree is straightforward, the following issues need to be addressed carefully.

- *Tree distribution*: how tree nodes are assigned (replicated) among peers in accordance with their access frequencies. This is crucial to the search performance and maintenance overheads.

- *Overlay structure*: how peers form into an overlay that facilitates efficient navigation on the tree yet is easy to maintain.

- *Navigation*: how a query request is propagated to the destination. With each peer only having a partial view of the tree structure, navigation in DPTree is nontrivial.

- *Access load balancing*: how the access load is fairly distributed among peers. Majority of existing works attempt to achieve fair distribution of storage load in the system through random hashing. However, in order to efficiently support complex queries, data objects in DPTree are organized in accordance with their attribute values rather than randomly hashed values. In addition, accesses to different data objects are not uniformly distributed, and fair distribution of storage load does not imply fair distribution of access load. Therefore, we need to design an explicit access load balancing mechanism in DPTree.

To address these issues, we propose a suite of efficient solutions, which constitute the following four major contributions of this study.

1. We propose *tree branch oriented distribution*, which distributes (and replicates) *tree branches*, i.e., the tree nodes along the path from the root to leaf nodes, as atomic units to different peers. This scheme correlates the number of replicas for a tree node to its access frequency, and thus it does not incur single points of failure and performance bottlenecks.

2. We propose a *tree-aware overlay*, which maps peers to an ID space in accordance with their assigned tree nodes, and then organizes peers into an overlay that is insensitive to peer distribution in the ID space based on the design principle of skip graph [14]. This overlay structure enables efficient tree navigation yet is easy to maintain.

3. We propose *aggressive navigation* algorithm, through which peers aggressively forward messages to the destinations in $logN$ steps on DPTree with high probability even though each peer only has a partial view of the tree structure (N is the network size).

4. We propose *wavelet-based load balancing* mechanism, which leverages wavelet to monitor the load distribution in the system and adjust the load among peers in a light-weighted yet effective fashion.

As a proof of concept, we show how different types of queries, i.e., point query, range query and KNN query, can be easily and efficiently supported in DPTree. Through extensive performance evaluation, we demonstrate the superiority of DPTree over existing works on various aspects, including load balancing, routing, query processing, and maintenance.

The rest of this chapter is organized as follows. We provide the background and system model in next section. The design details of DPTree are given in Section 4.3, and the algorithms to process different queries are described in Section 4.4. The performance evaluation is presented in Section 4.5. Finally, we summarize this study and point out the relevant issues that can be explored further in Section 4.6.

4.2 Preliminaries

4.2.1 Background

We briefly introduce balanced tree indexes, skip graph, and wavelet, which are necessary to understand DPTree.

4.2.1.1 Balanced Tree Index

Balanced tree indexes are hierarchical structures that recursively decompose a set of data objects into f (called *fanout*) subgroups (tree nodes). The decomposition stops when the number of data objects in a subgroup falls below a threshold value c (called *leaf node capacity*). The top level (coarsest) subgroup is the root node and the finest subgroups are the leaf nodes (other subgroups are called *non-leaf nodes*). A leaf node stores the *coverage* (enclosing region) of its corresponding subgroup and the storage addresses of the data objects in the subgroup. A non-leaf node stores the coverage of itself and its immediate children.

We use R-tree [54], a well-known balanced tree index, as an illustrative example. The coverage of a tree node in the R-tree is represented by the smallest rectangle enclosing all the data objects in the subgroup, called *minimum bounding rectangle (MBR)*.

Figure 4.1 shows an example. The left side depicts the MBRs and the right side depicts the tree structure. Other balanced tree indexes have similar structures with different representation for the coverage of tree nodes.

Fig. 4.1. An illustrative example of R-tree.

To process a query in these index structures, the coverage of a tree node is examined against the query starting from the root node. If the coverage of a tree node does not enclose the query, the subtree rooted at this tree node does not need to be examined, i.e, the subtree can be *pruned*. Otherwise, the children's coverage is examined, and the query is continued at the child (or children) whose coverage encloses the query. The process continues till every part of the tree is either pruned or examined.

Balanced tree indexes have the following nice features. 1) Different from hashing techniques, they preserve locality among data objects, which is essential for efficient processing of complex queries. 2) They are fully dynamic and automatically adapt to data distribution. 3) As pointed out in Section 2.1.1, they do not need to index dead

space (the data space that is not populated by data objects), which is required in other overlays (e.g., CAN). 4) Mature algorithms for various types of queries based on these index structures have been developed over the years.

4.2.1.2 Skip Graph

Different from other overlays that establish overlay links based on the distance in an ID space, skip graph (or skipnet) establishes overlay links based on *peer distance* (the number of peers between two peers) [14, 55]. Therefore, skip graph is insensitive to the distribution of peers in the ID space.

In skip graph, peers first form into a doubly-linked ring. In addition, a peer maintains ($logN$-1) pairs of skip pointers (neighbors), each pair of which skip over 2^i peers clockwise and counterclockwise with high probability ($1 \leq i \leq logN$-1). Conceptually, skip graph is a hierarchical ring with $logN$ levels. The level-0 ring consists of all peers. It is split into two *child rings* (the ring before splitting is called *parent ring* accordingly), which are then recursively split until the number of peers in the ring is not greater than 2. A peer joins one ring at each level.

We first explain how to form a "perfect" skip graph, where a peer's neighbor at level-i of the overlay is at exactly 2^i peer distance away. Starting from level-0, the peers in a parent ring are alternatively assigned to one of the two child rings. Figure 4.2 illustrates an example of a four-level skip graph formed by 16 peers. Peer 1, 3, 5, 7, 9, 11, 13, and 15 form one child ring at level-1 and the rest of the peers form the other child ring at level-1. Each peer has four pairs of neighbors with one pair at each level.

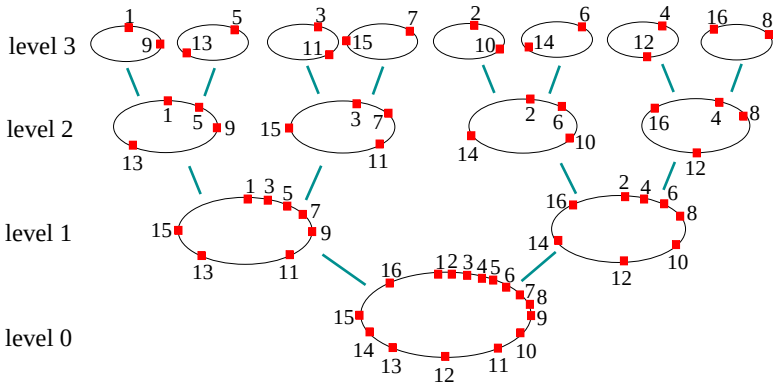


Fig. 4.2. An illustrative example of skip graph.

The above overlay construction is too rigid to accommodate dynamic peer join, leave or failure. Therefore, some randomness is introduced to make the overlay flexible by allowing a peer to randomly join one of the two child rings. With high probability, a peer's neighbor at level-i is at 2^i peer distance.

Routing is performed level-by-level starting from the top level. At a specific level, a message is always forwarded to the neighbor that is closest to the destination without overshooting it. If no such neighbor can be found, routing descends one level and the above process repeats. It is proven that routing can be resolved in $logN$ steps with high probability in skip graph [14].

4.2.1.3 Wavelet

Wavelet is a tool used extensively in signal processing (for details, please see [118]). It provides views of data at different resolutions, called as *levels of decomposition*. In the following, we introduce the basic concept of Harr Wavelet, which is simple and fast to

compute. Harr Wavelet consists of *average coefficient* (or *average*) and *detail coefficients* (or *differences*) of a signal. The average coefficients and detail coefficients at a level of decomposition are obtained by pairwise averaging and differing of the averages on the previous level of decomposition. There are total $\lceil logn \rceil$ levels of decomposition in the wavelet transform for a signal sequence with n items. Suppose we have a set of items with their values as $\{a_1, a_2, ..., a_n\}$. Denoting the j^{th} average and difference at level i of decomposition as $s_{i,j}$ and $d_{i,j}$, respectively,

$$s_{i,j} = \frac{s_{i-1,2j} + s_{i-1,2j+1}}{2 \cdot m_j}$$

and

$$d_{i,j} = \frac{s_{i-1,2j+1} - s_{i-1,2j}}{2 \cdot m_j}$$

where m_j is the normalization factor for level j of decomposition. In Harr wavelet, $m_j = \sqrt{2}^{\lceil logn \rceil - j}$. The *wavelet transform* is defined as the average coefficient at the top level of decomposition, followed by the detail coefficients at increasing resolution (decreasing levels of decomposition). Each of the individual coefficient is called *wavelet coefficient*. In the remaining discussion, we refer wavelet transform as wavelet when the context is clear.

We use a simple sequence consisting of {4, 2, 2, 2, 15, 11, 1, 11} to illustrate how Harr Wavelet is formed. The averages and differences at level-1 decomposition are obtained as {3, 2, 13, 6} and {-1, 0, -2, 5}, respectively. We then repeat this process on the averages ({3, 2, 13, 6}) to get the averages and differences at level-2 decomposition as {2.5, 9.5} and {-0.5, -3.5}, respectively. The average and difference

at level-3 decomposition are obtained similarly as {6} and {3.5}, respectively. Thus the wavelet transform for the original 8-value signal is {6, 3.5, -0.5, -3.5, -1, 0, -2, 5}.

The wavelet can be represented by a binary tree structure, called as *error tree* in the literature. In the error tree, the average at the top level of decomposition is the root of the tree. The detail coefficient at the top level of decomposition is the only child of the root. Each of the node representing a detail coefficient ($d_{i,j}$) has two children representing the detail coefficients at the next lower level of decomposition ($d_{i-1,2j}$ and $d_{i-1,2j+1}$). Figure 4.3 illustrates the error tree for the above 8-value signal. For illustration, we put the original values as the leaf nodes of the tree (depicted by dotted line). In addition, we depict the average coefficients for the levels other than the top level as lightly shaded numbers (they are not included in the wavelet transform/error tree).

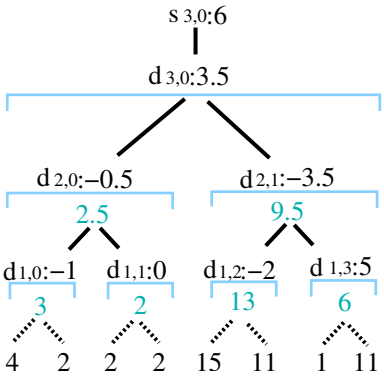


Fig. 4.3. An illustrative example of wavelet error tree.

The original signal can be reconstructed exactly from the wavelet coefficients by taking the reverse step of decomposition. One desirable feature of wavelet transform is that many detail coefficients turn out to be very small and setting them to 0 introduces only small errors in the signal reconstruction. Therefore, the original signal can be approximated by a small number of the most significant wavelet coefficients.

4.2.2 System Model

Without loss of generality, we consider balanced binary tree (fanout $f = 2$), i.e., a binary tree where the height of the left subtree and right subtree of any tree node differs by at most 1. Note that our proposal is not limited to fanout of 2 though. The P2P system consists of N peers with homogeneous resources[1]. We call the peer owning a data object as this data object's *owner peer*, and the peer storing the index of a data object as this data object's *index peer*. The index of a data object is a tuple ⟨value vector, location⟩ where the value vector is a vector of values for a data object on different attributes, and the location is the identifier (IP address) of the owner peer for such a data object.

4.3 Distributed Peer Tree (DPTree)

We first present the tree branch oriented distribution and the high level features of DPTree in Section 4.3.1. We then discuss the overlay structure and navigation algorithm in Section 4.3.2. The access load balancing mechanism is presented in Section 4.3.3.

[1] In the case when peers have heterogeneous resources, we can use proper weighting functions as suggested in [105].

Lastly, we explain how to maintain the overlay structure and tree structure upon peer join/leave/failure and data insertion/deletion in Section 4.3.4.

4.3.1 Overview of DPTree

Before presenting the details of tree branch oriented distribution, we explain the design rationale for this scheme.

First, as mentioned earlier, coupling each tree node with a peer would incur performance bottlenecks and single points of failure at the peers responsible for the higher levels of the tree. Thus, it is better to decouple the concept of a tree node from a peer. This not only avoids the aforementioned problems, but also renders the system the flexility to develop/optimize individual mechanisms for tree maintenance, overlay maintenance, and load balancing according to the patterns of data update, peer update and load distribution, respectively.

Second, we observe that access on a tree normally proceeds from the root down to a leaf node of interest by traversing all non-leaf nodes along the path. This observation implies that the tree nodes on a tree branch (consisting of the tree nodes along the path from the root to a leaf node) are accessed together. Thus, it is beneficial to distribute the tree nodes on a tree branch to the same (or same set of) peer(s).

Based on these observations, we propose tree branch oriented distribution, which distributes (and replicates) tree branches as atomic units to different peers. Each peer manages one or more tree branches (leading to one or more leaf nodes), which form the *local tree* of this peer. A peer stores the index of the data objects enclosed in the assigned leaf nodes. Thus, this peer is the index peer of these data objects. In addition, for each

non-leaf node in the local tree, a peer stores the coverage and height of its two immediate child nodes, used for tree navigation and tree balance invariance checking, respectively. Note that the two immediate children of a non-leaf node in the local tree may or may not be present in the local tree, called *local child node* and *remote child node*, respectively.

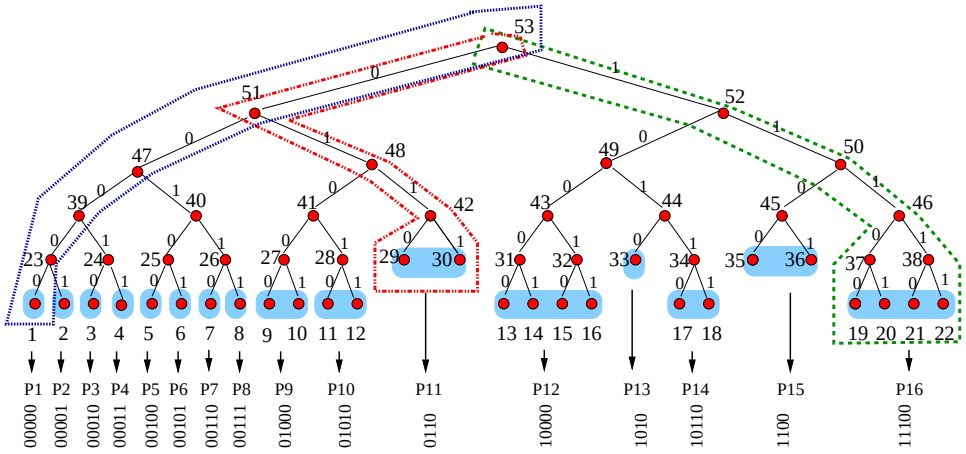


Fig. 4.4. An illustrative example of DPTree.

Figure 4.4 illustrates an example for DPTree. The leaf nodes are grouped into 16 partitions (depicted by the grey rectangles), which are allocated to 16 peers. For the clarity of presentation, we number the tree nodes as Node 1 to Node 53 and the partitions as P1 to P16. Each peer manages the assigned leaf nodes as well as the non-leaf nodes on the tree branches leading to the assigned leaf nodes. For readability, we only show the local trees of Peer 1, 11 and 16 depicted by the dotted polygons in the figure. For

instance, the local tree of peer 16 consists of the assigned leaf nodes 19-22 and the non-leaf nodes on the tree branches leading to these leaf nodes, i.e., Nodes 37–38, 46, 50, 52, 53. Nodes 45, 49 and 51 are the remote child nodes of Nodes 50, 52, and 53 in the local tree of Peer 16, respectively.

DPTree essentially is a fully distributed multi-dimensional balanced tree index. It inherits the nice properties of centralized balanced tree structures, i.e., locality preserving, adaptivity to dynamic data distribution, avoiding indexing dead space, and ability to support various types of queries. In addition, DPTree has its own unique feature, i.e., fair load distribution both vertically and horizontally. Vertically, the tree nodes at the higher levels of the tree have more replicas compared to the tree nodes at the lower levels of the tree. This design avoids the single points of failure and performance bottlenecks, which would normally be associated with the higher levels of a tree structure. Horizontally, the density of peers managing different portion of the tree (the number of tree branches assigned to different peers) reflects the access frequency of different data objects (through load balancing to be discussed later), i.e., more peers manage the frequently accessed portion of the tree.

4.3.2 Overlay Structure and Navigation Algorithm

4.3.2.1 Tree-Aware Overlay

One possible solution to construct the overlay is to couple the tree data structure with the overlay structure. That is, each peer maintains a pointer pointing to the other side of the subtree for each level of its local tree (i.e., pointing to each of the remote child nodes). However different peers might maintain different number of tree branches,

resulting in different peers having different number of routing pointers. This would lead to uneven distribution of routing load among peers.

We propose to organize peers into a *tree-aware overlay*, which is decoupled from but aware of the tree data structure. In tree-aware overlay, peers first obtain a total order through *tree-aware peer naming scheme* and then form an overlay over the ordered space based on the design principle of skip graph (Section 4.2.1.2). Tree-aware overlay enables efficient navigation on the tree yet is easy to maintain.

Tree-Aware Peer Naming. Tree-aware peer naming scheme assigns each peer a unique identifier, i.e., *peerID*. These peerIDs provide a total order of peers, which reflects the locality among assigned tree nodes. The naming scheme works as follows. Each edge in the tree is labelled as 0 or 1. A tree node obtains a *treenodeID*, which is the concatenation of the labels along the edges on the path from the root to this tree node. The treenodeID of a leaf node is called *leafID* specifically. While a peer might manage a couple of tree branches (and leaf nodes), it obtains its *peerID* as the smallest leafID among all the leafIDs of the leaf nodes that it manages. The lexicographic order among peerIDs defines the total order of the corresponding peers. Note that this naming scheme is similar to the one used in SSW (Section 3.1.4.1). The main difference is that here we use peerIDs with variable length while in Section 3.1.4.1, we use IDs with fixed length.

Let's go back to Figure 4.4, which also illustrates the naming scheme. The left edge and right edge of each node are labelled with 0 and 1, respectively. The peerIDs

are depicted at the bottom of the figure[2]. For instance, Peer 11 manages two leaf nodes 29-30, which have leafID as 0110 and 0111, respectively. Thus, Peer 11 obtains its peerID as 0110. The left-to-right order depicted in the figure represents the total order of the 16 peers.

Skip Graph Based Overlay. Above naming scheme maps peers to an ID space. The distribution of peers in the ID space might not be uniform as a result of load balancing. Therefore, we need to construct an overlay that is insensitive to the skewed distribution of peers in the ID space. We observe that skip graph satisfies this requirement. Thus, we organize peers into a skip graph in the ID space. In addition, a peer periodically exchanges heartbeat messages with its neighbors to maintain the consistency of the overlay structure (to be detailed in Section 4.3.4).

4.3.2.2 Aggressive Navigation

Navigation addresses how to navigate to the index peer managing the index of a requested data object[3]. Since each peer has only a partial view of the tree structure, navigation in DPTree is nontrivial. We first need to determine which part of the overlay (tree) might cover the requested data object or destination, i.e., *search space resolution*, and then get to that part of the overlay, i.e., *routing*.

Search Space Resolution. Since tree-aware overlay is constructed over the ID space, search space resolution is to estimate the leafID(s) of the leaf node(s) covering

[2]treenodeID, leafID or peerID bear no relationship with the numbers used to label each tree node (1–53) and partition (P1–P16). The latter is for the purpose of illustration only.

[3]Once the index peer is found, the owner peer can be obtained easily from the index peer. Thus we focus on navigating to the index peer in this paper.

the destination, denoted as *destID*. This is achieved by examining a peer's local tree as follows. A peer examines the tree node in its local tree with the treenodeID as the currently obtained destID (initially set to wildcard $*$). If there is no such tree node in its local tree, this peer can not refine the destID further. Otherwise, the child node that covers the destination is entered and examined further. This process continues till either a leaf node or a remote child node of the local tree is reached. In the former case, the current peer is the destination and the navigation terminates. In the latter case, destID is refined as $A*$, which indicates A, the treenodeID of this remote child node, is a prefix of the destID.

Routing. The destID (with wildcard $*$) obtained through search space resolution as described above actually represents a range of IDs in the ID space. Therefore, the routing algorithm for skip graph as mentioned in Section 4.2.1.2 can not be simply applied here. Instead, we first need to decide which one of the IDs in the range specified by the destID should be used for routing. After careful examination, we observe that the ID furthest away from current peer's peerID should be used for routing. The rationale behind this aggressive navigation algorithm is that even though we might overshoot the destination by routing towards the furthest possible destID, the furthest possible distance to the destination is halved at each step. With the initial furthest possible distance to the destination as N, the navigation to the destination is finished in $logN$ steps with high probability (Theorem 4.1). Algorithm 4 illustrates the pseudo-codes for the navigation.

Algorithm 4 Algorithm for Aggressive Navigation in DPTree.

Navigation at Peer *i*: $i.navigate(q, destID, j)$ **(destID indicates the estimated leafID for q. j, initialized to the top level of the overlay, indicates at which level of the overlay the routing should proceed.)**

```
1: if q ∈ i.index then
2:    Stop.
3: else
4:    Refine destID for q by invoking search space resolution.
5:    x = the furthest ID from Peer i specified by destID.
6:    m = level-j neighbor of i that is closest to x without overshooting x.
7:    if m= NULL then
8:       j = j-1.
9:       GOTO 6.
10:   end if
11:   Forward navigate(q, destID, j) to m.
12: end if
```

THEOREM 4.1. *Given N peers in DPTree, a peer can navigate to any part of the overlay in $O(logN)$ hops (steps) with high probability by using the proposed aggressive navigation algorithm.*

Proof: We prove this theorem by induction. For presentation clarity, we assume that N is power of 2. In the case that N is not power of 2, we can replace N by $N' = 2^{\lceil logN \rceil}$ in the following proof. Given the peer distance to the destination at the beginning of the i^{th} step as $x_i \leq \frac{N}{2^{i-1}}$, we prove the peer distance is halved after this routing step, i.e., $x'_i \leq \frac{N}{2^i}$. Once this is proven, we can easily derive after $logN$ steps, the peer distance to the destination is reduced to 1. The message can then be trivially forwarded to the destination with one additional step.

Assume the highest level of the overlay is h ($h = logN$ with high probability). Recall that each peer has $logN$ pairs of neighbors with the pair of level-j neighbors at peer distance (denoted as d_j) as 2^j with high probability ($0 \leq j \leq logN$-1).

We start from the base case with $i = 1$. It is obvious that $x_1 \leq \frac{N}{2^{1-1}} = N$.

Assume that we are at the beginning of the i^{th} step and the peer distance to the destination is $x_i \leq \frac{N}{2^{i-1}}$. We have two possible scenarios: 1) $x_i > \frac{N}{2^i}$; 2) $x_i \leq \frac{N}{2^i}$. For the first scenario, one of the current peer's neighbors at level-(h-i) must be closer to the furthest possible destID without overshooting it. Therefore, this neighbor is chosen to forward the request. As a result, the peer distance to the destination (x_i') is then reduced to $x_i - d_{h-i} = x_i - \frac{N}{2^i} \leq \frac{N}{2^i}$. For the second scenario, the request may or may not be forwarded at this routing step. If the request is forwarded, x_i' is reduced to $d_{h-i} - x_i$ ($\leq \frac{N}{2^i}$). Otherwise, x_i' equals to x_i ($\leq \frac{N}{2^i}$). □

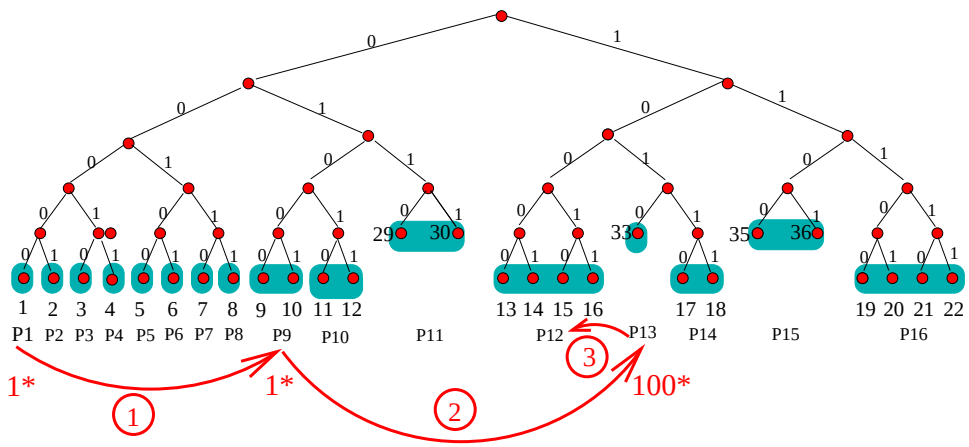


Fig. 4.5. An illustrative example for navigation in DPTree.

Figure 4.5 illustrates an example for navigation in DPTree. Assume Peer 1 wants to navigate to leaf node 13 (please refer to Figure 4.2 for the overlay structure). Peer 1

invokes search space resolution and obtains the destID as $1*$. Since the peerID of Peer 1 is 00000, the furthest possible destID is 111...111. Peer 1 examines its neighbors at the top level and routes the message to Peer 9. When Peer 9 receives the message, it invokes search space resolution and still obtains the destID as $1*$. Similarly, Peer 9 examines its neighbors at level 3. Since the only neighbor at this level (Peer 1) is further away from the destID, routing descends one level to level 2. At this level, Peer 9 forwards the message to Peer 13, which is closer to 111...111 without overshooting it. When Peer 13 receives the message, it invokes search space resolution and refines the destID as $100*$. The furthest destID from Peer 13 is 1000...000. Peer 13 examines its neighbors at level-2 and does not forward the message at this level since neither neighbor qualifies. Routing descends to level-1. Similarly, neither neighbor at level-1 qualifies and routing descends one more level to level-0, where the message is forwarded to the destination Peer 12.

4.3.3 Wavelet-assisted Load balancing

In the following, we first describe an intuitive solution for load balancing and point out the limitations, which motivate wavelet-assisted load balancing. We refer access load as load if the context is clear.

One solution for load balancing as suggested by majority of existing works is to let an overloaded peer choose the least loaded peer (e.g., [27, 48]) or a random peer (e.g., [65, 105]) in the system as the *target peer* to shed part of its load to. The target peer merges its load to its neighbors, leaves its original place, and rejoins the overlay as the neighbor of the overloaded peer to take part of its load. We call this solution as

leave-rejoin (LR) mechanism and the two variants as *LeastLR* (the target peer is the least loaded peer) and *RandomLR* (the target peer is a randomly selected peer).

LR mechanisms have a couple of drawbacks. First, the peers in the neighborhood of the target peer might not be lightly loaded. Merging the load from the target peer to its neighbors might cause the neighbors become heavily loaded, resulting in cascading load balancing operations. Second, the leave-rejoin process causes changes on the overlay links, and thus requires update on the overlay, which might be costly. As suggested later, in some cases when the target peer and the overloaded peer are nearby in the overlay, it might be a better option to move the data rather than the peer by letting the extra load ripple through the neighbors to reach the target peer without affecting the overlay structure. Finally, LeastLR mandates complicated and costly mechanisms to maintain the load information on the peers.

We observe that if a peer somehow has an approximate global view (with sufficient accuracy) of the load distribution in the system, all of the weaknesses mentioned above can be avoided. To obtain such a global view, we need to summarize and disseminate the load distribution of the system in a compact yet sufficiently accurate format. Inspired by wavelet, a well-studied compression tool in signal procession, we propose wavelet-assisted load balancing, which leverages wavelet to facilitate *load monitoring* and *load adjusting* in P2P systems in a light-weighted yet effective fashion.

To perform load monitoring, peers exchange load information with their neighbors at each level of the overlay through heartbeat messages and form the approximate wavelet of the load distribution in the system, called *loadwavelet*. Load adjusting is performed in the following three steps. 1) Overloading detection: guided by loadwavelet, peers can

easily determine whether they are overloaded. 2) Target peer selection: an overloaded peer leverages the multi-resolution view of loadwavelet to find a target peer, i.e., the peer that is lightly loaded and whose neighborhood is lightly loaded as well. 3) Load shedding: load shedding consists of two steps: moving the load of the target peer to its neighbors, and moving half of the load from the overloaded peer to the target peer. To perform the first step, we propose *rippled target load moving (RTLM)*; to perform the second step, we propose *elastic load shedding (ELS)*, where peers[4] switch between two different load shedding mechanisms, i.e., *rippled load shedding* and *direct load shedding*, depending on which one is more cost-effective by taking into consideration the cost incurred by index movement as well as overlay maintenance. Figure 4.6 illustrates the high level flow of wavelet-assisted load balancing.

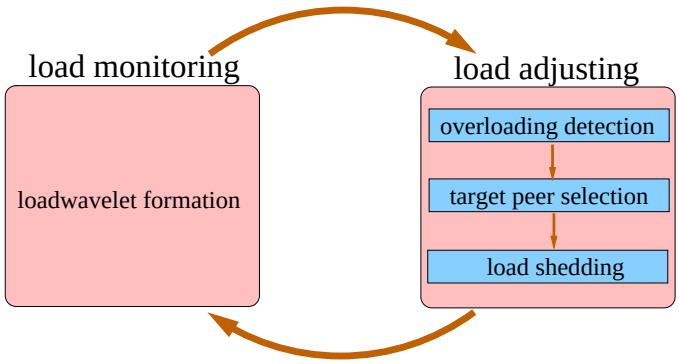


Fig. 4.6. Wavelet-assisted load balancing.

[4]We set the leaf node capacity c to be the payload size of a packet. The rationale is to let a leaf node also serve as the finest unit of load transferred between two peers during load balancing.

Compared to the aforementioned existing works, our load balancing mechanism has the following three advantages.

1. It uses a light-weighted mechanism to maintain a sufficiently accurate summary of the load distribution in the system.

2. It avoids cascading load balancing through the following two designs. First, it takes into consideration the load on a peer as well as the load on the peers in its neighborhood when choosing a target peer. Second, RTLM moves the load on the target peer to multiple neighbors when necessary to avoid overloading a single neighbor.

3. It takes into account the cost incurred by both index movement and overlay maintenance, and switches between two different load shedding mechanisms depending on the cost. In contrast, prior works only consider the cost of index movement and conduct load shedding using a variant of direct load shedding.

We now explain the details of load monitoring and load shedding.

4.3.3.1 Load Monitoring

We first assume that we have a perfect skip graph (where a peer's neighbor at level i is at exactly 2^i peer distance) and unbounded communication resource to form an exact (complete) loadwavelet. Later on, we discuss how to relax these assumptions.

A peer first exchanges its current load with its level-0 neighbor clockwise on the overlay and forms the wavelet for the "signal" consisting of two values, i.e., its current

load and its neighbor's current load. This peer then exchanges this wavelet with its level-1 neighbor clockwise on the overlay and forms the wavelet for the signal consisting of four values, i.e., the load on this peer and the following three consecutive peers. Following this process, through message exchange with a level-i neighbor on the overlay, the wavelet covering 2^{i+1} consecutive peers is obtained. This process continues till the top level of the overlay is reached. At this point, we obtain the loadwavelet of all the peers in the system.

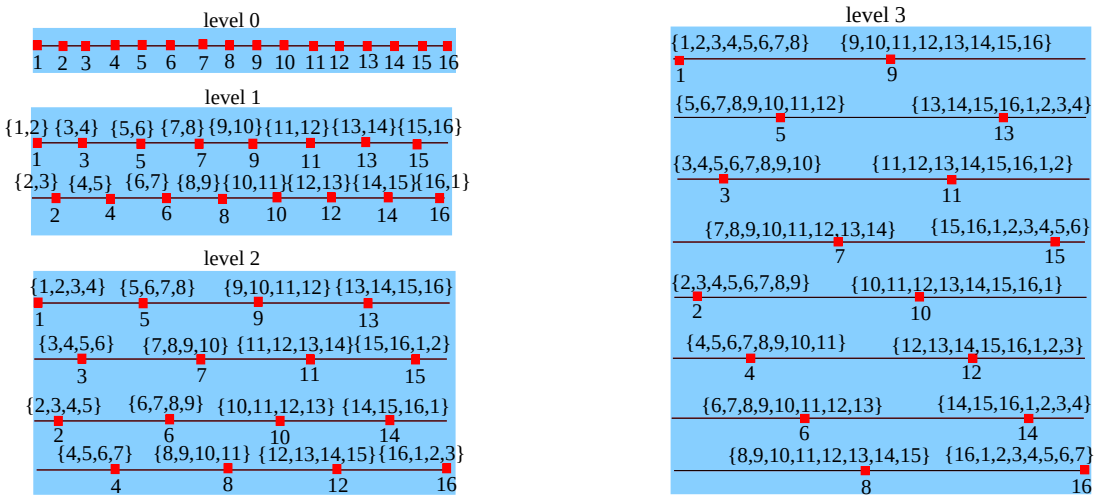


Fig. 4.7. An illustrative example of loadwavelet formation.

We use Figure 4.7 to illustrate an example of loadwavelet formation in a system consisting of 16 peers (please refer to Figure 4.2 for the overlay structure). For illustration, we represent peers on each level of the overlay on a straight line. The numbers in

the parentheses are the peers whose load values are included in the loadwavelet formed at the previous level of the overlay. After message exchange with the neighbor at the top level (level-3), each peer obtains the loadwavelet of the system.

Now we discuss how to relax the assumption of unbounded communication resource. The length of the complete wavelet formed at higher levels of the overlay is large (e.g., the length of the loadwavelet formed at level-i of the overlay is 2^{i+1}), and exchanging such complete wavelets is costly. As mentioned in Section 4.2.1.3, one of the desirable features of wavelet is that the most significant wavelet coefficients can approximate a signal within sufficient accuracy. Here, we take advantage of this feature of wavelet by selecting the m most significant wavelet coefficients to approximate the loadwavelet. m is a tunable parameter balancing the precision and the construction cost of the wavelet (which will be evaluated in details later).

Up to now, we assume that a peer's level-i neighbor is at exactly 2^i peer distance. In reality, a peer's level-i neighbor is at 2^i peer distance with high probability instead of exactly. Therefore, during the above wavelet formation, the load values for certain peers might be counted more than once or not be counted (when a level-i neighbor is at peer distance less than 2^i or greater than 2^i, respectively). Since the percentage of redundant load values or missing load values is expected to be very small, this doesn't affect the accuracy of loadwavelet significantly as confirmed through evaluation later in Section 4.5.

4.3.3.2 Load Adjusting

As mentioned earlier, load adjusting consists of three tasks, i.e., overloading detection, target peer selection, and load shedding. We denote the load on Peer i as l_i and the average load of the system as $\bar{l}$.

Overloading Detection. A peer obtains the average load of the system easily from the loadwavelet, i.e, the first wavelet coefficient. If the current peer's load is more than δ times of the average load, it is marked as an overloaded peer. δ is a tunable system parameter determining the tradeoff between the cost of load balancing and how well the system is balanced. A smaller value creates a more balanced system at higher cost.

Target Peer Selection. In addition to providing a compact summary of the load distribution, loadwavelet also provides multi-resolution views of the load distribution in the system. We exploit this multi-resolution feature to choose the lightly-loaded peer residing in a lightly-loaded neighborhood as the target peer. For presentation clarity, we use the wavelet error tree (described in Section 4.2.1.3) to explain how target peer selection is performed. A peer examines the error tree starting from the root node. If the detail coefficient is greater than 0, the average load on the left half of the overlay is smaller. Thus, we drill down one level on the error tree and enter the left child. On the other hand, if the detail coefficient is smaller than 0, the right child is entered. In the case when the detail coefficient is 0, we examine both children's detail coefficients and enter the child node with larger absolute value (implying one quarter of the overlay

has the lightest load among the four quarters). This process continues till we reach the bottom level of the error tree where the target peer is obtained.

Using Figure 4.8 as an example for the load distribution in a system with 8 peers, target peer selection proceeds as follows. We first examine $d_{3,0}$. Since $d_{3,0}$ is a positive value, we enter the left child and examine $d_{2,0}$. $d_{2,0}$ is -0.5. Thus, we enter the right child $d_{1,1}$. The value of $d_{1,1}$ is 0 and we randomly select one of the two leaf child nodes as the target peer. In this case, we select the third peer (with load 2) as the target peer. As a comparison, LeastLR chooses the seventh peer (with load 1) as the target peer. Notice that both neighbors of the seventh peer have high load (as 11).

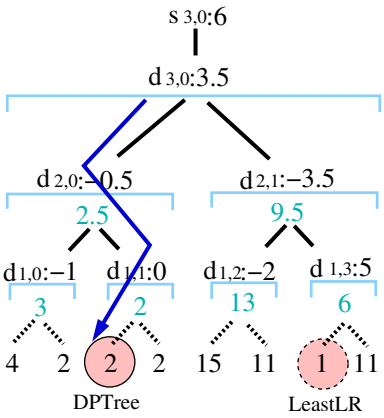


Fig. 4.8. An illustrative example of target peer selection.

Load Shedding. Load shedding consists of two steps: moving the load of the target peer to its neighbor(s) through RTLM, and shedding half of the load from the

overloaded peer to the target peer through ELS by switching between rippled load shedding and direct load shedding, depending on the cost incurred by both index movement and overlay maintenance. In the following, we first describe the ideas of RTLM and ELS. We then explain how to estimate the cost of rippled load shedding and direct load shedding.

The intuition of RTLM is to ripple the load on the target peer to multiple consecutive neighbor peers on the overlay structure to avoid overloading a single neighbor peer of the target peer. More specifically, the target peer first moves its load (by moving the corresponding leaf nodes) to one of its two immediate neighbors that has smaller load. If after this load moving, this neighbor is not overloaded (the current load l_i of this peer is less than $\delta \cdot \bar{l}$), RTLM terminates. Otherwise, the extra load $(l_i - \delta \cdot \bar{l})$ is moved from this neighbor to this neighbor's neighbor. The above process terminates until the last reached neighbor is not overloaded (after some load is moved to this neighbor).

We now explain the details of rippled load shedding and direct load shedding in ELS. In rippled load shedding, the load ripples through the neighbors to reach the target peer. The overloaded peer sheds half of its load to its immediate neighbor (by moving the corresponding leaf nodes), which then sheds the same amount of load to the neighbor's neighbor. This process continues till the target peer is reached. During rippled load shedding, although moving the leaf tree nodes between neighbors causes changes on the peerIDs of the peers involved, the order among peerIDs is not changed and thus the overlay links are not changed. Therefore, the advantage of rippled load shedding is that it incurs only index movement cost but no overlay maintenance cost.

Rippled load shedding works well when the overloaded peer and target peer are nearby in the ID space.

Direct load shedding is similar to the load shedding scheme adopted in LR mechanisms. It requires the target peer merge its load with its neighbors, leave its original place, and rejoin the overlay as a neighbor of the overloaded peer to take over half of its load. In addition to incurring index movement cost, the leave and rejoin of the target peer affects the overlay links, incurring overlay maintenance cost.

We now explain how a peer estimates the cost of both rippled load shedding and direct load shedding. Before we give out the cost estimation, we first introduce some notations used in the following. i, j denote the peerID of the overloaded peer and the target peer, respectively. $C_i(x)$ denotes the cost incurred at Peer i by shedding x load. Since both rippled load shedding and direct load shedding involve RTLM, we ignore the cost incurred by RTLM during the cost estimation for simplicity (this does not affect the final result since we only need to know whether rippled load shedding or direct load shedding has smaller estimated cost). The cost for rippled load shedding (denoted by C_r) is estimated as:

$$C_r = \sum_{k=i}^{j-1} C_k(l_i/2)$$

where $C_k(l_i/2)$ is the cost incurred by rippling half of the load from Peer i through the neighbors. The cost for direct load shedding (denoted by C_d) is estimated as:

$$C_d = 5logN + C_i(l_i/2)$$

where $logN$ messages are incurred for the overloaded peer to inform the target peer, another $4logN$ messages are incurred to update the overlay links ($2logN$ messages for the target peer to establish new neighbors, and $2logN$ messages for the target peer to inform its old neighbors to update their neighbors). Either rippled load shedding or direct load shedding is chosen depending on which one has smaller cost as estimated above.

Note that in the process of load balancing, the distribution of tree nodes among peers is changed. In addition, some part of the overlay structure is changed if direct load shedding is adopted. However, the tree structure itself is unchanged. This confirms the benefits of separating the tree structure from the overlay structure. In addition, our proposed load shedding mechanisms are generic enough to be applicable in other load balancing mechanisms. For instance, they can be applied on LR mechanisms to improve their performance as demonstrated later in Section 4.5.

4.3.4 Maintenance in DPTree

We adopt the soft state mechanism to maintain the consistency of the overlay structure and tree data structure. The basic idea of soft state mechanism is to associate a state with a timer, and refreshes (or deletes) the state if a refreshment message is (or is not) received before the associated timer expires. Based on this idea, a peer associates a timer with each of its neighbors and each of its indexed data objects. A peer sends heartbeat messages to its neighbors periodically. If a peer does not receive a heartbeat message from a neighbor before the associated timer expires, it infers this neighbor leaves or fails (and invokes overlay update to be detailed shortly). Similarly, a peer republishes

(refreshes) its data objects to the system periodically. If the index peer of a data object does not receive a refreshment message before the associated timer expires, it infers this data object disappears from the system (and invokes data deletion). This mechanism ensures the consistency of both overlay structure and tree data structure in a simple yet light-weighted fashion.

In the following, we provide the detailed operations performed upon peer join, peer leave, peer failure, data insertion, and data deletion.

4.3.4.1 Peer Join/Leave/Failure

Peer Join. A peer first decides the place to join in the overlay. Then the peer joins the overlay level-by-level by establishing neighbors at each level. Finally, it publishes the data brought with it to the system.

A peer can decide the place to join in the overlay either randomly or in accordance with certain heuristics. Without loss of generality, we let a peer randomly choose the place to join in the overlay. A peer then joins the overlay level-by-level starting from level-0 by establishing two neighbors at each level. This process incurs $2logN$ messages in total. To join level-0 in the overlay, a peer establishes two neighbors with its neighbor peers at level-0. In addition, it informs these two neighbors to update their neighbor pointers at level-0 (pointing to the newly joined peer). Then this peer joins the higher level of the overlay (level-i with $1 \leq i \leq logN$-1) as follows. It first randomly chooses a ring (R) to join between the two level-i child rings of its level-(i-1) ring. Then it walks on its level-(i-1) ring clockwise and counterclockwise till reaching the first peers in either direction that also join level-i ring R as its two neighbors at level-i. In addition,

the newly joined peer informs its neighbors at level-i to update their neighbor pointers (pointing to the newly joined peer). Finally, the newly joined peer publishes the index information for the local data to the system (to be detailed shortly).

Peer Leave/Failure. Peers exchange heartbeat messages with its neighbors periodically to see whether they are still alive. Once a peer detects that one of its neighbors at level-i is not alive (due to lack of heartbeat messages as described above), it starts to re-establish its neighbors level-by-level starting from level-i. This process incurs at most $2logN$ messages in total. In addition, the tree branches previously assigned to this peer is re-distributed to other alive peers through the index republishing process as described above. To facilitate neighbor recovering, we assume that a peer maintains a backup neighbor set on level-0, which consists of a couple of consecutive neighbor peers of this peer's neighbor on level-0 on the overlay. Since a peer can always recover its neighbors at higher level on the overlay through neighbors at lower levels on the overlay, we do not need to maintain backup sets for neighbors at levels other than level-0.

4.3.4.2 Data Insertion/Deletion

Data Insertion. Inserting a data object basically is to publish the index of this data object. This involves two steps: locating the leaf node (and corresponding peer) to insert the index, and inserting the index of the data object to the chosen leaf node. We use Algorithm 5 to explain the basic operations in data insertion.

Locating the leaf node is performed similarly as tree navigation with the following modifications. A peer checks its own local tree starting from the root node. If the root node does not cover this new data object, the coverage of the root node is enlarged to

Algorithm 5 Algorithm for inserting a data object into DPTree.

insert(t, o) (t, initialized to the root node, is the current tree node under consideration. o is the data object to be inserted.)

```
1: if t is a remote child node then
2:    Propagate the request to a peer that has t as the local child node.
3: else
4:    if o ∉ t.coverage then
5:       Enlarge t.coverage to accommodate o.
6:       Propagate the coverage change to the affected peers.
7:    end if
8:    if t is not a leaf node then
9:       if o ∈ t.left.coverage then
10:         insert(t.left, o).
11:      else if o ∈ t.right.coverage then
12:         insert(t.right, o).
13:      else
14:         Calculate the size of the coverage enlargement to accommodate o for t.left and t.right.
15:         if t.left requires smaller coverage enlargement to accommodate o then
16:            insert(t.left, o)
17:         else
18:            insert(t.right, o).
19:         end if
20:      end if
21:   else
22:      if t is not full then
23:         Insert o into the leaf node t.
24:      else
25:         Split t into two leaf nodes (t1 and t2) and insert o into t1 or t2.
26:         t1.parent = t2.parent = t.
27:         t.height = 1.
28:         Propagate the height change to the affected peers and invoke height balancing when
            necessary.
29:      end if
30:   end if
31: end if
```

accommodate this data object. The two child nodes of the root node are then examined. If the new data object is covered by one of the child nodes of a node in the local tree, the corresponding child node is entered and examined further (as shown in Lines 9-12 of Algorithm 5). On the other hand, when the new data object is not covered by neither of the two child nodes of a tree node, one of the child nodes that requires smaller coverage enlargement to accommodate the new data object is chosen to enlarge its coverage and accommodate the new data object (as shown in Lines 13-19). The change on the coverage of a tree node is propagated to the affected peers through *coverage update* (Lines 4-7, to be detailed shortly). This process continues till the leaf node, where the index of the new data object will be inserted, is reached.

Inserting the index to the chosen leaf node is performed as follows. If the leaf node has some space to accommodate the new data object (the number of data objects is less than the leaf node capacity c), the index of the new data object is inserted there (as shown in Lines 22-23). Otherwise, the leaf node is split into two new leaf nodes (Lines 25-28). The original leaf node now acts as the parent of these two new leaf nodes. Therefore, it updates its height to 1 and propagates this change on the height to the affected peers through *height update* when necessary (to be detailed shortly). In the case that the tree becomes unbalanced in height, *height balancing* is invoked to restore the height balance invariance. Height balancing basically involves rotations similar to the ones for AVL tree balancing [75]. After performing the rotation, the coverage of tree nodes involved in the rotation need to be updated and propagated to relevant peers similarly as coverage update (to be detailed shortly).

Data Deletion: When a peer infers a data object disappears from the system (due to the lack of refreshment messages as described above), it deletes the index of this data object as follows. It first reaches the leaf node covering the data object to be deleted, and then deletes the data object from the leaf node. If the number of data objects in this leaf node falls below c/2, this leaf node is merged with its sibling, and the changes on the height and coverage of the parent node of this leaf node are propagated to the affected peers when necessary. Here we delay the merging for some period of time to allow some new data objects to be inserted into this leaf node so that we can avoid frequent node splitting and node merging. Algorithm 6 illustrates the basic operations in data deletion.

Algorithm 6 Algorithm for deleting a data object from DPTree.

delete(t, o) (o is the data object to be deleted. t is the leaf tree node storing o.)

```
1: Delete o from leaf node t.
2: if t is less than half full then
3:    Merge t with its sibling leaf node.
4:    Propagate the height change and coverage change to the affected peers.
5: else
6:    if t.coverage is shrunk then
7:       Propagate the coverage change to the affected peers.
8:    end if
9: end if
```

The changes on the coverage and height of a tree nodes upon data insertion/deletion are propagated to the peers managing the tree nodes in the subtree rooted at the parent node of the tree node performing the changes (since all these peers record the coverage/height information of this tree node). We observe that through the tree aware peer

naming scheme (Section 4.3.2.1), all these affected peers are consecutively positioned in the ID space. Therefore, this update incurs x messages and $logx$ propagation hops where x is the number of affected peers. In addition, the cost of propagating coverage change and height change is expected to be low due to the following reasons. First, with the increase of the tree levels, the coverage changes on the corresponding tree nodes are expected to become less frequent. This is beneficial since the propagation of the coverage changes on the tree nodes at the higher levels is more costly than that at the lower levels. Second, the height change is expected to be very infrequent since leaf node splitting and leaf node merging are infrequent events, which happen in every $\frac{1}{c}$ data insertions or data deletions on average. Height balancing is even less frequent since not every leaf node splitting/merging will violate the height balance invariance. These expectations are confirmed by our extensive experimentations to be presented in Section 4.5.

Note that all these changes on the tree structure caused by data insertion/deletion do not affect the overlay structure since the total order among peers does not change. This confirms the advantage of decoupling the tree structure from the overlay structure as discussed in Section 4.3.1.

4.4 Application of DPTree

In the following, we show how to support two most common types of complex queries, i.e., range query and KNN query, in DPTree (to support point query, we can directly apply the navigation algorithm presented in Section 4.3.2.2 with the query as the destination).

4.4.1 Range Query

A range query, specified by a query center q and a query radius r, returns all the data objects that are at distance less than r to q as the query results. We extend the navigation algorithm given earlier (Section 4.3.2.2) to process a range query. The difference of a range query from a point query is that the destination is specified as a query range instead of a query point. Therefore, during search space resolution, all the tree nodes overlapping with the query range need to be entered and examined. If multiple child nodes need to be examined, multiple threads of processing are invoked to examine these nodes in parallel. The process terminates when either the subtree corresponding to a specific destID is pruned or the peer managing corresponding leaf node is reached. In the latter case, the qualifying data objects are returned to the query issuer as the query result.

4.4.2 K Nearest Neighbor Query

A K nearest neighbor (KNN) query, specified by a reference data object (reference point) q and a real value K, returns the K data objects that are closest to q in the data set as the query result. A KNN query is processed in two steps, obtaining a good enough candidate set, and refining the candidate set through range query. To obtain a good enough candidate set, the peer managing the leaf node that covers the reference data object or that is closest to the reference data object (in the case that none of the leaf nodes covers the reference data object) is reached through the navigation algorithm. This peer then obtains the K data objects (either owned or indexed by this peer) that are closest to the reference data objects as the candidate set. It is possible that some data

objects in other peers might be closer to the reference data object than the data objects in the candidate set are. In order to obtain these closer data object, the second step is invoked as follows. The current peer obtains a query range centered at the reference data object with the distance to the K^{th} element in the candidate set as the radius. Then similar procedure as range query is employed. Different from the range query algorithm as described earlier where multiple streams of parallel processing are invoked when multiple child nodes overlap with the query range, here we require sequential processing is invoked to facilitate candidate set refining. Basically, we choose to examine the child node that is closest to the reference data object first. As soon as a closer data object is obtained, the candidate set and the query range are refined. This process continues until the refined query range is completely examined. The K data objects in the final candidate set are returned as the query result.

4.5 Performance Evaluation

We now proceed to the evaluation of DPTree. We import R-tree according to the proposed DPTree framework. We evaluate DPTree from four aspects, i.e., load balancing, routing, query processing, and maintenance.

4.5.1 Load Balancing

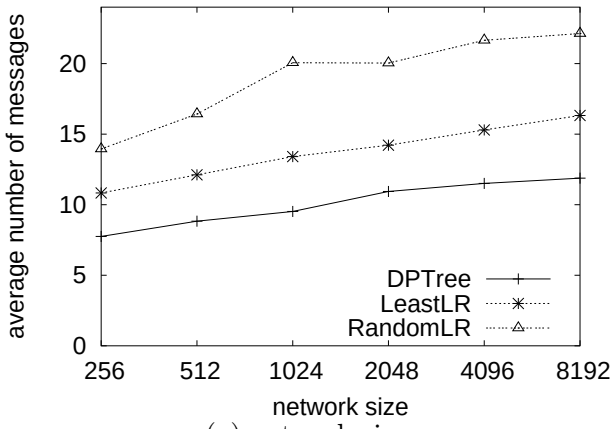
To evaluate the performance of our proposed load balancing mechanism, we measure the average number of messages incurred per peer to make the system δ-balanced, defined as a system with $\forall i \in \{0, 1, 2, ..., N\}, l_i \leq \delta \cdot \bar{l}$ (l_i is the load of Peer i and $\bar{l}$ is the

average load of the system). For comparison, we implement LeastLR (proposed in Mercury [27, 48]) and RandomLR (proposed in [65, 105]) as described in Section 4.3.3. In LeastLR, the least loaded peer is chosen as the target peer. In RandomLR, a randomly selected peer is chosen as the target peer. In both LeastLR and RandomLR, the load of the target peer is simply moved to its two immediate neighbors, and the load on the overloaded peer is adjusted through direct load shedding.

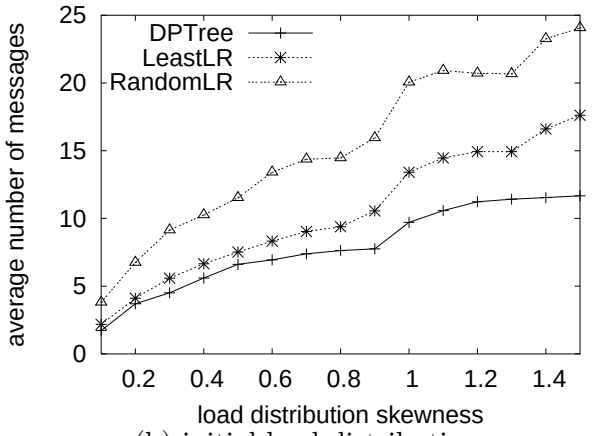
We evaluate the average number of messages incurred per peer by load balancing under different network sizes, different initial access load distribution, and different sizes of (approximate) wavelet (indicated by m as discussed in Section 4.3.3.1). The network size varies from 256 to 8192 and the default setting is 1024. We use Zipf-distribution controlled by a skewness parameter, called *load distribution skewness*, to model the initial access load distribution in the system. When the skewness is large, the distribution is skewed. When the skewness is 0, the distribution is uniform. The default setting for the network size, load distribution skewness, and wavelet size is 1024, 1, and $5\% \cdot N$, respectively. In addition, we also improve the original LR mechanisms by replacing the load shedding mechanisms by our proposed techniques. We compare these variants with the original LR mechanisms to demonstrate the improvement by incorporating our load shedding techniques in LR mechanisms. For presentation brevity, we present the results with δ set to 2 (the general trends observed under different setting for δ are similar).

4.5.1.1 Effect of Network Size

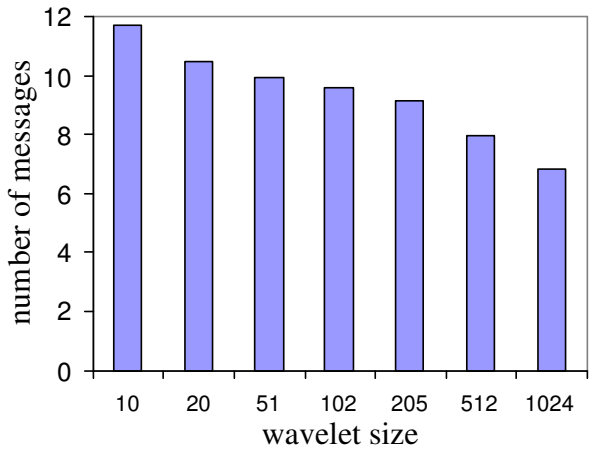
Figure 4.9(a) shows the result under different network sizes. The x-axis is on logarithmic scale for readability. From this figure, we see the average number of messages



(a) network size



(b) initial load distribution



(c) wavelet size

Fig. 4.9. Cost of load balancing in DPTree.

incurred by load balancing increases almost linearly with the network size, which is expected. In addition, the average number of messages incurred by wavelet-assisted load balancing is smaller than those incurred by LR mechanisms.

4.5.1.2 Effect of Initial Load Distribution

Figure 4.9(b) shows the result under different initial load distribution (different skewness values). The average number of messages increases with the skewness value. This is expected since more skewed load distribution requires more load to be redistributed among peers to make the system δ-balanced. The increase rate of the cost incurred by wavelet-assisted load balancing is much smaller than those incurred by LeastLR and RandomLR. This demonstrate the superiority of wavelet-assisted load balancing under more skewed load distribution.

4.5.1.3 Effect of Wavelet Size

Figure 4.9(c) shows the result under different wavelet sizes, i.e., 10, 20, 51, 102, 204, 512, corresponding to 1%, 2%, 5%, 10%, 20% and 50% of the size of the original wavelet transform (1024). From this figure, we see that when the wavelet size decreases, the average number of messages increases. This is because more errors are introduced in signal reconstruction by a more compact approximate wavelet, which causes the selection of the target peers deviate from the optimal ones, incurring some extra messages. However, even when the size of the approximate wavelet is only 1% of the original wavelet, the average number of messages incurred by wavelet-assisted load balancing is

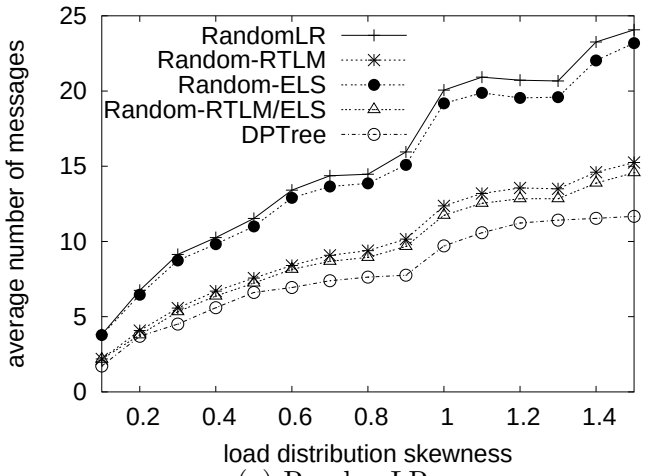
still smaller than those incurred by LeastLR and RandomLR, which are around 13 and 20, respectively.

This set of experiments demonstrates the superiority of wavelet-assisted load balancing. In addition, a compact approximate wavelet can be formed as the by-product of heartbeat messages exchange between neighbors at almost no additional cost. In contrast, LeastLR requires non-trivial maintenance to keep track of the load distribution in the system.

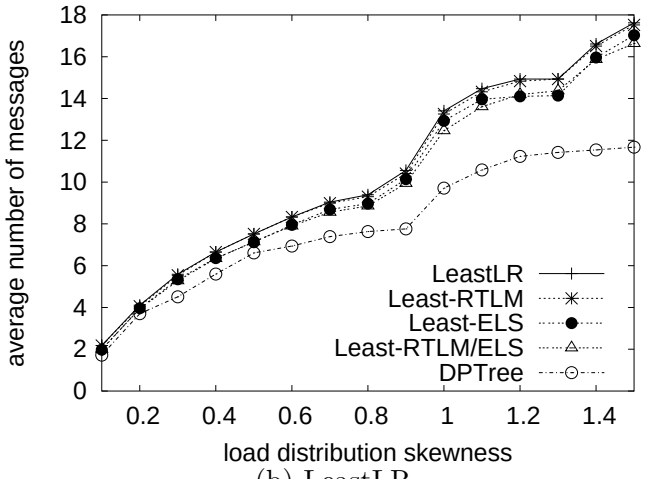
4.5.1.4 Enhancement on LR Mechanisms

We implement some variants of the LR mechanisms by replacing the original load shedding mechanisms with our proposed techniques. Random-ELS or Least-ELS are RandomLR or LeastLR with our ELS load shedding mechanism; Random-RTLM or Least-RTLM are RandomLR or LeastLR with our RTLM mechanism; Random-RTLM/ELS or Least-RTLM/ELS are RandomLR or LeastLR with our RTLM and ELS mechanisms.

Figure 4.10 shows the performance of these variants. For comparison, we also include the plot of DPTree. The general trend that we observed from this set of experiments is that Random-RTLM/ELS performs better than Random-RTLM or Random-ELS, which in turn perform better than the original RandomLR mechanism. Similarly, Least-RTLM/ELS performs better than Least-RTLM or Least-ELS, which in turn perform better than the original LeastLR mechanism. The performance difference among these variants is more significant in RandomLR.



(a) RandomLR



(b) LeastLR

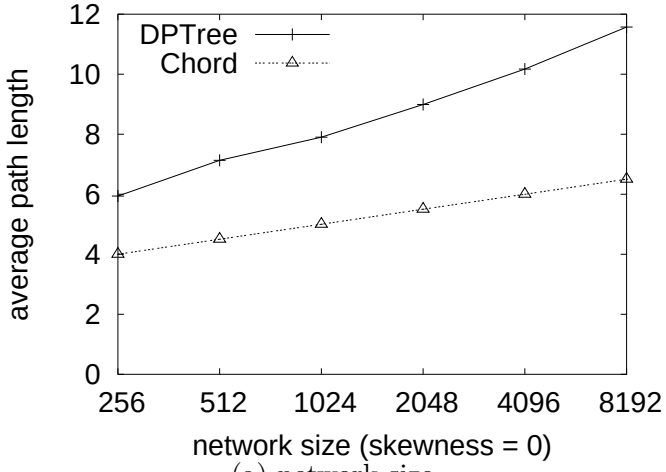
Fig. 4.10. Enhancement on LR mechanisms by incorporating RTLM/ELS.

4.5.2 Routing

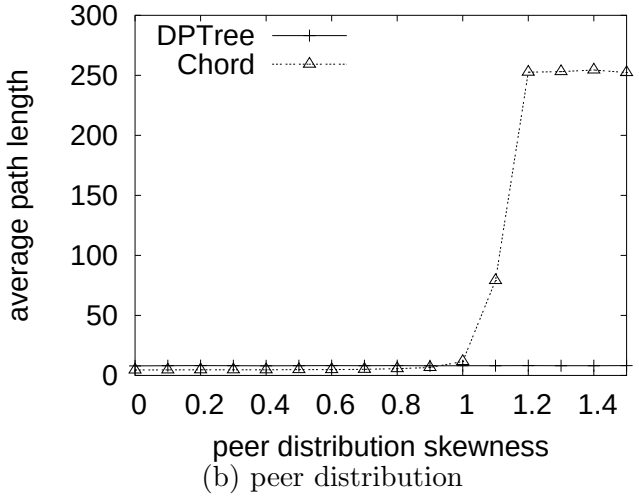
To demonstrate the efficiency of our proposed overlay structure and aggressive navigation algorithm, we distribute peers in an ID space with domain as $\{0, 2^{30}\}$. We conduct experiments to measure the average routing path length under different network size and different distributions of peers in the ID space. We use Zipf-distribution controlled by a skewness parameter, called *peer distribution skewness*, to model the peer distribution in the ID space. If unspecified otherwise, the default setting for the skewness is set to 0. For comparison on routing performance, we implemented modified Chord where each peer has $logN$ pairs of finger table entries with a pair pointing to peers at distance 2^i clockwise and counterclockwise ($0 \leq i \leq logN - 1$) in the ID space. Each peer issues 100 random routing requests and the results presented below are averaged over these requests.

4.5.2.1 Effect of Network Size

Figure 4.11(a) shows the average path length when the network size varies from 256 to 8192. Note that the x-axis is on logarithmic scale for readability. From this figure, we see that the average path length using DPTree is around $logN$, confirming Theorem 4.1 in Section 4.3.2.2. The routing path length using Chord is slightly lower compared to DPTree. The slight better performance of Chord over DPTree under uniform peer distribution is due to the fact that DPTree requires that routing be performed level-by-level while Chord does not have this restriction. However, Chord performs much worse under skewed peer distribution as shown shortly.



(a) network size



(b) peer distribution

Fig. 4.11. Routing performance of DPTree.

4.5.2.2 Effect of Peer Distribution

Figure 4.11(b) shows the average path length when the peer distribution skewness varies from 0 to 1.5 with network size as 1024. From this figure, we can see that the average path length in DPTree is not affected by the peer distribution. However, the average path length of Chord is very sensitive to the peer distribution. When the skewness is small (0 to 0.9), i.e., the peer distribution is less skewed, the performance of Chord is similar to DPTree. When the skewness increases, the path length of Chord increases sharply to 1/4 of the network size (similar results are observed under other network sizes). This result confirms our design rationale of the overlay structure and aggressive navigation algorithm as described in Section 4.3.2.

4.5.3 Query Performance

We implement the algorithms to process three different queries, i.e., point query, range query and KNN query, in DPTree. For comparison, we modify CAN overlay (by placing data objects in accordance with their attribute values instead of randomly hashed values) and implement the query algorithms (similar to that described in Section 4.4) on top of CAN. We choose CAN overlay for comparison in this set of experiments due to the following two reasons. First, majority of prior works on queries for multi-dimensional data objects are based on CAN or some variants of CAN. Second, although some proposals (e.g., [27]) might be superior to CAN in terms of supporting one specific type of query, they can not support all types of queries that we are considering here.

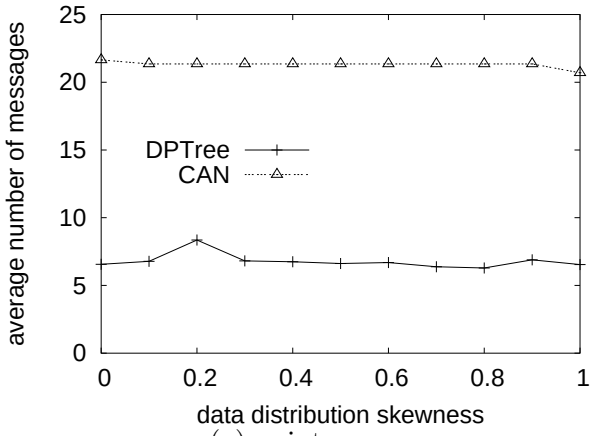
We evaluate the query performance under different data sets, query workloads, and network sizes. Without loss of generality, the dimensionality of data objects is set

to 2. The data sets are generated as follows. A certain number of seed points are first randomly generated in the 2-dimensional data space. Then from each of these seed points, we generate some random data points with distance to the seed point following Zipf distribution controlled by *data distribution skewness*. By varying the skewness value, we obtain a spectrum of synthetic data sets ranging from uniformly distributed data set to highly skewed data set. The total number of data points is $100 \cdot N$.

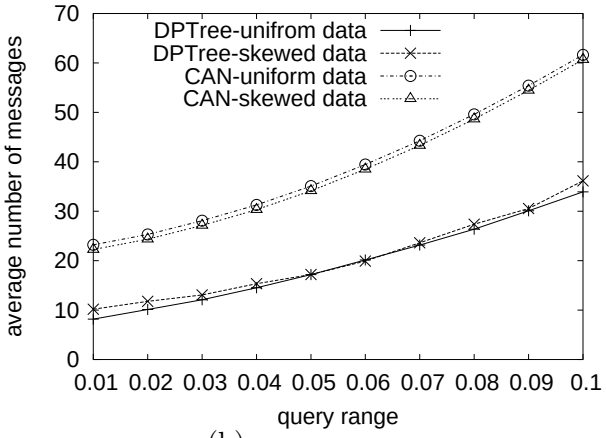
The query workload is generated as follows. We randomly select a point as the query point for point query or reference data object for range query and KNN query. For range queries, we vary the query radius from 0.01 to 0.1. For KNN queries, we vary the value of K from 1 to 10. We inject 1000 random queries into the system and the results presented below are the average results over these queries. Since the general trends observed under different network sizes are similar, we only show the results under network size as 1024.

4.5.3.1 Point Query

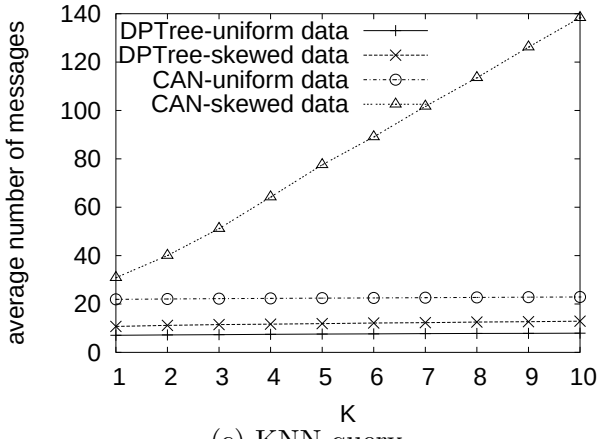
Figure 4.12(a) shows the result of point query with data distribution skewness varying from 0 to 1. The number of messages incurred by DPTree is always smaller than that of CAN. This confirms the efficiency of the tree-aware overlay and aggressive navigation algorithm, and the benefits of avoiding indexing dead space in DPTree as discussed in Section 4.2.1.1. In addition, the number of messages incurred by DPTree is insensitive to data distribution skewness. This confirms the adaptivity of DPTree to data distribution. It seems that the number of messages incurred by CAN also does not change significantly under different skewness values. However, this comes with the price



(a) point query



(b) range query



(c) KNN query

Fig. 4.12. Query performance of DPTree.

of uneven load distribution among peers due to the naive data space partition scheme adopted in CAN. In contrast, DPTree always achieves fair load distribution through the initial load-aware data placement using tree branch oriented distribution and the subsequent wavelet-based load balancing.

4.5.3.2 Range Query

Figure 4.12(b) shows the result of range query with query range varying from 0.01 to 0.1. For readability, we only present the result with data distribution skewness set to 0 and 1, respectively. As expected, the number of messages increases with the query range. In addition, the number of messages incurred by DPTree is always smaller than that incurred by CAN. This again confirms the efficiency of our tree-aware overlay and navigation algorithm, and the benefits of avoiding indexing dead space in DPTree.

4.5.3.3 KNN Query

Figure 4.12(c) shows the results of KNN query with the K value varying from 1 to 10. Similarly, we only present the result with data distribution skewness set to 0 and 1, respectively. From this figure, we see that the number of messages incurred by DPTree increases very slowly with the K value under both uniform data set and skewed data set, and the number of messages incurred by DPTree is always smaller than that of CAN. Furthermore, the number of messages incurred by DPTree under skewed data set is only slightly larger than that under uniform data set. In addition to validating the efficiency of our tree-aware overlay and navigation algorithm, this further confirms DPTree is adaptive to data distribution, which benefits KNN query processing. In contrast, the

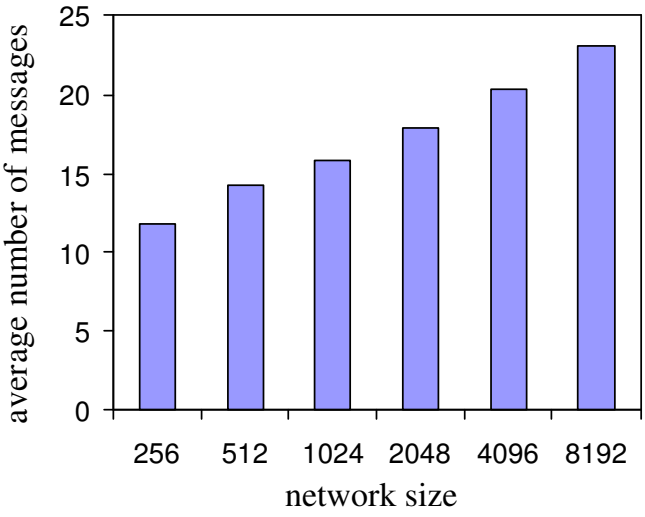
number of messages incurred by CAN under skewed data set increases rapidly with K values. This shows the lack of the ability to adapt to data distribution seriously degrades the performance of CAN under skewed data set.

4.5.4 Maintenance Overheads

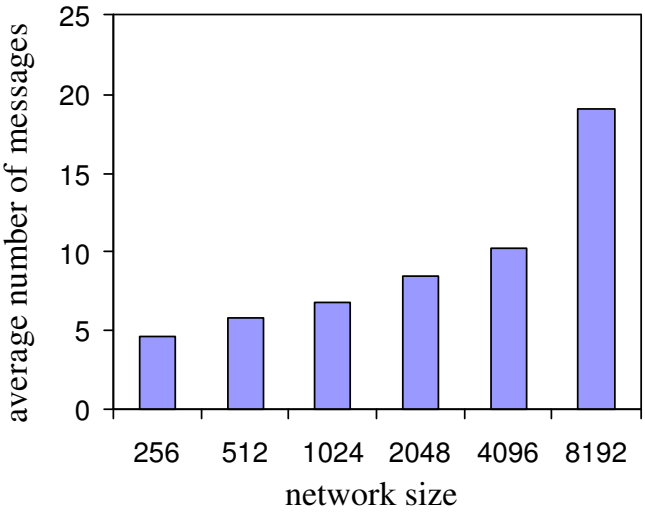
We evaluate two types of maintenance overheads (in terms of number of messages), i.e., overlay maintenance overheads incurred by peer join/leave/failure, and tree maintenance overheads incurred by data insertion/deletion. To evaluate the overlay maintenance overheads, we inject some random peer join/leave/failure events. The proportion of peer join and leave/failure is kept the same to make the size of the system constant. To evaluate the tree maintenance overheads, we first randomly insert half of the synthesized data objects (i.e., $50 \cdot N$), which are described earlier in Section 4.5.3, into the system. We then start to run the simulation by randomly injecting $25 \cdot N$ data insertion operations and $25 \cdot N$ data deletion operations into the system. At the end of the simulation, 50% of the data objects in the system is new compared to the data objects in the system at the beginning of the simulation.

4.5.4.1 Overlay Maintenance Overheads

Figure 4.13(a) shows the average number of messages incurred by a peer join/leave/failure event under different network sizes. From this figure, we can see that the average number of messages incurred upon peer join/leave/failure is rather low, i.e., around $2 \cdot logN$. This confirms our discussions in Section 4.3.4.1.



(a) cost of peer join/leave/failure



(b) cost of data insertion/deletion

Fig. 4.13. Maintenance overheads of DPTree.

4.5.4.2 Tree Maintenance Overheads

Figure 4.13(b) shows the average number of messages incurred by data insertion/deletion under uniform data distribution (the results obtained under skewed data distribution are very close to what are presented here. Thus, for presentation brevity and readers' interest, we only show the results under uniformly distributed data objects). This figure demonstrates that the number of messages incurred per data insertion/deletion is low, confirming our expectations as discussed in Section 4.3.4.2.

The set of experiments confirms that DPTree incurs rather low maintenance overheads upon peer join/leave/failure and data insertion/deletion, and thus it is adaptive to dynamic environments.

4.6 Summary

One of the fundamental challenges faced by peer-to-peer (P2P) systems is to efficiently support complex queries on multi-dimensional data objects. Although some works have studied this issue, they suffer from some fundamental limitations. We propose a framework, called distributed peer tree (DPTree), to efficiently support various types of queries on multi-dimensional data in P2P systems based on balanced tree indexes. DPTree combines a number of innovative ideas to achieve the efficiency: *tree branch oriented distribution*, *tree-aware overlay*, *aggressive navigation*, and *wavelet-assisted load balancing*. Through extensive performance evaluation, we demonstrate the superiority of DPTree over existing works on various aspects, including load balancing, routing, query processing, and maintenance.

DPTree provides a sound foundation where various data management tasks can be explored. We plan to exploit DPTree to investigate other types of complex queries and various data mining tasks in P2P systems.

Chapter 5

Identifying Frequent Items

As peer-to-peer (P2P) systems receive growing acceptance, the need of identifying 'frequent items' from the system usage data in such systems appears in a variety of applications, ranging from cache management to network attack detection. In this study, we define the problem of identifying frequent items (IFI) and propose an efficient in-network processing technique, called netFilter (in-network filtering) to address this important fundamental problem. netFilter consists of two phases: 1) candidate filtering: data items are grouped by filters into item groups and the aggregates of item groups are obtained to prune majority of infrequent items; and 2) candidate verification: the aggregates for the remaining candidate items are obtained to determine whether they are truly frequent items. We address various issues faced in realizing netFilter, including aggregate computation, candidate set optimization, and candidate set materialization. In addition, we analyze the performance of netFilter, derive the optimal setting mathematically, and discuss how to achieve the optimal setting in practice. Finally, we validate the effectiveness of netFilter through extensive simulations.

5.1 Introduction

Since there is no centralized administration in most P2P systems, various valuable system information and statistics from the usage data are distributed amongst peers,

making the operations or applications which need a global view of such information particularly challenging to implement. For example, a music marketing firm may want to find out which MP3 files have been downloaded more than 10,000 times in the past week. These kinds of statistics collected from the system can help designers and administrators to better understand/manage their systems and even detect on-going network attacks [40, 71]. In addition, they can help understand user behavior, which may be used for fine tuning the system performance. Since many applications require the statistics for frequently accessed files (e.g., MP3), frequently happened events (e.g., network attacks), frequently issued queries (e.g., keywords), and etc, which are typically represented as data items, in this paper we focus on developing efficient techniques for identifying *frequent items* in P2P systems.

Here we first define the problem of identifying frequent item (IFI) in P2P systems. Assume that a P2P system has N peers and the data set of interest $\mathcal{A}$ has n distinct data items (e.g., recording number of downloads of n pop songs). Peer i ($1 \leq i \leq N$) has a local item set $\mathcal{A}_i \subseteq \mathcal{A}$. Each item ($x$) in $\mathcal{A}_i$ has a *local value*, denoted by v_x^i, indicating the value associated with Item x at Peer i. Given a threshold value t, IFI identifies the frequent items whose *global values*, i.e., sum of the local values at all the peers in the system, is greater than t. Denoting the global value of Item x by v_x, i.e., $v_x = \sum_{i=1}^{N} v_x^i$, IFI can be formally defined as:

$$IFI(\mathcal{A}, t) = \{x | x \in \mathcal{A}, v_x \geq t\}.$$

Essential operations in a variety of P2P applications can be transformed into the problem of IFI as summarized in Table 5.1. In the table, we briefly summarize the operations, their corresponding applications, and the local item set maintained at a peer (Peer i with $1 \leq i \leq N$) and the local value associated with an item in transformation to the problem of IFI. All of these applications can be realized by identifying the frequent items that have global values greater than a given threshold value.

Table 5.1. Applications of IFI

Operations	Applications	Local item set (at Peer i)	Local value (of an item at Peer i)
Frequent keywords identification	Cache management	Keywords appearing in the queries issued by Peer i	Number of queries (among the queries issued by Peer i) that a keyword appears in
Frequent documents identification	Search technique design	Documents stored at Peer i	Number of replicas for a document maintained at Peer i
Frequently co-occurring keyword pairs identification	Query refinement	Pairs of keywords co-occurring in the queries issued by Peer i	Number of queries (among the queries issued by Peer i) that a pair of keywords co-occur in
Popular peers identification	Content mirroring Incentive mechanism	Peers that provide satisfactory results to the queries issued by Peer i	Number of queries (among the queries issued by Peer i) that a peer provides satisfactory results to
Frequently contacted peer pairs identification	Network topology optimization Social relationship analysis	Pairs of source/destination addresses that Peer i has seen in the packets passing through it	Number of packets (among the packets passing through Peer i) that are exchanged between a pair of source and destination
Large flow of traffic (to certain destination) identification	Denial of service attack detection [40]	Destination addresses that Peer i has seen in the packets passing through it	Size of the packets (among the packets passing through Peer i) that destine to an address
Frequent byte sequences identification	Internet worm detecting [71]	Byte sequences that appear in the traffic passing through Peer i	Number of packets (among all of the packets passing through Peer i) containing a byte sequence

Although the problem of IFI is prevailing in P2P systems, it is not yet addressed by previous works based on the authors' best knowledge. In this study, we investigate this problem in depth. Due to the complexity and diversity of items (keywords, documents, packets, and etc.), we assume that peers form an unstructured P2P systems where no global index structure is maintained for the items of interest.

Effectively identifying the frequent items in P2P systems is a non-trivial issue given the large scale of the system, the lack of central administration, and the large number of items. A naive solution is to collect the global value for each item and then output the items with global values exceeding the threshold as the results. However, the frequent items are usually a small portion of the whole set of items in the system. Therefore, collecting the global value for each item obviously is an overkill, which incurs excessive communication overheads.

In this study, we propose netFilter for identifying frequent items in P2P systems. netFilter consists of two phases: *candidate filtering* and *candidate verification*. To perform candidate filtering, items are grouped into disjoin item groups, and the aggregates[1] for these item groups are obtained from the system. These aggregates of item groups act as *filters* to prune majority of unqualified item groups from further consideration. A small subset of items is retained as the *candidates*. Candidate verification is then invoked to obtain the global values for these candidates to verify whether they truly satisfy the threshold condition. The set of frequent items identified through netFilter is precise in

[1]The aggregate for an item or a set of items refers to the combined value for the item(s) from different peers in the system. For instance, the aggregate for Item x (a_x) is $\sum_{i \in \mathcal{P}} v_x^i$ ($\mathcal{P} \subseteq \{1, 2, ..., N\}$). When $\mathcal{P} = \{1, 2, ..., N\}$, the aggregate for x is also the global value of x. In such cases, we use global value and aggregate interchangeable if the context is clear.

terms of the following two aspects. First, there is no *false positives* (infrequent items that are incorrectly reported as frequent items) or *false negatives* (frequent items that are not identified). Second, the reported global values of the frequent items are precise.

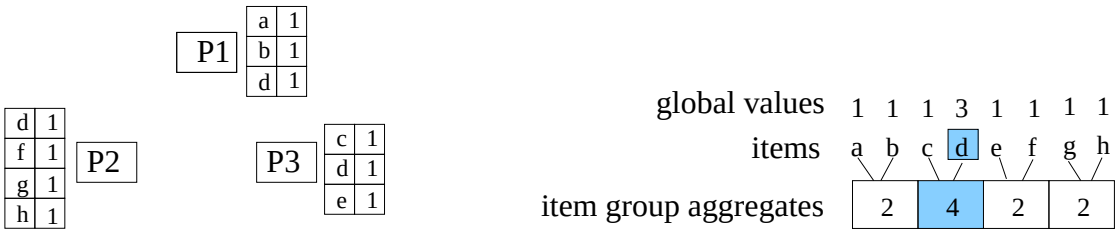


Fig. 5.1. An illustrative example of netFilter.

The example in Figure 5.1 illustrates the basic concept behinds netFilter. The system has three peers (P1, P2 and P3). Their local item sets are depicted in the tables besides the peers, where the left and right columns represent the identifiers of the items and their local values, respectively. The threshold value (t) is 3. For illustration purpose, we assume the eight items are assigned to four item groups as follows: Items a and b are assigned to Item-group 1, Items c and d are assigned to Item-group 2, and so on. During candidate filtering, we collect the aggregate for each item group. Among the four item groups, only Item-group 2 has its aggregate above the threshold value. Thus, only the two items (Items c and d) belonging to Item-group 2 are retained as the candidates. During candidate verification, the global values for Items c and d are obtained as 1 and 3, respectively. Thus, Item d is returned as the result.

While the above netFilter algorithm seems straightforward conceptually, three important issues need to be addressed:

- **Aggregate computation**, i.e., how to obtain the aggregate for an item or a set of items from all the peers in the system.
- **Candidate set optimization**, i.e., how to obtain a good candidate set with majority of them truly satisfying the threshold condition.
- **Candidate set materialization**, i.e., how to generate or materialize the candidate set for candidate verification with no peer having a global view of the complete set of items in the system. We propose techniques to address above issues. Moreover, to optimize the performance of netFilter, we derive the optimal setting mathematically, and discuss how to achieve the optimal setting in practice. Finally, we evaluate the effectiveness of netFilter through extensive simulations.

In summary, the primary contributions of this work are four-fold:

1. We identify and formally define the problem of IFI, and discuss how the need of addressing IFI exists in a variety of applications in P2P systems.
2. To address IFI, we propose an efficient in-network processing approach, netFilter, which is a two-phase technique consisting of candidate filtering and candidate verification. We address various issues faced in realizing netFilter, including aggregate computation, candidate set optimization and candidate set materialization.
3. We analyze the performance of netFilter, derive the optimal setting mathematically, and discuss how to set netFilter optimally in practice.

4. We conduct extensive performance evaluation to demonstrate the effectiveness of our proposal.

The rest of this chapter is organized as follows. The details of netFilter and its analysis are presented in Section 5.2 and Section 5.3, respectively. The evaluation of netFilter is presented in Section 5.4. Finally, we summarize this study and point out the relevant issues that can be explored further in Section 5.5.

5.2 In-Network Filtering

In the previous section, we have introduced the basic idea of netFilter. Here we look into the detailed design of netFilter, discuss related issues and propose our solutions.

netFilter consists of two phases: candidate filtering and candidate verification. We summarize the basic operations of netFilter in Algorithm 7 where candidate filtering and candidate verification correspond to Lines 1-3 and Line 4 in Algorithm 7, respectively.

Algorithm 7 Algorithm for netFilter.

1: Partition items into item groups.
2: Obtain aggregate for each item group.
3: Prune unqualified item groups (with aggregates below the threshold), and obtain the items in the remaining item groups as candidates.
4: Obtain the global values for the candidates, and return the items with global values above the threshold as the results.

While the above netFilter algorithm seems straightforward conceptually, three important issues need to be addressed carefully:

- *Aggregate computation.* Both candidate filtering and candidate verification require computing the aggregates (corresponding to the item groups during candidate filtering and candidate items during candidate verification, respectively). Since each peer only has the local values for a partial set of items, peers have to collaborate with each other to obtain the global value for an item or a set of items.

- *Candidate set optimization.* The effectiveness of netFilter relies on the 'goodness' of the candidate set, i.e., how many items in the candidate set truly satisfy the threshold, and how many do not. The latter are false positives in the candidate set. How to reduce the number of false positives in the candidate set is an important issue. Note that the false positives here refer to infrequent items in the candidate set. There is no false position in the set of frequent items returned after candidate verification.

- *Candidate set materialization.* After candidate filtering, the candidate set should be materialized, and then disseminated to the system for candidate verification. Given the item groups satisfying the threshold, a peer would be able to produce the complete candidate set if it had the knowledge of the complete set of items in the whole system. Unfortunately, each peer normally only has a partial set of items in its local item set, i.e., $\mathcal{A}_i \subset \mathcal{A}$.

Figure 5.2 illustrates the whole procedure of netFilter and the issues involved in different phases of netFilter. Our strategies to address these issues are detailed in the subsequent subsections.

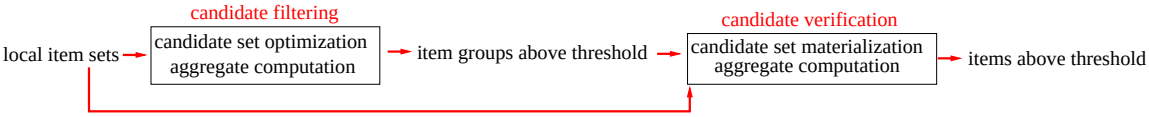


Fig. 5.2. The procedure of netFilter.

5.2.1 Aggregate Computation

To obtain the aggregate for an item or a set of items, two possible approaches are: 1) gossip-based aggregate computation (called *gossip aggregation* in short) [20, 68, 131], where peers exchange their current aggregates with their neighbors till the aggregates (almost) converge to the global values, 2) hierarchy-based aggregate computation (called *hierarchical aggregation* in short) [20, 101], where peers form into a hierarchical infrastructure and pass the aggregates in a bottom up fashion along the hierarchy. The gossip aggregation mechanisms proposed in the literature require multiple ($O(logN)$) rounds of communication among peers till the aggregates (almost) converge. Among these proposals, [68] is vulnerable to peer failure. Compared to [68], proposals in [20, 131] are more fault-resilient but only obtain approximate aggregates. On the other hand, hierarchical aggregation normally requires one or two rounds of communications among peers to obtain the precise aggregates, even though it is more sensitive to the leave or failure of the peers at the higher levels of the hierarchy.

In this study, due to the aforementioned limitations of gossip aggregation and the simplicity and low communication overhead of hierarchical aggregation, we follow the basic principle of hierarchical aggregation to design our aggregate computation. The

major concern with forming a hierarchy in P2P systems is the frequent update of the hierarchy upon peer join/leave/failure. To avoid this, we only recruit peers that are more stable, i.e., being online for a longer time, to perform netFilter where other peers forward their local item sets to one of these peers participating in netFilter. The assumption of existence of stable peers is reasonable as discussed in [121]. Nevertheless, the proposed technique is applicable to a well-designed gossip aggregation that is left as our future work. In the following discussions, if unspecified otherwise, peers refer to the peers participating in netFilter.

Proceeding to the details of aggregate computation, we first discuss how peers form the hierarchy for aggregate computation. We then discuss how to obtain the aggregates along the hierarchy and how to update the hierarchy upon peer join/leave/failure.

5.2.1.1 Forming Hierarchy

Peers form a hierarchy based on the principle of breadth-first-search (BFS). Basically, each peer (Peer i with $1 \leq i \leq N$) becomes a node in the hierarchy at depth $d(i)$ as follows where $d(i)$ is the length of the shortest path (in terms of logical hops) from the *root* of the hierarchy to this peer. A designated peer is first chosen as the root node of the hierarchy (with depth as 0). This designated peer could be a randomly selected peer, the most stable peer, or a peer that is close to the center of the network. In this study, we choose a peer randomly as the root node and leave other options for future exploration. The immediate neighbors of the root node become the nodes at depth 1 in the hierarchy. They set the root node as their upstream neighbor and set themselves as the downstream neighbors of the root node. The immediate neighbors of the peers

at depth 1 that are not yet included in the hierarchy become the nodes at depth 2 in the hierarchy. These peers' upstream neighbors and downstream neighbors are set up accordingly. This process continues till we reach the peers whose immediate neighbors are all already included in the hierarchy, i.e., these peers have no downstream neighbors. Such peers are the *leaf nodes* of the hierarchy. The nodes that are neither the root or leaf nodes are called the *internal nodes* of the hierarchy. We associated each peer in the hierarchy with its depth so that the messages can be propagated along the hierarchy properly.

Figure 5.3 illustrates one example of the hierarchy. The peers depicted by the squares are the peers participating in netFilter. These peers form a hierarchy with four levels. Other peers depicted by the dark circles connect to one of those peers participating in netFilter and report their local item sets to them.

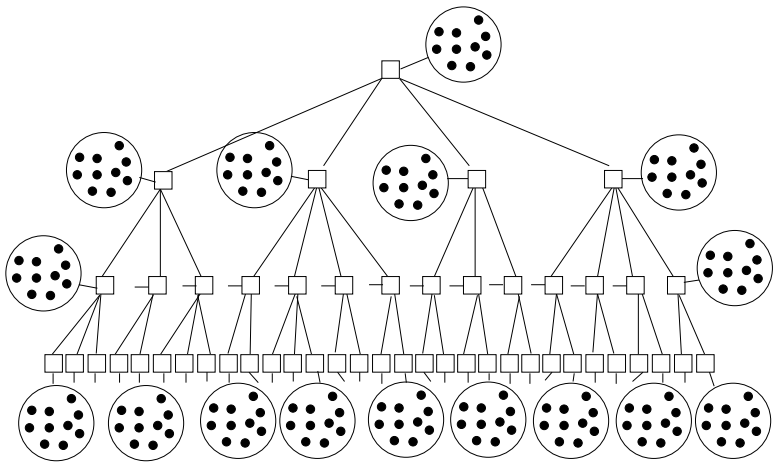


Fig. 5.3. An illustrative example of aggregate computation.

Note that multiple peers might simultaneously issue requests for identifying frequent items with different threshold values. Instead of forming a separate hierarchy and invoking individual netFilter for each request, we use the following technique. The requests from different peers are first forwarded to the root node, which then invokes netFilter with the threshold value t set to the minimum threshold value among all the requests that it receives. The returned result set, i.e., the set of frequent items with their global values above the minimum threshold value (among all the requests), is the superset of the result sets for the requests with larger threshold values. We can simply form the proper result set for each request from this superset and forwards it to the corresponding peer requesting such a frequent item set. In this way, multiple requests from different peers are sharing the same hierarchy and netFilter process.

5.2.1.2 Computing Aggregates

To compute the aggregate for an item or a set of items, the peers corresponding to the leaf nodes propagate the corresponding local values to their upstream neighbors, respectively. A peer representing an internal node merges its own local value for the item(s) under consideration with the values received from its downstream neighbors, and then forwards the merged result to its upstream neighbor. Eventually, the root node has the final aggregate (global value) for the item(s). Note the communication cost incurred at the peers located at the higher levels of the hierarchy is almost the same as that incurred at the peers located at the lower levels of the hierarchy. Thus, this aggregate computation mechanism does not result in performance bottlenecks at the higher levels of the hierarchy.

5.2.1.3 Updating Hierarchy

In most deployed P2P systems, peers exchange heartbeat messages with their neighbors periodically to inform the aliveness among each other. Here we modify these heartbeat messages slightly by including a DEPTH counter, indicating the depth of the message sender in the hierarchy, so that they can be used to update the hierarchy upon peer join/leave/failure. To accommodate a new peer participating in netFilter, the upstream neighbor and downstream neighbors of the newly joined peer are set up similarly as described in Section 5.2.1.1. The repair of the hierarchy upon peer leave/failure is more complex. Upon detecting that the upstream neighbor leaves or fails by the lack of heartbeat messages from this neighbor for a predefined time interval, a peer invokes the repair of the hierarchy as follows. It first sets its depth in the hierarchy as ∞. In addition, the downstream neighbors of this peer are recursively informed to set their depth as ∞. When a peer with depth as ∞ receives a heartbeat message from a neighbor (P_i) with depth ($d(P_i)$) less than ∞, it becomes a node at depth ($d(P_i)$+1) in the hierarchy by setting P_i as its upstream neighbor and setting itself as one of the downstream neighbors of P_i.

5.2.2 Candidate Filtering

To perform candidate filtering, items are first partitioned into disjoint item groups. The aggregates for item groups are obtained through aggregate computation as presented in Section 5.2.1.2. These aggregates of the item groups act as filters to filter out unqualified items (i.e., those in the item groups with aggregates below the threshold) and retain the items in the item groups with aggregates above the threshold as the candidates. We

call the items with their global values below the threshold as *light items* and the items with their global values above the threshold as *heavy items*. Similarly, we call the item groups with their aggregates below (above) the threshold as *light (heavy) item groups*. In the following, we first discuss how items are partitioned into item groups. We then discuss strategies for candidate set optimization.

5.2.2.1 Item Partitioning

Given the fact that a peer only has a partial set of items in the system, a solution to partition items into item groups that requires a priori knowledge of the complete set of items in the system would mandate heavy coordination among peers. Here a natural solution for item partitioning is *hashing*. Each of the n items is mapped to one of the g item groups through a hashing function $h(x) : \mathcal{A} \rightarrow \mathcal{B}$, where $\mathcal{A}$ and $\mathcal{B}$ are the set of items and set of item groups, respectively. g is referred to as the *filter size*. Each peer obtains the local values for the item groups as follows. It assigns each of its local items to one of the g item groups and increases the local value of the corresponding item group accordingly. For instance, if an item x of Peer i is mapped to item group B_j $(1 \leq j \leq g)$, the local value for B_j at Peer i is increased by v_x^i.

5.2.2.2 Candidate Set Optimization

Since the aggregate of an item group is the summation of the global values of all the items in the item group, some items in the heavy item group might have their global values below the threshold but are retained as the candidates, i.e., these candidates are

false positives. There are two types of false positives in the candidate set caused by two different reasons:

1. **Homogeneous false positive** The global values of all the items in a heavy item group are below the threshold but their summation exceeds the threshold (the heavy item group consists of light items only).

2. **Heterogeneous false positive** Some items with their global values below the threshold are grouped into the same item group with other items with global values above the threshold (the heavy item group consists of some light items and some heavy items).

Two strategies are proposed to minimize the number of false positives in the candidate set:

Strategy 1: Setting the filter size properly. Small g (filter size) incurs low communication overhead during candidate filtering but results in a large number of false positives in the candidate set. Large g reduces the number of false positives in the candidate set but incurs high communication overhead during candidate filtering. Therefore, we should set g to a value achieving a nice balance between the number of false positives and the communication overhead incurred during candidate filtering.

Strategy 2: Applying multiple filters. To further reduce the number of false positives, we apply multiple (f) filters. Each filter is defined by a hash function, i.e., $h(x)^i$ $(1 \leq i \leq f) : \mathcal{A} \rightarrow \mathcal{B}^i$, which maps one of the n items to the set of item groups $\mathcal{B}^i$. An item becomes a candidate only if it is retained by all of the f filters, i.e., each of the f item groups that it belongs to is heavy.

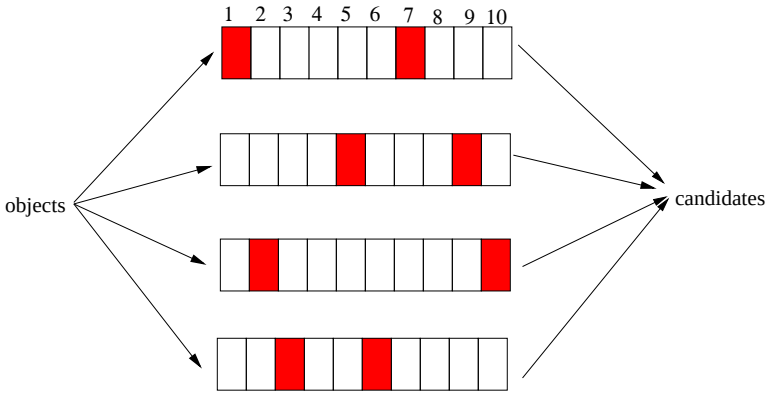


Fig. 5.4. An illustrative example of multiple filters.

We use Figure 5.4 to illustrate an example for multiple filters. We apply four filters and obtain four sets of aggregates with one corresponding to each filter. The shaded rectangles indicate heavy item groups and blank rectangles indicate light item groups. Suppose an item x is hashed by the four filters to Item-groups 1, 5, 2, and 3, respectively. Another item y is hashed to Item-groups 7, 5, 10, and 1, respectively. Since all of the four item groups that Item x belongs to are heavy, Item x becomes a candidate. On the other hand, Item y is pruned since one of the item groups that Item y belongs to (Item-group 1 according to Filter 4) is light.

Note that appropriate settings of f and g are crucial to the effectiveness of net-Filter. An analysis that derives the optimal settings for them will be given in Section 5.3.

5.2.3 Candidate Verification

In order to perform candidate verification, we need to address candidate set materialization, i.e., generating the candidate set so that peers can compute the global values for these candidates. If the root of the hierarchy has a complete view of all the items in the system, it can materialize the candidate set by examining the heavy item groups obtained from candidate filtering. However, in practice, the root may not have the complete set of items in the system. One possible solution is to perform another aggregate computation to obtain the complete set of items in the system. Given the large number of items, this aggregate computation is almost as costly as the naive approach, and thus it is not feasible.

We observe that with the list of heavy item groups provided, each individual peer can determine which of the items in its local item set are candidates. Thus, each individual peer can materialize part of the candidate set and obtains a partial candidate set. During aggregate computation, these partial candidate sets are merged implicitly. When we reach the root of the hierarchy, we obtain the complete set of candidates and their global values. Based on this observation, we propose to perform candidate set materialization and aggregate computation for the candidates (called *candidate aggregate computation*) in an integrated fashion as illustrated in Algorithm 8. Eventually, the root node has the global value for each item in the complete set of candidates exactly. It then outputs those items with global values above the threshold as the results.

Algorithm 8 Algorithm for candidate set materialization and candidate aggregate computation.

1: The root propagates the identifiers of the heavy item groups downwards along the hierarchy recursively.
2: Upon receiving the identifiers of the heavy item groups, a peer materializes the candidate set according to its local item set and obtains a partial candidate set.
3: The peers corresponding to the leaf nodes of the hierarchy start to propagate the pairs of $\langle identifier, local\_value \rangle$ of the items in its partial candidate set to their upstream neighbors.
4: A upstream neighbor merges its partial candidate set with the partial candidate sets received from the downstream neighbors, updates the values for the candidates accordingly, and propagates the merged result upwards along the hierarchy.

5.3 Analysis of netFilter

In the following, we first derive the cost model for netFilter. From the cost model, we derive the optimal setting for the filter size g (the number of item groups per filter) and the number of filters f. We then discuss how to achieve the optimal settings for netFilter in practice. For comparison, we also derive the cost model for the naive approach where the host nodes forward their local item sets along the hierarchy. As we explained before, we recruit a set of stable peers to form the hierarchy. Thus, we expect that the events of join/leave/failure of the peers in the hierarchy are rare. In addition, both of netFilter and the naive approach under consideration are based on hierarchical aggregation. Therefore, we ignore the cost incurred by hierarchy formation and update.

Table 5.2 lists the symbols used in the following discussions. Note that threshold t can be either an exact value (e.g., 1000) or an expression, e.g., $t = \delta \cdot v$, where v is the summation over all of the local values of all the items in the system (i.e., $v = \sum_{j=1}^{n} \sum_{i=1}^{N} v_j^i$). Thus, $t = \delta \cdot v$ indicates t is δ fraction of v. We call δ as the *threshold ratio*. Without loss of generality, we assume that t is expressed as $\delta \cdot v$ in the following discussions. In

Table 5.2. Symbols used in the analysis of netFilter.

Symbols	**Descriptions**
N	Number of peers in the network
n	Number of distinct items in the system
o	Number of distinct items in the local item set of a peer
t	Threshold value
δ	Threshold ratio ($t = \delta \cdot v$)
$\bar{v}$	Average global value of items
$\bar{v}_{light}$	Average global value of light items
r	Number of heavy items
w	Number of heavy item groups
fp	Number of false positives
b	Number of downstream neighbors per peer
h	Height of the hierarchy
g	Size of filter (number of item groups per filter)
f	Number of filters
s_a	Size of the value representing an aggregate
s_g	Size of the identifier of an item group
s_i	Size of the identifier of an item

addition, we assume that we have the values of v and N through simple aggregate computation. To obtain v, each peer contributes a single value, i.e., the summation over all of the local values of the items in its local item set, to the final aggregate. To obtain N, each peer contributes the single value of 1 to the final aggregate. The aggregate computation for v and N can be combined with other aggregate computation since they only need to propagate one single value along the hierarchy, respectively.

Since the main focus of this paper is to identify frequent items at a minimum communication overhead, we use the following performance metric in the analysis:

- **Communication cost:** the average number of bytes propagated per peer.

Communication cost includes the cost incurred by candidate filtering and candidate verification. The cost incurred by candidate verification itself includes the cost to disseminate the identifiers of the heavy item groups to the system (so that peers can materialize the candidate set) and the cost to compute the aggregates for the candidates. We refer the cost incurred by candidate filtering, dissemination of the identifiers of the heavy item groups, and candidate aggregate computation as *candidate filtering cost*, *candidate dissemination cost* and *candidate aggregation cost*, respectively.

5.3.1 Cost Model for netFilter

During candidate filtering, a peer propagates the aggregate for each item group. Thus, the candidate filtering cost is $s_a \cdot f \cdot g$. After candidate filtering, the root propagates the identifiers of the heavy item groups to the system. Thus the candidate dissemination cost is $s_g \cdot f \cdot w$. To obtain the global value for a candidate item, each peer propagates the pair of $\langle identifier, value \rangle$ for this item. Therefore, the candidate verification cost

is $(s_a+s_i)\cdot(r+fp)$. We can then derive the total cost incurred by netFilter, denoted by C_{filter}, as

$$C_{filter} = s_a \cdot f \cdot g + s_g \cdot f \cdot w + (s_a + s_i) \cdot (r + fp). \tag{5.1}$$

5.3.2 Cost Model for the Naive Approach

We derive the communication cost incurred by the naive approach, denoted by C_{naive}, as:

$$(s_a + s_i) \cdot o \leq C_{naive} \leq (s_a + s_i) \cdot o \cdot h \cdot b. \tag{5.2}$$

Note the above result may seem surprising since intuitively we expect that the communication cost incurred by the naive approach is $O(n{\cdot}N)$. The reason that this cost is smaller than $O(n{\cdot}N)$ is that a peer only needs to propagate the pairs of $\langle identifier, value\rangle$ for the items with nonzero values (these items are the union of the items in its own local item set and the items propagated from its downstream neighbors). The number of items propagated by the peers at the lower levels of the hierarchy is close to o, the size of their local item sets, and the number of items propagated by the peers at the higher levels of the hierarchy increases up to n. Averaging over all the peers, the number of items that a peer needs to propagate is in the order of o, resulting in the above Formula.

5.3.3 Optimal Setting for the Size of Filters (g)

Given the average global value of light items in the system ($\bar{v}_{light}$), in order to avoid homogeneous false positives, no more than $\frac{t}{\bar{v}_{light}}$ items should be hashed to an item group. Therefore, g should be greater than $\frac{n\cdot\bar{v}_{light}}{t}$. Since $t = \delta{\cdot}v$ and $v = n{\cdot}\bar{v}$, we

obtain the optimal setting for g (denoted by g_{opt}) as

$$g_{opt} = c + \frac{\bar{v}_{light}}{\delta \cdot \bar{v}} \tag{5.3}$$

with c as a small positive constant.

5.3.4 Optimal Setting for the Number of Filters (f)

We argue that the optimal f setting (denoted by f_{opt}) is the one that can make the number of heterogeneous false positives (denoted by fp_2) as small as $\frac{g \cdot s_a}{s_a + s_i}$. At this point, netFilter incurs the minimum communication overhead, denoted by C_{opt}. We first derive the formula for fp_2 before we explain the reason.

The number of heterogeneous false positives (fp_2) is the total number of light items multiplied by the probability of heterogeneous false positives (denoted by p_{fp2}). The total number of light items is $(n - r)$. p_{fp2} equals the probability that a light item resides in the same item groups with a heavy item for all the f filters. In the following, we derive p_{fp2}. We first consider a specific filter i with $1 \leq i \leq f$. The probability that a heavy item resides in a specific item group for filter i is $\frac{1}{g}$, and the probability that a light item does not reside in the same item group with this heavy item for filter i is $1 - \frac{1}{g}$. From this, we can then derive the probability that a light item does not reside in the same item group with any of the r heavy items for filter i is $(1 - \frac{1}{g})^r$. Therefore, the probability that a light item does reside in the same item group with at least one of the r heavy items for filter i is $1 - (1 - \frac{1}{g})^r$. We can then derive p_{fp2} (equal to the probability that a light item resides in the same item groups with a heavy item for all

the f filters), i.e., $p_{fp2} = (1 - (1 - \frac{1}{g})^r)^f$. Thus,

$$fp_2 = (n - r) \cdot p_{fp2} = (n - r) \cdot \left(1 - \left(1 - \frac{1}{g}\right)^r\right)^f. \tag{5.4}$$

We now explain why we can achieve the minimum communication cost ($Copt$) with the optimal f setting (f_{opt}) as the one that makes fp_2 as small as $\frac{g \cdot s_a}{s_a + s_i}$. Since the number of heavy item groups normally is much smaller than the filter size (i.e., $w << g$), the candidate dissemination cost is much smaller than the candidate filtering cost and is ignored in the following derivation. With the proper setting on g as discussed above, we avoid homogeneous false positives, and thus $fp = fp_2$. Therefore, Formula 5.1 can be simplified as follows:

$$C_{filter} \approx s_a \cdot f \cdot g + (s_a + s_i) \cdot (r + fp_2). \tag{5.5}$$

According to the above formula, when we increase f to $f_{opt}+1$, we increase the candidate filtering cost by $g \cdot s_a$. However, the decrease on fp_2 will be at most $\frac{g \cdot s_a}{s_a + s_i}$ (since $fp_2 = \frac{g \cdot s_a}{s_a + s_i}$ when $f = f_{opt}$, and $fp_2 >= 0$ when $f = f_{opt}+1$). We can then derive the candidate aggregation cost decreases at most by $(s_a + s_i) \cdot \frac{g \cdot s_a}{s_a + s_i} = g \cdot s_a$. Therefore, the communication cost incurred by netFilter when $f = f_{opt}+1$ is not smaller than C_{opt}. Increasing f further will incur communication cost greater than that incurred when $f = f_{opt}+1$.

On the other hand, when we decrease f to $f_{opt}-1$, the candidate filtering cost decreases $g \cdot s_a$. According to Formula 5.4, fp_2 increases y times with $y = \frac{1}{1-(1-1/g)^r}$,

i.e., the increase on fp_2 is $\frac{g \cdot s_a}{s_a + s_i} \cdot (y\text{-}1)$. In most cases y is greater than 2. We can then derive the increase on the candidate aggregation cost is greater than $(s_a + s_i) \cdot \frac{g \cdot s_a}{s_a + s_i} = g \cdot s_a$ in general. Therefore, the communication cost incurred when $f = f_{opt} - 1$ is greater than C_{opt}. Similarly, decreasing f further will increase the communication cost further beyond that incurred when $f = f_{opt} - 1$. From above discussion, we can see that the communication cost is the minimum when $f = f_{opt}$. Replacing fp_2 by $\frac{g \cdot s_a}{s_a + s_i}$ in Formula 5.4, we can derive

$$f_{opt} = \left\lceil log_{\frac{1}{1-(1-\frac{1}{g})^r}} \frac{(s_a + s_i) \cdot (n - r)}{g \cdot s_a} \right\rceil . \tag{5.6}$$

5.3.5 Setting netFilter Optimally In Practice

From Formulae 5.3 and 5.6, we can see that in order to obtain the optimal setting for g and f, we need to know $\bar{v}_{light}$, $\bar{v}$, n, and r (with δ, s_a, and s_i given). In the following, we explain how to obtain the estimation for these values in practice.

We perform random sampling to obtain $\bar{v}$, $\bar{v}_{light}$ and n. An intuitive approach for random sampling is to collect statistics from the peers within one or two hops away from the root of the hierarchy. However, it is possible that there exists some correlation among the local values of the items in a close neighborhood (within one or two hops), which may have ill effect on random sampling. Therefore, we propose to randomly select a few branches in the hierarchy, e.g., the peers along the path from the root to the leaf nodes, for sampling. Each of the sampled peers randomly selects some of the local items from its local item set, for which the aggregates are collected from these sampled peers.

Assume there are x distinct items in the sampled item set, and each of such items has an aggregate from the sampled peers as v'_i $(1 \leq i \leq x)$. We estimate the global value

of each of such items as $\tilde{v}_i = \frac{v'_i \cdot v}{\sum_{i=1}^{x} v'_i}$ (as mentioned in the beginning of this section, v is the summation over all of the local values of all the items in the system, and can be obtained by simple aggregate computation). We then derive the estimation for $\bar{v}_{light}$, denoted by $\tilde{v}_{light}$, as

$$\tilde{v}_{light} = \frac{\sum_{\substack{1 \le i \le x \\ \tilde{v}_i < t}} v'_i}{\sum_{\substack{1 \le i \le x \\ \tilde{v}_i < t}} 1}, \tag{5.7}$$

and the estimation for $\bar{v}$, denoted by $\tilde{v}$, as

$$\tilde{v} = \frac{\sum_{1 \le i \le x} v'_i}{x}. \tag{5.8}$$

To obtain an estimation for n, the sampled peers propagate and merge their local item sets along the hierarchy similarly as aggregate computation. Eventually, the root obtains the set of distinct items from the sampled peers. Denoting the number of distinct items obtained from the sampled peers by n', the estimation for n (denoted by $\tilde{n}$) is

$$\tilde{n} = \frac{n' \cdot N}{N_s} \tag{5.9}$$

where N and N_s are the total number of peers in the system and the number of sampled peers, respectively. As described in the beginning of this section, N can be obtained through simple aggregate computation. We can obtain the value of N_s similarly from the sampled peers.

The value of r is difficult to obtain. We perform some empirical experiments and find that r normally is a small constant for a variety of data distributions and δ

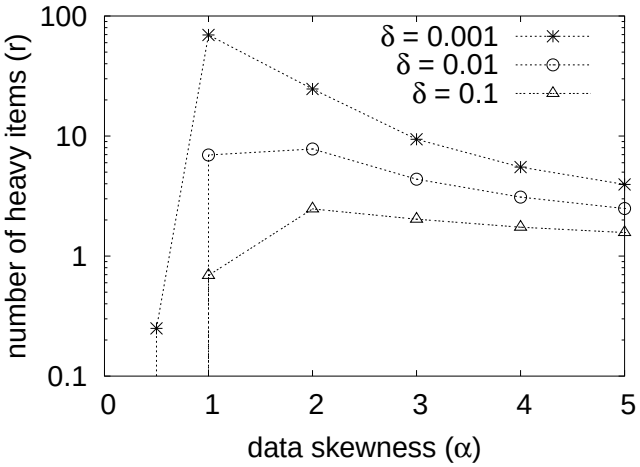


Fig. 5.5. Number of heavy items (n = 10^6).

values. As demonstrated in previous studies, the global values of items usually follow zipf-distribution or zipf-like distribution [10, 40]. Therefore, we obtain the r values for a set of data with their values following zipf-distribution, which provides a ready parameter (α) controlling the skewness/uniformity of the data distribution. When α is 0, items have randomly uniformly distributed values. When α is large, the value distribution is skewed where majority of the items have small values and a few items have very large values. Figure 5.5 shows the values of r with varying α values and δ values (the number of items is 10^6). From this figure, we can see that r is only a very small fraction of the total number of items. For instance, the maximum r value is around 70 when $\delta = 0.001$ and $\alpha = 1$.

5.4 Performance Evaluation

We move on to evaluate netFilter's effectiveness using extensive simulations. We first tune the performance of netFilter by varying the size of filters (g) and the number of filters (f). We then evaluate the performance of netFilter under different settings of data skewness (α) and threshold ratios (δ). For comparison, we also implement the naive approach and evaluate its performance[2]. In the following, we first describe the simulation setup and performance metrics. We then present the details of the simulation results.

Table 5.3. Parameters used in the simulations of netFilter

Symbols	**Descriptions**	**Default**
N	Number of peers in the network	1000
n	Number of distinct items in the system	10^5
o	Number of distinct items in the local item set of a peer	1000
δ	Threshold ratio	0.01
α	Skewness of zipf distribution	1
b	Number of downstream neighbors per peer	3
s_a	Size of the value representing an aggregate	4 bytes
s_g	Size of the identifier of an item group	4 bytes
s_i	Size of the identifier of an item	4 bytes

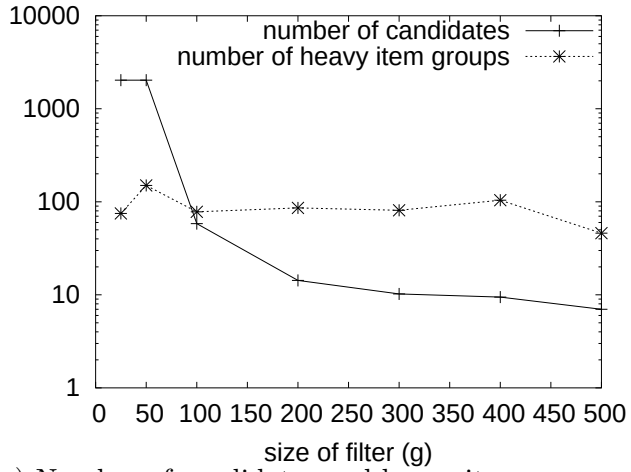
[2]As mentioned in Section 2.3, some studies obtain an approximate set of frequent items and the communication cost incurred is $O(\frac{a}{\epsilon})$ where a is either a constant or proportional to $log(n)$ and ϵ is the given error tolerance. We do not compare with these techniques since the result set returned in these studies is approximate, which is different from our focus here. But the rule of thumb is that when the given error tolerance is very small, the communication cost incurred by these techniques is even higher than the cost incurred to obtain a precise set of frequent items using our technique.

The simulation parameters and their default values (unless otherwise stated) are given in Table 5.3. Most of these parameters are self-explanatory. More details for some of the parameters are given as follows. We use zipf distribution (with data skewness parameter α) to model the distribution of values for items. We generate $10{\cdot}n$ instances of these items with their frequencies (global values) following zipf-distribution. We then randomly distribute these $10{\cdot}n$ items to the N nodes. Therefore, the number of items on each peer is $\frac{10{\cdot}n}{N}$ (the default is $\frac{10{\cdot}10^5}{1000} = 1000$). Without loss of generality, we use 4-bytes integers to represent the aggregate values, identifiers of item groups, and identifiers of items (i.e., $s_a = s_g = s_i = 4$ bytes).

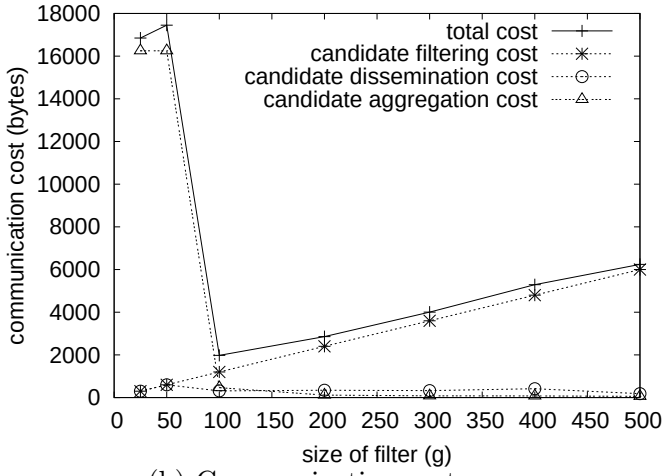
We use the total communication cost defined in Section 5.3 as the main performance metric. To better understand the performance of netFilter, we distinguish the candidate filtering cost, candidate dissemination cost and candidate aggregation cost at some points. If unspecified otherwise, the communication cost refers to the total communication cost, which is the lumped sum of these three individual costs.

5.4.1 Effect of the Filter Sizes

We conduct experiments to evaluate the effect of the filter sizes on the performance of netFilter. Figure 5.6(a) shows the average number of candidates propagated per peer during candidate verification and the number of heavy item groups with g varying from 25 to 500 (the number of filters is set to 3). For readability, the y-axis is on log scale. From this figure, we can see that when the filter size is very small (<100), the filtering performance is poor. In fact, at this range of filter sizes, none of the items are pruned. Thus, the candidate verification performs similarly as the naive approach, and



(a) Number of candidates and heavy item groups



(b) Communication cost

Fig. 5.6. Effect of filter sizes on netFilter.

the average number of candidates propagated per peer during candidate verification is in the order of the size of local item set[3]. When the filter size increases, the filtering performance improves significantly, resulting in significant reduction on the number of candidates propagated per peer. For instance, when the filter size increases from 50 to 100, the number of candidates propagated per peer reduces from 2030 to 58. When the filter size increases further, the decrease on the number of candidates propagated per peer becomes less significant.

The major trend observed from the number of heavy item groups is that it increases initially, then decreases. When the filter size is small ($<$100), the number of items assigned to an item group is very large, which makes all item groups heavy. Thus, at this range of filter sizes, the number of heavy item groups increases with the filter size. When the filter size becomes larger ($\geq$ 100), the number of items assigned to an item group becomes less, resulting in less number of heavy item groups.

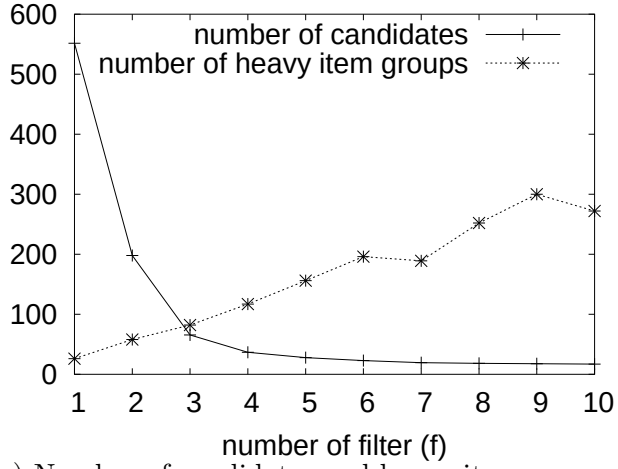
Figure 5.6(b) shows the communication cost incurred by netFilter under the same setting as for Figure 5.6(a). We also plot the candidate filtering cost, candidate dissemination cost and candidate aggregation cost in the figure. The candidate filtering cost increases linearly with the filter size. The candidate aggregation cost and candidate dissemination cost are proportional to the number of candidates propagated per peer and the number of heavy item groups as displayed in Figure 5.6(a), respectively. This confirms our analysis in Section 5.3.1.

[3]The reason that this number is smaller than the total number of candidates (in this case, n) is the same as explained in Section 5.3.2.

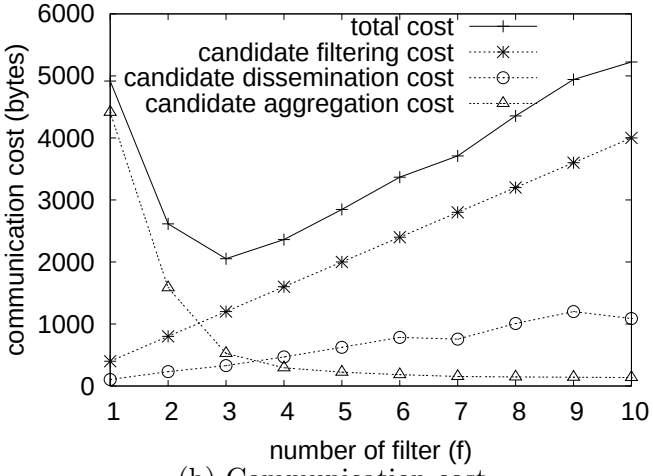
With above discussion, we can easily explain the plot for the total cost in Figure 5.6(b). The total cost initially increases slightly, then decreases significantly, followed by an increase again. When the filter size is very small (<100), the total cost is dominated by the candidate aggregation cost. This explains the initial increase of the total cost (since candidate aggregation cost increases at this range of filter sizes). When the filter size becomes larger ($\geq$ 100), the improved filtering performance results in significant decrease on the candidate aggregation cost. When the filter size increases further, the total cost is dominated by the candidate filtering cost, explaining the latter increase of the total cost. Among all of the tested cases, the total cost reaches the smallest when g is 100. This confirms our analysis in Section 5.3.3. According to Formula 5.3, $g_{opt} = c + \frac{\bar{v}_{light}}{\delta \cdot \bar{v}}$ (c is a small positive constant). Since δ is 0.01 and $\frac{\bar{v}_{light}}{\bar{v}}$ is around 0.8, we obtain $g_{opt} = c + 80$.

5.4.2 Effect of the Number of Filters

We vary the number of filters (f) from 1 to 10 and evaluate its effect on the performance of netFilter. Figure 5.7(a) shows the number of candidates propagated per peer during candidate verification and the number of heavy item groups (the filter size is set to 100). The number of candidates propagated per peer decreases with the number of filters, which confirms the benefits of multiple filters as discussion in Section 5.2.2. The major trend observed on the number of heavy item groups is that it almost increases linearly with the number of filters, which is expected.



(a) Number of candidates and heavy item groups



(b) Communication cost

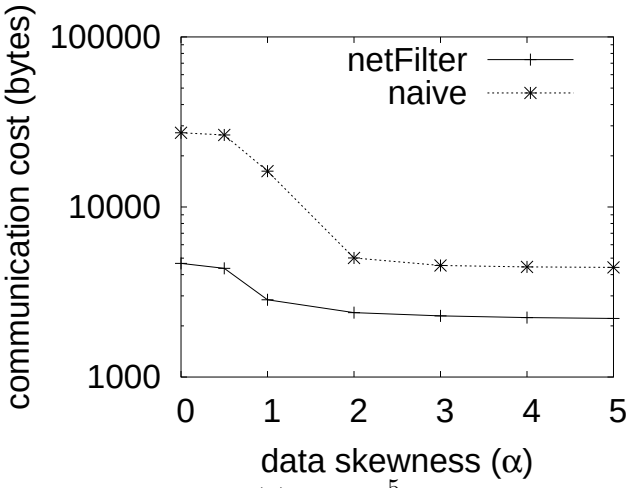
Fig. 5.7. Effect of number of filters on netFilter.

Figure 5.7(b) shows the communication cost under the same setting as for Figure 5.7(a). We also plot the candidate filtering cost, candidate dissemination cost and candidate aggregation cost in the figure. When the number of filters is small, the candidate aggregation cost is large. With the increase of the number of filters, the filtering performance improves, resulting in reduced candidate aggregation cost. On the other hand, the major trend observed on both candidate filtering cost and candidate dissemination cost is that they increase linearly as expected.

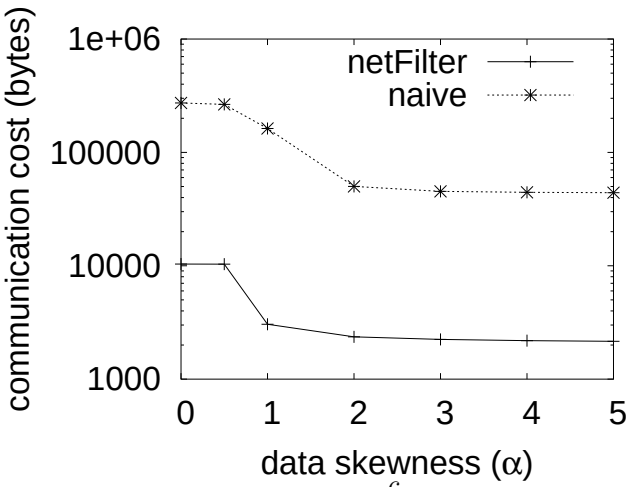
The total cost initially decreases, then increases. When the number of filters is small, the total cost is dominated by the candidate aggregation cost. The initial decrease is explained by the improved candidate aggregation cost. With the further increase on the number of filters, the total cost is dominated by the candidate filtering cost, which explains the latter increase of the total cost. Among all of the tested cases, the total cost is the smallest when f is 3. This again confirms our derivation in Section 5.3.4. According to Formula 5.6, $f_{opt} = \lceil log_{1/(1-(1-1/g)^r)}(s_a + s_i) \cdot (n-r)/(s_a \cdot g) \rceil = \lceil log_{1/(1-(1-1/100)^7)}(4+4) \cdot (10^5 - 7)/(4 \cdot 100) \rceil = 3$.

5.4.3 Effect of Data Skewness

We conduct experiments to evaluate the effect of data skewness on the performance of netFilter. Figure 5.8(a) and (b) show the results under different α values (data skewness) with N as 10^5 and 10^6, respectively. We adopt the optimal setting for netFilter obtained from similar experiments as in the above two subsections (the filter size is set to 100, and the number of filters is set to 3 and 5 under n as 10^5 and 10^6, respectively). We include the communication cost incurred by the naive approach for



(a) n = 10^5



(b) n = 10^6

Fig. 5.8. Effect of data skewness on netFilter.

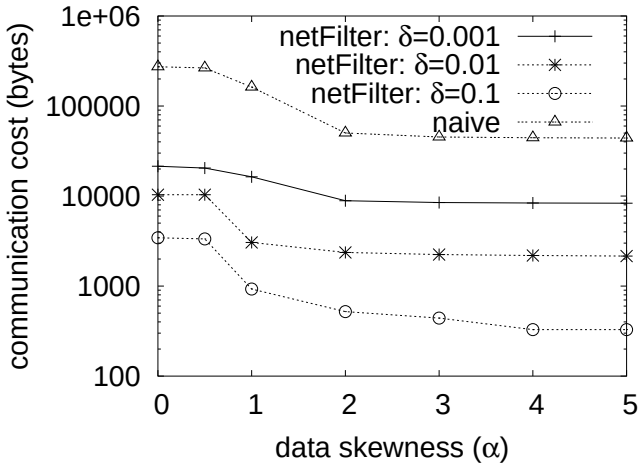


Fig. 5.9. Effect of threshold on netFilter ($n = 10^6$).

comparison. For readability, the y-axis is on log scale. From these figures, we can see that the cost incurred by netFilter is much smaller than that incurred by the naive approach. For instance, with n as 10^6, the cost incurred by netFilter is only $2\% - 5\%$ of that incurred by the naive approach. This confirms our analysis in previous section. In addition, the cost incurred by both netFilter and the naive approach decreases with the data skewness. For netFilter, this is explained by the improved filtering performance under very skewed data. For the naive approach, as the data skewness increases, the average number of distinct items that a peer propagates along the hierarchy is reduced, resulting in reduced communication cost.

5.4.4 Effect of Threshold

To evaluate the effect of the threshold, we conduct experiments under different threshold ratios (δ). Figure 5.9 shows the results with δ set to $0.1, 0.01$ and 0.001. We plot the communication cost incurred by the naive approach for comparison. Again, we adopt the optimal setting for netFilter (the size of filter and number of filters are set to (10, 6) under $\delta = 0.1$, (100, 5) under $\delta = 0.01$, and (1000, 2) under $\delta = 0.001$). From this figure, we can see when the threshold ratio increases, the communication cost decreases. This is as expected since larger threshold implies less number of items satisfies the threshold condition.

5.5 Summary

The need of identifying frequent items (IFI) appears in a variety of P2P applications. In this study, we investigate the problem of IFI in P2P systems. We first formally define the problem of IFI, and discuss how the essential operations in various P2P applications can be transformed to the problem of IFI. We propose an efficient in-network processing technique called *netFilter* to address IFI. In addition, we address various issues involved in the design of netFilter, i.e., aggregate computation, candidate set optimization, and candidate set materialization. Furthermore, we obtain the cost model for netFilter, derive the optimal setting for netFilter mathematically, and provide solutions to achieve the optimal setting in practice. Through extensive simulations, we validate the cost model and demonstrate the effectiveness of netFilter.

In the future, we plan to investigate a fault-tolerant gossip aggregation that can obtain the precise aggregates from the network. We then plan to extend the solutions proposed in this study on gossip aggregation. In addition, we plan to leverage our proposal to address various issues in P2P systems as summarized in Table 5.1, e.g., cache management, query refinement, network attack detection, and etc.

Chapter 6

Monitoring Changes on the Data Distribution

A massive amount of data is collected and stored at a large number of host nodes connected via wired networks or wireless networks. Users are often interested in monitoring some interesting patterns or abnormal events hidden in the data rather than the raw data. Transferring all the raw data observed at each host node to a central coordinator for processing is costly and unnecessary. In this study, we investigate monitoring changes on the data distribution in the networks (MCDN), which is a prevailing task for a wide range of applications. To address this problem, we propose a technique called *wavenet*. The basic idea is to compress the local item set of each host node into a compact yet accurate summary, called *localwavelet*, for communication with the coordinator. In addition, we identify various issues involved in the design of wavenet, i.e., *localwavelet construction in a sparsely populated data domain* and *localwavelet propagation*, and propose *localwavelet refinement* and *adaptive monitoring* to address these issues, respectively. The extensive performance evaluation demonstrates the efficiency of wavenet.

6.1 Introduction

A massive amount of data is collected and stored at a large number of host nodes connected via wired networks or wireless networks, such as sensor networks, mobile ad

hoc networks, Internet, and etc. Users are often interested in monitoring some "interesting" information hidden among the data, such as anomalies or patterns, rather than the raw data themselves. For instance, monitoring patterns or trends itself usually is one of the major tasks for sensor networks. In addition, monitoring trends or anomalies in the network traffic can help pinpoint the performance bottlenecks of the network, facilitate the designers and administrators make strategic decisions on the system design and resource management, ensure the network's proper operation, and even detect on-going network attacks [40, 71, 77]. For such applications, simply transferring the raw data collected at all the host nodes to a central coordinator for processing is unnecessary and also impractical due to the vast amount of data and large scale of the network.

In this study, we investigate efficient techniques for MCDN, one of the important monitoring tasks. In the following, we use three examples to motivate this task.

- Changes on the distribution of network traffic features (e.g., IP addresses, ports) observed in the flow traces can be used to detect a wide spectrum of network anomalies, including alpha flows, DOS attacks, flash crowds, and etc. ([77]). For instance, alpha flows are characterized by a concentrated distribution in source and destination addresses. DOS attacks are characterized by a concentrated distribution in destination addresses. Flash crowds are characterized by traffic originated from a dispersed set of source ports to a concentrated set of destination addresses.

- In applications that query networked data sources, statistics about the selectivity of query predicates often need to be collected from the networks to facilitate query

optimization. MCDN can be adopted in such applications to help collecting the required statistics from the networks efficiently.

- MCDN can be applied on various distributed data mining tasks (e.g., clustering, outlier detection) to significantly reduce the communication cost. A common approach to perform distributed data mining in the network system (e.g., [120]) is to first construct a global state based on the data distribution in the system and then disseminate the global state to the host nodes, which then perform local computations based on the obtained global state. It would be too costly to constantly evaluate/re-construct the global state by pulling the raw data from all the host nodes. A better solution is to monitor changes on the data distribution in the networks, which then trigger the re-evaluation and/or re-construction of the global state to ensure the validity of the global state.

6.1.1 Problem Formulation

Assume in the networks, N remote host nodes are collecting data. A designated coordinator is responsible for detecting and reporting changes on the distribution of the data, which are the union of the data collected at all the host nodes. The host nodes communicate with the coordinator, and themselves do not communicate with each other (aggregation techniques as proposed in [94, 95] could be adopted if the host nodes organize into a multi-level hierarchical structure. In this study, we do not apply aggregation and assume the host nodes do not communicate with each other to highlight the contributions made by our proposal). Figure 6.1 illustrates the system model. This model has been adopted in most prior works on distributed monitoring problem [15,

30, 31, 37, 69, 104]. As a matter of fact, this model is the representative model used in a large class of applications, including network monitoring where network elements (e.g., routers) distributed in the networks collect network traffic statistics, such as link bandwidth utilization or the volume of traffic exchanged among host nodes, and report to a central Network Operation Center (NOC) for processing.

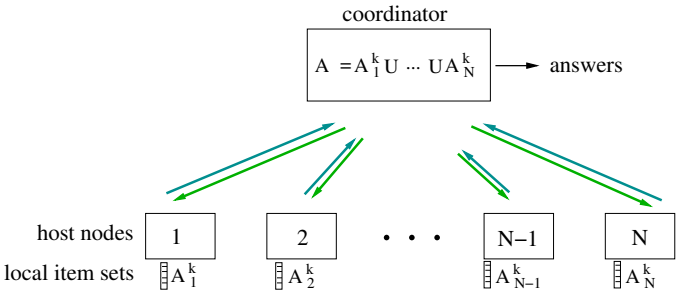


Fig. 6.1. System Model of MCDN.

We use t to denote the number of distinct items in the system. The item set of interest is represented by A, which consists of t items, i.e., Item 1, Item 2, ..., Item t. Therefore, the domain of the item set A is $\{1, 2, ..., t\}$. Each host node $j \in \{1, 2, ..., N\}$ has a local item set at time k, denoted by A_j^k, which is a subset of A. We use $v_{i,j}^k$ to denote the non-negative *local value* associated with Item i in host node j at time k. An item's *global value* at time k, denoted by v_i^k, is the sum of the local values for Item i in all the host nodes at time k, i.e., $v_i^k = \sum_{j=1}^{N} v_{i,j}^k$. v^k denotes the sum of the global values of all the items in the system at time k, i.e., $v^k = \sum_{i=1}^{t} v_i^k$.

As an example, in the case of network monitoring where the routers collect the volume of traffic exchanged between source-destination IP address pairs [77], the interpretation of the aforementioned terms is as follows. The item set A is the set of source-destination IP address pairs, and the domain of A is $\{1, 2, ..., 2^{64}\}$. The local value $v_{i,j}^k$ is the size of the packets exchanged between the source-destination IP address pair i that are observed by host node (router) j at time k. The global value v_i^k is the size of the packets exchanged between the source-destination IP address pair i that are observed by all the host nodes (routers).

To measure the distribution of the global values among items in the system, we could adopt different metrics, such as entropy, cumulative sum, or even histogram. Note that regardless of the metric used, we expect the research issues to be discussed in the following are inherent to the problem of MCDN and the proposed techniques are not affected significantly by the adopted metric. In this study, we use (*normalized*) entropy (as exemplified in [77], entropy is very useful to measure traffic feature distribution to reveal network anomalies). Here, we normalize entropy by $log(t)$ so that the value of normalized entropy is in the range of $[-1, 0]$. In the following, we refer normalized entropy as entropy if the context is clear.

Now we proceed to the formal definition of MCDN. MCDN is to detect and report *distribution changes with respect to* ϵ, defined as the changes on the normalized entropy that exceed a predefined threshold ϵ. We denote the normalized entropy of an item set at time k as E_k, i.e.,

$$E_k = \frac{-\sum_{i=1}^{t} \frac{v_i^k}{v^k} log \frac{v_i^k}{v^k}}{log(t)} \tag{6.1}$$

Suppose the most recently reported entropy is $E_{k'}$. MCDN is to report E_k with $|E_k - E_{k'}| \geq \epsilon$ where $k > k'$. Formally,

$$MCDN = \{(k, E_k)||E_k - E_{k'}| \geq \epsilon\} \tag{6.2}$$

MCDN implicitly requires the global knowledge of all the distinct items in the system. It is a nontrivial task to solve MCDN efficiently given the following characteristics of the data and the system to be monitored: 1) Decentralization: different host nodes, connected via wired or wireless networks, are collecting data; 2) Large scale: the number of host nodes is normally very large; 3) Continuality: data are constantly generated at the host nodes; 4) Real time: the changes on the data distribution should be detected early enough to be useful.

6.1.2 Contributions

An intuitive solution to address MCDN is to use histogram to summarize the local item sets and propagate the histograms to the coordinator. As discussed later, this simple solution is not adequate to address MCDN. In this study, we propose a distributed monitoring framework, called *wavenet*, to monitor changes on the data distribution in the networks efficiently. Wavenet, drawing inspirations from the wavelet technique, compresses the local item set at a host node into a compact and accurate summary, called *localwavelet*, for communication with the coordinator. We identify two research issues involved in the design of wavenet, i.e., *localwavelet construction in a sparsely populated data domain* and *localwavenet propagation*. To address localwavelet construction in a

sparsely populated data domain, we propose *localwavelet refinement*, which transforms the items in the original data domain into a compact *virtual valid item set* that carries the values for the *valid items* (items with non-zero global values) in the system based on probabilistic counting [45], and then constructs *refined localwavelets* on the virtual valid item set. To address localwavelet propagation, we propose *adaptive monitoring* that propagates localwavelets to the coordinator only when necessary, i.e., when changes on the local values of items are significant enough to potentially result in distribution changes with respect to ϵ.

According to the authors' best knowledge, this is the first study addressing the problem of MCDN. We summarize the primary contributions of this paper as follows:

1. We identify and formally define a fundamental problem, i.e., monitoring changes on the data distribution in the networks (MCDN), which is encountered in a wide range of applications, e.g., network anomalies detection, query optimization on networked data, and distributed data mining.

2. We propose a distributed monitoring framework called wavenet that efficiently monitors changes on the data distribution in the networks.

3. We propose a suite of techniques to address various issues involved in the design of wavenet, i.e., localwavelet construction in a sparsely populated data domain and localwavenet propagation.

4. We perform extensive evaluation to demonstrate the efficiency of wavenet.

A couple of recent studies also leverage the technique of wavelet, e.g., [72, 85, 98]. However, the problems addressed in those studies are different from the problem that

we address here, and thus the research issues involved and the proposed techniques are completely different. In [72], wavelet is applied to remove noises from a signal so that congested links can be detected. In our previous study [85], we leverage wavelet to facilitate load balancing in P2P systems. In [98], wavelet technique is applied to estimate query selectivity.

The rest of this chapter is organized as follows. We provide some preliminary background knowledge in next section. We then present the design details of wavenet in Section 6.3. The performance evaluation is provided in Section 6.4. Lastly, we draw the conclusion and outline the further direction in Section 6.5.

6.2 Preliminaries

In this section, we present the background knowledge on the probabilistic counting technique (the background on wavelet is provided in Section 4.2.1.3.

Flajolet and Martin propose a very simple yet efficient probabilistic counting technique in [45] to efficiently estimate the cardinalities of large data sets. The basic idea is to capture some statistics of the item set in a compact data structure called *FM-sketch*, which is then used to estimate the number of distinct items in the item set. FM-sketch is a k-bits long bit string where 2^k is the upper bound on the number of distinct items in the system. k can be set to be a large enough number (e.g., 64 or 128). Each item is first randomly hashed to a number in $\{0, 1, 2, ..., k\}$ where the probability that an item is hashed to i is $\frac{1}{2^{i+1}}$. Denoting the hashed value of Item x as $h(x)$, the $h(x)^{th}$ bit in FM-sketch is then set to 1 if it has not been set yet. Given b as the position of the least significant unset bit in the FM-sketch after we perform the above operations

for all the items, 2^b is a good estimation of the number of distinct items in the item set. Figure 6.2 shows one example of FM-sketch generated from an item set. The rightmost bit represents the least significant bit. The shaded rectangles indicate set bits and blank rectangles indicate unset bits. The least significant unset bit is bit 9. Thus, the number of distinct items in this item set is estimated as $2^9 = 512$.

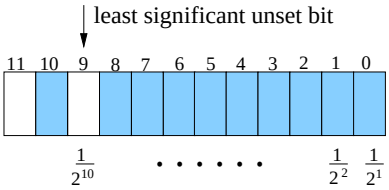


Fig. 6.2. An illustrative example of FM-sketch.

6.3 Wavenet

The naive solution for MCDN is to ask the host nodes to propagate their local item sets to the coordinator. This incurs excessive communication cost. In this study, we propose wavenet, which approximates the local item set of a host node in a very compact yet accurate way by leveraging the wavelet technique. Wavenet reduces the communication cost significantly compared to the naive approach yet without compromising the detection accuracy too much. Wavenet conceptually consists of the following three steps, *data summarization*, *value recovery* and *change detection*. To perform data

summarization, each host node compresses its local item set into a compact yet accurate summary, i.e., localwavelet, which is forwarded to the coordinator. To perform value recovery, the coordinator combines the received localwavelets by pairwise addition and reconstructs the (approximate) global values from the combined localwavelet. Finally, in change detection, the coordinator calculates the entropy based on the obtained global values and checks whether there is a distribution change with respect to ϵ. If that is the case, the change is reported.

In the following, we first present the center idea of wavenet, i.e., using wavelet as the data summarization technique to compress the local item set of a host node into localwavelet. We then discuss the research issues involved in the design of wavenet and present our solutions to address these issues.

6.3.1 Data Summarization

In the following, we first discuss the pitfalls of applying histogram, the well-known data summarization technique, to address MCDN. Following that, we present the idea of localwavelet.

6.3.1.1 The Weakness of Histogram

The basic idea of histogram is to partition the data domain into *buckets* with each one holding multiple items. The *bucket size* is the number of items held in each bucket. Here we focus on histogram with equal sized buckets. Some studies (e.g., [58]) in the literatures have proposed techniques to construct histogram with variably sized

buckets. These techniques require complicated computation to decide the optimal or sub-optimal boundaries for buckets, which might not be feasible to be implemented at certain network elements, such as routers. In addition, to transfer these types of histogram, we need to propagate the boundary together with the value of each bucket, which incurs extra communication overheads. Therefore, here we assume the buckets have the same size.

We can adopt the histogram technique to compress the local item sets into *histogram summaries* as follows. Suppose we have x buckets in the histogram (i.e., the bucket size is $\frac{t}{x}$). We define the sum of the global values of all the items held in a bucket as the *bucket aggregate*. For each bucket b_m ($m \in \{1, ..., x\}$), we can first obtain its bucket aggregate at time k, denoted by v_{bm}^k, as $\sum_{i \in b_m} v_i^k$. Then we use $v_{bm}^k / bucket\_size$ ($= v_{bm}^k \cdot x/t$), i.e., the average value obtained from the bucket aggregate of a bucket, as the estimated global value of each item in this bucket.

Figure 6.3 illustrates one example. Assume that we have eight items distributed on two host nodes (host node 1 and host node 2). Suppose the number of buckets are four (the bucket size is 2). Each host node obtains a histogram summary for its local item set as displayed in the figure. These histogram summaries are then propagated to the coordinator, which combines these histogram summaries and then estimates the global values for the items based on the merged histogram summary as $\{1.5, 1.5, 4.5, 4.5, 121, 121, 1, 1\}$. The exact global values for these 8 items are $\{2, 1, 3, 6, 240, 2, 1, 1\}$.

The histogram approach as described above is very simple. However, histogram is suitable for compressing relatively uniformly distributed item set but not skewly distributed item set since it smoothes out the skewness existing among items. Therefore,

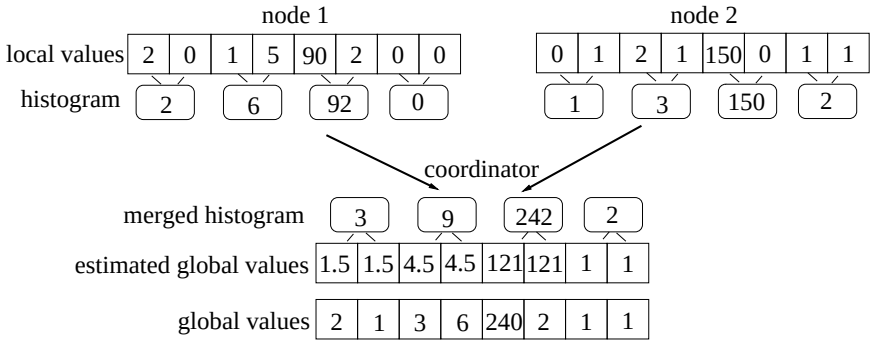


Fig. 6.3. An illustrative example of adopting histogram as the data summarization technique.

we expect that this approach can not detect significant changes on the data distribution, i.e., changes from uniformly distributed item set to extremely skewly distributed item set or vice versa. For instance, suppose initially we have a uniformly distributed item set with 8 items. The entropy is -1 at this point. Later at time k, the values of the items change to the case as displayed in Figure 6.3. According to Equation 1, the exact value of the entropy for this item set is -0.164. Suppose the threshold ϵ is 0.8. This change (from the uniformly distributed item set to the item set displayed in Figure 6.3) is a distribution change with respect to 0.8 since $|-0.164-(-1)| = 0.836 > 0.8$. However, if we adopt the above histogram approach to summarize the local item sets, this change will be missed since the entropy estimated from the histogram summary is -0.445 and $|-0.445-(-1)| = 0.555 < 0.8$.

6.3.1.2 Localwavelet

In contrast to histogram, we observe that wavelet can be applied to compress both skewly distributed item set and uniformly distributed item set. To approximate uniformly distributed item set, we can simply use the average wavelet coefficient. To approximate skewly distributed item set, we can use a subset of the most significant wavelet coefficients, which capture the skewness among items. Figure 6.4 illustrates an example of how wavelet can compress skewly distributed item set. Note that the values of the eight items in the figure are not uniformly distributed, and Item 5's value is significant larger than other items' values. The four significantly larger wavelet coefficients as marked by the shaded rectangles can be used to accurately approximate this skewly distributed item set.

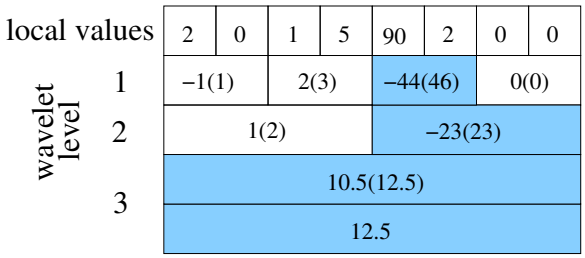


Fig. 6.4. An illustrative example of representing skewly distributed item set in the wavelet transform.

In the following, we explain how to adopt wavelet transform to address MCDN. Here we first assume the data domain is densely populated by items (we will explain

how to address MCDN when the data domain is sparsely populated by items later). To address MCDN, each host node generates a wavelet transform capturing the local values of the items in its local item set. It then chooses the most significant wavelet coefficients to approximately summarize its local item set. The set of the chosen wavelet coefficients forms the localwavelet of this host node. More specifically, the localwavelet of a host node consists of pairs $\langle position, value \rangle$ for each chosen wavelet coefficient (the position here is the position of a wavelet coefficient in the wavelet transform). Host nodes then propagate their localwavelets to the coordinator, which combines the received localwavelets by pairwise addition.

One issue here is to properly set the size of the localwavelet, also called as *summary size*. To address this issue, we first introduce a metric, called *summary error (e)*, to measure the error introduced by a data summarization technique. e is defined as: $e = |\frac{\tilde{E}-E}{\tilde{E}+E}|$ where E and $\tilde{E}$ are the exact entropy and the entropy estimated from the localwavelet, respectively. When $\tilde{E}$ equals to E, the summary error is 0. When $\tilde{E}$ deviates from E significantly (i.e., $\tilde{E} >> E$ or $\tilde{E} << E$), the summary error is close to 1. We use $\tilde{E} + E$ as the divisor so that the maximum summary error is bounded by 1.

Each host node performs local computation to set the summary size as the smallest size that introduces a summary error below a predefined error threshold as follows. It starts by setting the summary size to 1 (i.e., the most significant wavelet coefficient is chosen) and forming a localwavelet. Then it computes the summary error introduced by this localwavelet. If the summary error is larger than the predefined error threshold, it increases the summary size to 2, repeats the above procedure and so on until the summary error falls below the predefined error threshold.

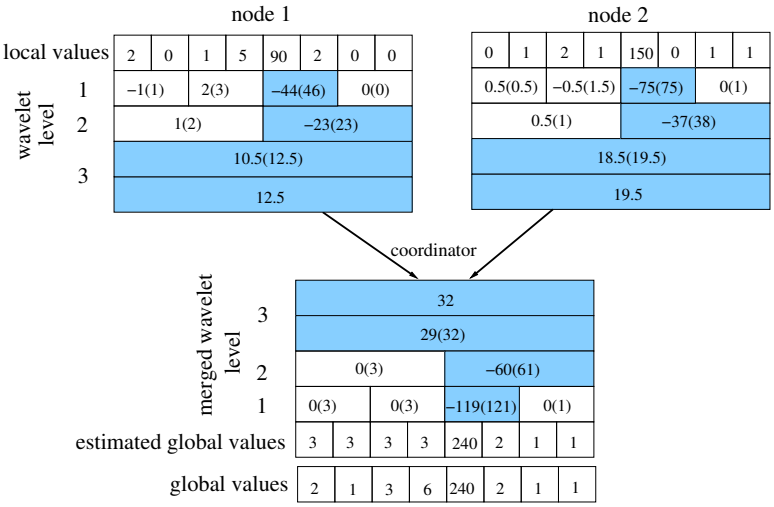


Fig. 6.5. An illustrative example of localwavelet.

Figure 6.5 illustrates the concept of localwavelet. We use the same item sets that are used in the example of histogram (Figure 6.3). Suppose both of the host nodes set the summary size as 4. In the figure, we illustrate the wavelet transform obtained at both of the host nodes. For the clarity of presentation, we omit the normalization process in the figure. Host node 1 chooses the four most significant wavelet coefficients (as marked by the shaded rectangles in the figure) to form its localwavelet, which is $\{(12.5, 1),\ (10.5, 2),\ (-23, 4),\ (-44, 7)\}$. Similarly, host node 2 chooses the four most significant coefficients (also marked by the shaded rectangles in Figure 6.5) to form its localwavelet, which is $\{(19.5, 1), (18.5, 2), (-37, 4), (-75, 7)\}$. These two host nodes then propagate their localwavelets to the coordinator, which merges the received localwavelets by pairwise addition. Thus, we obtain the merged localwavelet as (32,1), (29,2), (-60,4), and (-119,7). The coordinator then obtains the wavelet for the items in the system as

$\{32, 29, 0, -60, 0, 0, -119, 0\}$. Based on this wavelet, it reconstructs the global values for all the items by following the reverse steps of wavelet construction. The global values estimated from this wavelet are $\{3, 3, 3, 3, 240, 2, 1, 1\}$, which are very close to the exact global values (also displayed in the figure). As a matter of fact, the entropy estimated from localwavelet in this example is -0.168, which is very close to the exact entropy -0.164.

6.3.2 Design Issues of Wavenet

A couple of critical research issues need to be addressed in the design of wavenet:

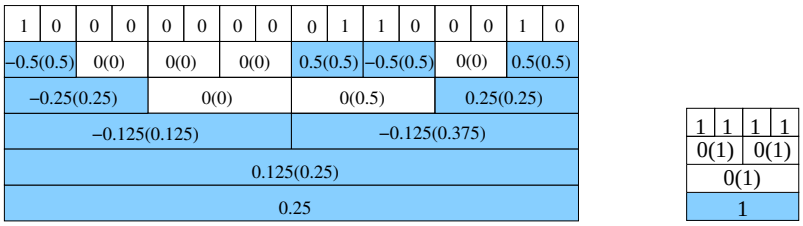
1. **Localwavelet construction in a sparsely populated data domain.** For some applications, the data domain is sparsely populated by items. For instance, in the network monitoring example, only a small portion of the 2^{64} source-destination IP address pairs has valid connections at a particular instance of time, i.e., the global values of majority of the items are 0. The wavelet obtained on a sparsely populated data domain has a lot of non-zero wavelet coefficients. For instance, given an item set with values as {1, 0 ,0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1} where only 25% of the 16 items have nonzero values, the wavelet transform for this item set is {0.25, 0.125, −0.125, −0.125, −0.25, 0, 0, 0.25, −0.5, 0, 0, 0, 0.5, −0.5, 0, 0.5} as depicted in Figure 6.6(a). The number of non-zero wavelet coefficients as marked in the figure, 10, is much greater than the number of nonzero item (4) in the original item set. Thus, if we simply summarize this item set in the sparsely populated data domain, we will either compromise the detection accuracy too much or we will end up with a very large data summary, which can not save communication cost too much.

2. **Localwavelet propagation.** An intuitive approach for localwavelet propagation is to propagate the localwavelets periodically. However, it is hard if not infeasible to set an optimal interval between propagations. This approach either will incur unnecessary communication cost when we set the interval too short and the values of items do not change much within an interval, or will not be able to detect distribution changes early enough if we set the interval too long.

To address the above two issues, we propose localwavelet refinement and adaptive monitoring, respectively. Localwavelet refinement constructs a compact virtual valid item set carrying the values of valid items (with non-zero global values) and then constructs a refined localwavelet on the virtual valid item set. Adaptive monitoring filters the changes on the local values of items at a host node that won't cause significant change on the data distribution and only invokes localwavelet propagation when the changes are significant. Note that the aforementioned two issues are faced by other data summarization techniques (e.g., histogram) as well. The solutions to be discussed shortly are applicable for other data summarization techniques with minor modifications.

6.3.3 Localwavelet Construction in a Sparsely Populated Data Domain

As mentioned above, for some applications, the data domain is sparsely populated by items. Thus, if we simply summarize this item set in the sparsely populated data domain within sufficient accuracy, the size of the localwavelet has to be large enough. However, if we can first remove the items with zero values from the item set and obtain a *valid item set* consisting of only valid items, and then construct the localwavelet on the valid item set, we will be able to obtain a much more compact localwavelet. For instance,



(a) original data domain (b) valid data domain

Fig. 6.6. An illustrative example of localwavelet construction on a sparsely populated data domain.

in Figure 6.6(b), we first remove all of the zero entries from the item set in the example shown in Figure 6.6(a), and obtain a valid item set consisting of the four valid items (items $1, 10, 11, 15\}$. The wavelet transform of this valid item set is $\{1, 0, 0, 0\}$ (as shown in Figure 6.6(b)). Only 1 out of the 4 wavelet coefficients is non-zero. Thus, we can represent this items set by only one wavelet coefficient instead of 10 wavelet coefficients as in the wavelet transform constructed on the original sparsely populated data domain.

The question is how we can obtain the valid item set so that we can construct a compact localwavelet in the valid item set. A simple solution is to ask each host node to propagate the identifiers of each item in its local item set to the coordinator. The coordinator can then obtain the valid item set by merging the items obtained from all the host nodes. Although simple, this approach is very costly. Using this approach, the communication cost incurred by obtaining the valid item set alone is half of the cost incurred to address MCDN using the naive approach as mentioned in the beginning of Section 6.3. Here we have the following observation, which motivates us to address

the above issue from a different angle without incurring the aforementioned excessive overhead.

Observation: As long as the coordinator knows the set of global values in the system, it can obtain the entropy. It does not need to know specifically which items are valid and which valid item has what global value.

The reason is as follows. Suppose the set of global values for an item set A is $\{v_1, v_2, ..., v_t\}$ (v_i is the global value of Item i). Any random permutation on the set of global values will not change the entropy value of this item set.

Based on this observation, we propose an technique, called localwavelet refinement. The basic idea is to transform the items in the original data domain into a virtual valid item set that carries the values of the valid items but not the identifier of the valid items, and then construct refined localwavelet on the virtual valid item set. In the following, we first explain the concept of virtual valid item set, and then explain how to perform localwavelet refinement.

We denote the number of valid items in the system as C. The *virtual valid data domain* is then $\{1, 2, ..., C\}$. Each valid item in the original data domain is randomly mapped to an item in the virtual valid item set (with the domain as the virtual valid data domain), i.e., $f(x) : \{1, 2, ..., n\} \rightarrow \{1, 2, ..., C\}$. We call $f(x)$, the mapped value of item x, as the *virtual identifier* of this item, which serves as the identifier for this item in the virtual valid item set. Note that all the host nodes apply the same mapping function so that the same item at different host nodes is mapped to the same virtual identifier in the virtual valid data domain.

Localwavelet refinement consists of three steps: 1) *valid item monitoring*: host nodes monitor the number of valid items in the system; 2) *valid data summarization*: host nodes construct refined localwavelets and forward them to the coordinator; 3) *value recovery for valid items*: the coordinator obtains the global values for the valid items from the combined localwavelet. In the following, we first explain Steps 2 and 3, after which it will become clear why we only need to monitor the number of valid items in the system (C). Following that, we then discuss the details of Step 1.

Valid Data Summarization. For now we assume that all the host nodes know the value of C (the number of valid items in the whole system). Each host node obtains its *local virtual valid item set* as follows. It maps the local value of each (valid) item in its local item set to the value of the corresponding item in the virtual valid item set, and sets the values of any other remaining items in the virtual valid item set as 0. A host node then constructs a localwavelet on the virtual valid item set and forwards it to the coordinator.

Figure 6.7 shows one example. The original data domain is $\{1, 2, ..., 16\}$ and the valid item set consists of items 1, 10, 11 and 15. Assume the mapping from the valid items to the virtual identifier is $\{1 \rightarrow 1, 10 \rightarrow 2, 11 \rightarrow 3, 15 \rightarrow 4\}$. That is the first item in the original data domain is mapped to the first item in the virtual valid data domain; the 10^{th} item in the original data domain is mapped to the second item in the virtual valid data domain; and so on. The local values on the items in the virtual valid item sets at host nodes 1 and 2 are $\{1, 1, 1, 1\}$ and $\{2, 0, 0, 2\}$, respectively. The host nodes then construct wavelet (as shown in the figure) and choose a certain number of the most

significant wavelet coefficients as explained earlier in Section 6.3.1.2 to form their refined localwavelets, respectively.

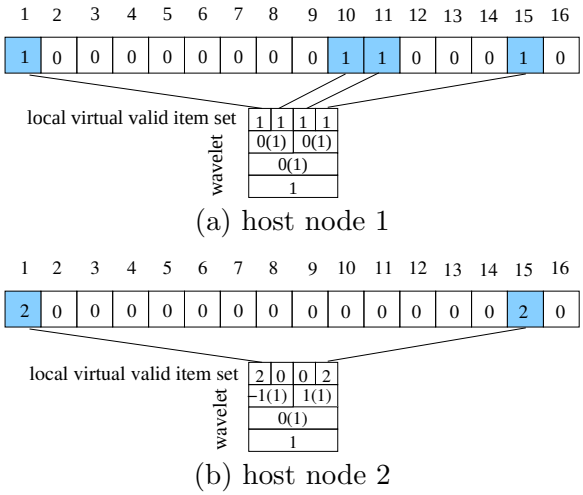


Fig. 6.7. An illustrative example of valid data summarization.

Value Recovery for Valid Items. After the coordinator receives the refined localwavelets from all the host nodes, it obtains the (approximate) global values for the valid items from the merged localwavelet and sets the global values for other items as 0. Based on Observation 1, the coordinator can calculate the entropy for the system without knowing which specific item has what global value.

Valid Item Monitoring. At this point, it is clear that as long as the host nodes know the total number of valid items in the system (C), Step 2 and 3 can be conducted correctly. Now the question becomes how we can obtain the number of valid items

in the system. One possible solution is to ask host nodes to propagate the numbers of valid items in their local item sets to the coordinator, which adds these numbers together. Unfortunately, this simple solution does not work since there might be overlapping among the local item sets of different host nodes. Simply adding together the numbers of valid items from different host nodes will count the same item appearing at multiple host nodes multiple times.

In this study, we adopt FM-sketch as explained in Section 6.2 to obtain an approximate estimation for C (denoted as $\hat{C}$) as follows. As we demonstrate later, it is sufficient to obtain an approximate number of valid items.

Each host node first constructs a FM-sketch locally based on its local item set. All host nodes apply the same hashing functions when they construct the FM-sketches locally so that the same item appearing at different host nodes is hashed to the same bit in the FM-sketches of different host nodes. The host nodes then propagate the local FM-sketches to the coordinator, which simply combines these FM-sketches by bitwise-oring them and obtains $\hat{C}$ from the combined FM-sketch (we also propose techniques to adaptively estimate C, similar to the principle of adaptive monitoring to be discussed shortly.).

6.3.4 Adaptive Monitoring

As discussed earlier, periodically propagating localwavelets from all the host nodes to the coordinator either incurs unnecessary communication overhead or is not sufficient to detect distribution changes promptly. In this study, we propose adaptive monitoring that invokes propagation only when necessary, i.e., when the changes on the local values

of items are significant enough to potentially result in distribution changes with respect to ϵ.

To perform adaptive monitoring, we install a *local filter* for each item at a host node. The settings of local filters at host nodes are also known to the coordinator. If the changes on the local values of all the items at a host node satisfy the constraints associated with their local filters, there is no need for this host node to propagate the localwavelet (or histogram summary) to the coordinator. Otherwise, we say the local filter for an item at this host node is violated, called as *filter violation*. Upon filter violation, *filter resolution* is invoked to resolve the violation. In addition, the relevant local filters are also reset when necessary. The general idea of adaptive monitoring is intuitive and straightforward. Other studies adopted similar ideas as well (e.g., [15, 69]). However, different from these previous studies, the issues of setting up the local filters at the host nodes and resolving the filters upon violation are more complicated and require nontrivial solutions, which are explained shortly. For easy reference, we list the symbols used in the rest of the paper in Table 6.1.

6.3.4.1 Local Filter Setup

We observe that as long as we know $|E_k - E_{k'}| < \epsilon$, there is no need to propagate the localwavelets to the coordinator. By plugging Equation 6.1 into the above equation, we obtain

$$|-\frac{\sum_{i=1}^{t}\frac{v_i^k}{v^k}log\frac{v_i^k}{v^k}}{logt} - E_{k'}| < \epsilon \Longrightarrow$$

Table 6.1. Symbols used in wavenet

Symbols: Descriptions
N: Number of host nodes in the network
t: Number of distinct items in the system
ϵ: Threshold value
$v_{i,j}^k$: Local value of Item i in host node j at time k
v_i^k: Global value of Item i at time k
v_j^k: Sum of the local values of all the items at host node j
v^k: Sum of the global values of all the items at time k

$$logt \cdot (E_{k'} - \epsilon) < -\sum_{i=1}^{t} \frac{v_i^k}{v^k} log \frac{v_i^k}{v^k} < logt \cdot (E_{k'} + \epsilon).$$

By associating with the local filter of Item i a positive *weight coefficient* (denoted by w_i, $\sum_{i=1}^{t} w_i = 1$), we can obtain the setting on the local filter for Item i as

$$-w_i \cdot logt \cdot (E_{k'} + \epsilon) < \frac{v_i^k}{v^k} log \frac{v_i^k}{v^k} < -w_i \cdot logt \cdot (E_{k'} - \epsilon). \tag{6.3}$$

In our design, we set $w_i = v_i^{k'}/v^{k'}$, i.e., the weight coefficient for the local filter of Item i is proportional to its global value at time k'. The intuition behind this setting is that the items with larger global values may have more significant changes on their global values, and associating larger weight coefficients with the local filters of such items allows more room for changes on these items.

In the above equation, we have the term v_i^k (the global value for Item i at time k) and v^k (the sum of the global values of all the items), which are not available. Nevertheless, we observe that $v_i^{k'}$ and $v^{k'}$ are available (since at time k', all the host nodes

propagate their localwavelets to the coordinator, which combines these localwavelets and reconstructs the global values for all the items). Thus, we can remove the terms v_i^k and v^k from the above equation by expressing them in terms of $v_i^{k'}$ and $v^{k'}$, respectively.

We first explain how to express v^k in term of $v^{k'}$. We constrain the maximum change allowed on the sum of local values at each host node (denoted as v_j^k) as δ, i.e., $v_j^{k'}/\delta \leq v_j^k \leq \delta \cdot v_j^{k'}$. Therefore, the maximum change on v^k ($= \sum_{j=1}^{N} v_j^k$) is constrained to δ as well, i.e., $v^{k'}/\delta \leq v^k \leq \delta \cdot v^{k'}$. If the change on v_j^k (the sum of local values at a host node j) exceeds δ, a message is sent to the coordinator, which solicits v_j^k from all the host nodes to obtain a new $v^{k'}$. In the design, we set δ to 2 to avoid frequent solicitation of v_j^k from the host nodes.

Replacing v_i^k/v^k with y in Equation 6.3, we can simplify $\frac{v_i^k}{v^k} log \frac{v_i^k}{v^k}$ to $ylog(y)$. Function $ylog(y)$ is monotonically increasing when $y > 2^{-1/ln2}$ (≈ 0.36378) and monotonically decreasing when $y \leq 2^{-1/ln2}$. In the following, for presentation brevity, we focus on the case when $y \leq 2^{-1/ln2}$. If $v^{k'}/\delta \leq v^k \leq \delta \cdot v^{k'}$, we can obtain

$$\frac{v_i^k}{v^{k'}/\delta} log \frac{v_i^k}{v^{k'}/\delta} \leq \frac{v_i^k}{v^k} log \frac{v_i^k}{v^k} \leq \frac{v_i^k}{\delta \cdot v^{k'}} log \frac{v_i^k}{\delta \cdot v^{k'}}.$$

Therefore, Equation 6.3 is satisfied if the following is satisfied:

$$\begin{cases} \frac{v_i^k}{\delta \cdot v^{k'}} log \frac{v_i^k}{\delta \cdot v^{k'}} < -w_i \cdot logt \cdot (E_{k'} - \epsilon) \\ \frac{v_i^k}{v^{k'}/\delta} log \frac{v_i^k}{v^{k'}/\delta} > -w_i \cdot logt \cdot (E_{k'} + \epsilon). \end{cases} \tag{6.4}$$

We now explain how to express v_i^k in terms of $v_i^{k'}$. To do so, we associate a slack coefficient ξ_i ($\xi_i \geq 1$) with each item (i). We will explain how to obtain ξ_i shortly. ξ_i indicates the maximum change allowed on the local value of Item i without violating the setting of the local filter as indicated in Equation 6.4. When the change on the local value for Item i is constrained within ξ_i ($v_{i,j}^{k'}/\xi_i \leq v_{i,j}^k \leq \xi_i \cdot v_{i,j}^{k'}$), the change on the global value for Item i is constrained within ξ_i as well, i.e., $v_i^{k'}/\xi_i \leq v_i^k \leq \xi_i \cdot v_i^{k'}$. Therefore, we can obtain

$$\begin{cases} \frac{v_i^k}{\delta \cdot v^{k'}} log \frac{v_i^k}{\delta \cdot v^{k'}} \leq \frac{v_i^{k'}}{\xi_i \cdot \delta \cdot v^{k'}} log \frac{v_i^{k'}}{\xi_i \cdot \delta \cdot v^{k'}} \\ \frac{v_i^k}{v^{k'}/\delta} log \frac{v_i^k}{v^{k'}/\delta} \geq \frac{v_i^{k'}}{v^{k'}/(\xi_i \cdot \delta)} log \frac{v_i^{k'}}{v^{k'}/(\xi_i \cdot \delta)}. \end{cases} \tag{6.5}$$

By plugging the above equation into Equation 6.4, we can satisfy Equation 6.4 if the following is satisfied:

$$\begin{cases} \frac{v_i^{k'}}{\xi_i \cdot \delta \cdot v^{k'}} log \frac{v_i^{k'}}{\xi_i \cdot \delta \cdot v^{k'}} < -w_i \cdot logt \cdot (E_{k'} - \epsilon) \\ \frac{v_i^{k'}}{v^{k'}/(\xi_i \cdot \delta)} log \frac{v_i^{k'}}{v^{k'}/(\xi_i \cdot \delta)} > -w_i \cdot logt \cdot (E_{k'} + \epsilon), \end{cases} \tag{6.6}$$

which is the setting of local filter for Item i at the host nodes.

We can check whether a local value for an item satisfies the above local filter as follows. We can first calculate the change on the local value for Item i in host node j at time k (denoted by $c_{i,j}^k$). We then replace ξ_i with $c_{i,j}^k$ in the above equation and check whether the above equation is satisfied. If Equation 6.6 is satisfied, there is no need

for communication with the coordinator. Otherwise, the filter resolution as described shortly is invoked.

6.3.4.2 Filter Resolution

Upon a violation on a local filter, the host node with item(s) violating the local filter, called *violating item(s)*, first reports the violating item(s) to the coordinator, which then invokes filter resolution. We propose two different filter resolution techniques, i.e., *system resolution* and *item resolution*. Depending on the costs estimated for system resolution and item resolution, the one with smaller estimated cost is invoked to resolve the filter violation. In the following, we first explain the details of system resolution and item resolution. We then discuss how to estimate the costs of system resolution and item resolution.

System Resolution. To perform system resolution, all the host nodes propagate their current localwavelets to the coordinator, which then merges the received localwavelets by pairwise addition and reconstructs the (approximate) global values for all the items. It then calculates the entropy and determines whether there is a distribution changes with respect to ϵ. If that is the case, this change is reported. In addition, the local filters are reset properly if necessary.

Item Resolution. Item resolution obtains the global values for all the violating items and determines whether the global values for these items still satisfy Equation 6.4. If the global values of these items do satisfy this equation, the filter resolution terminates. Otherwise, system resolution is invoked to further determine whether there is a distribution change with respect to ϵ. The benefit of conducting item resolution

is that we can avoid localwavelet propagation if item resolution (which may have lower cost) can resolve the filter violation.

We now proceed to estimate the costs for item resolution and system resolution. The cost of item resolution, denoted as T_i, is the cost incurred by obtaining the global values of the violating items. We can simply estimate T_i as

$$T_i = N \cdot y,$$

given y as the number of violating items reported to the coordinator.

The cost of system resolution, denoted by T_s, is the cost incurred by propagating the localwavelets from all the host nodes to the coordinator. Estimating T_s is nontrivial. Here we adopt an aging technique to estimate T_s as a function of the previous costs incurred by localwavelet propagation. Assume that we have made q system resolutions up to now and we maintain the cost incurred by the most recent localwavelet propagation (denoted as $T_{s,q}$) and the average cost incurred by localwavelet propagation for the past q propagations (denoted as $\overline{T_s}$). We then estimate the cost of the current localwavelet propagation, denoted by $T_{s,q+1}$, as

$$T_{s,q+1} = \beta \cdot T_{s,q} + (1 - \beta) \cdot \overline{T_s}.$$

β determines the weights associated with the most recent cost and the average cost incurred by localwavelet propagation. Large β indicates that the estimation for $T_{s,q+1}$ is affected more by the most recent value ($T_{s,q}$) and smaller β indicates that the estimation

is affected more by the average value ($\overline{T_s}$). Putting all together, algorithm 9 summarizes the basic operations in filter resolution.

Algorithm 9 Algorithm for filter resolution.

```
1: T_i = N · y.
2: T_{s,q+1} = β · T_{s,q} + (1 − β) · T̄_s.
3: if T_i < T_{s,q+1} then
4:    Invoke item resolution.
5:    if Filter violation is not resolved then
6:       Invoke system resolution.
7:       Update local filters.
8:    end if
9: else
10:   Invoke system resolution.
11:   Update local filters.
12: end if
```

6.4 Performance Evaluation

We evaluate wavenet's efficiency using extensive simulations. For comparison, we also implement the histogram approach and evaluate its performance. In the following, we first explain the experiment setup and performance metrics. We then present the details of the results.

6.4.1 Experiments Setup

The simulation parameters and their default values (unless otherwise stated) are given in Table 6.2. In the following, we explain the data sets and the setting of local-wavelet and histogram in details.

Table 6.2. Parameters used in the simulations of wavenet

Symbols: Descriptions: Default
N: Number of host nodes in the network: 100
t: Number of distinct items in the system: 1000
ϵ: Threshold: 0.5
s: Summary size: 5%

6.4.1.1 Data Sets

We use zipf distribution (with data skewness parameter α) to model the distribution of values among items at a specific time k. The number of distinct items in the system is 1000. We generate $1000 \cdot n$ instances of these items with their frequencies (global values) following zipf-distribution. We then randomly distribute these $1000 \cdot n$ instances of items to the N host nodes.

For every 5 time units, we randomly draw a value from $[0, 10]$ as the skewness α for the zipf-distribution. We randomly generate the item set following zipf-distribution with the aforementioned α setting and allocate these items among the host nodes as described above. For other time units, we vary the local values of 10% of the items in the local item set at each host node by 1. For each run, we simulate for 500 time units in total.

6.4.1.2 Setting of Localwavelet and Histogram

We explain how to set the summary size (denoted as s) for localwavelet and histogram. We denote the number of items in a local data set A_j^k as $|A_j^k|$. Each item

is represented by a pair of ⟨identifier, local value⟩. In the simulation, we use 8 bytes to represent the item identifiers and local values, respectively. Therefore, to represent the local item set at host node j, we need $16 \cdot |A_j^k|$ bytes. The total number of bytes (denoted by s_t) required to represent the local item sets of all the host nodes is $16 \cdot \sum_{j=1}^{N} |A_j^k| = 16 \cdot N \cdot |\bar{lA}|$ ($|\bar{lA}|$ is the average size of the local item set at a host node).

In localwavenet, each wavelet coefficient is represented by a pair of ⟨position, value⟩. Note that if we propagate the complete wavelet to the coordinator, we do not need to specify the position of a wavelet coefficient. However, we only propagate a number of the most significant wavelet coefficients (i.e., localwavelet) to the coordinator. Therefore, we need to specify the positions of the chosen wavelet coefficients so that the coordinator can perform merging on the localwavelets correctly. Again, we use 8 bytes to represent the position and value of a wavelet coefficient, respectively. x% summary size using wavenet means the total size of data summaries at all the host nodes is $x\% \cdot s_t$. As discussed in Section 6.3.1.2, each host node can set its summary size for localwavelet according to its local item set. Here for the purpose of comparison with the histogram approach, we set the summary size at all of the host nodes to the same value. Therefore, the summary size at each host node is $x\% \cdot s_t/N$. Since $s_t = 16 \cdot N \cdot |\bar{lA}|$, and each wavelet coefficient requires 16 bytes to represent, x% summary size means that $x\% \cdot |\bar{lA}|$ number of the most significant wavelet coefficients form the localwavelet at a host node.

For histogram, each bucket is represented by a pair of ⟨index, bucket aggregate⟩. Unlike in wavenet, the bucket aggregates for all the buckets are propagated to the coordinator, and thus the indexes of the buckets do not need to be propagated. The bucket

aggregate is represented by an 8-bytes number. Thus, x% summary size using histogram approach means that there are $2 \cdot x\% \cdot |\bar{lA}|$ buckets in the histogram.

6.4.2 Performance Metrics

Before discussing the performance metrics, we first define some terms. *True hit* is a distribution change with respect to ϵ that is reported by the system. *Missed hit* is a distribution change with respect to ϵ that is not reported. On the other hand, *false hit* is a change reported by the system that in fact is not a distribution change with respect to ϵ. We denote the number of true hits, missed hits and false hits as h_t, h_m and h_f, respectively.

Since the primary goal of this study is to design a distributed mechanism to efficiently monitor changes on the data distribution in the networks, we measure the performance of wavenet from two aspects, i.e., communication overhead and detection accuracy. We use relative communication cost to measures the communication overhead, and recall and precision to measure the detection accuracy.

- **Relative communication cost** is defined as the ratio of the bytes transmitted for monitoring changes on the data distribution in the networks to the total size of the local item sets at all the host nodes.

- **Recall** is defined as the ratio of the number of true hits to the total number of distribution changes with respect to ϵ, i.e, $recall = h_t/(h_t + h_m)$.

- **Precision** is defined as the ratio of the number of true hits to the total number of hits reported by the system, i.e, $precision = h_t/(h_t + h_f)$.

To better capture the relative communication overhead incurred by wavenet compared to the naive approach (where all the local item sets are propagated to the coordinator), we use the relative communication cost defined above as the performance metric. A high recall indicates a large percentage of the true hits are detected/reported by the system, i.e., the ratio of missed hits is small. A high precision indicates a large percentage of the hits reported by the system are true hits, i.e., the ratio of false hits is small. Note that in most applications, especially the ones involved in network attack detection, a missed hit is more devastating than a false hit, i.e., high recall is desirable.

6.4.3 Results

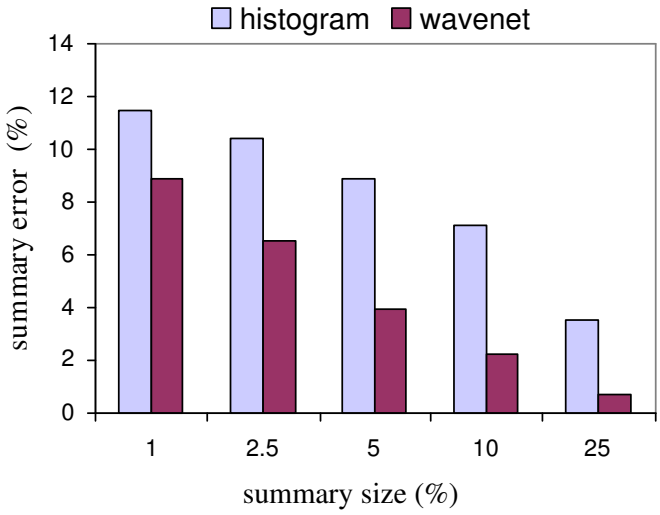
We first examine the summary errors introduced by wavenet under different summary sizes. We then evaluate the performance of wavenet under different settings of data skewness (α) and threshold (ϵ). Following that, we illustrate the effect of adaptive monitoring and localwavelet refinement. Except in the experiments demonstrating the effect of localwavelet refinement, we assume that each item in the original data domain is valid, i.e., the valid item set is the same as the original item set, and thus localwavelet refinement is disabled. Similarly, except in the experiments demonstrating the effect of adaptive monitoring, we disable local filters. As explained above, we set the summary size at all the host nodes to the same value. Therefore, when the local filters are disabled, the relative communication cost is exactly the same as the summary size. For instance, if the summary size is set to 1% of the size of the local item set, the relative communication cost with local filters disabled is 1% compared to the communication cost incurred by

the naive approach. For presentation brevity, we only show the results on the relative communication cost explicitly when the local filters are enabled.

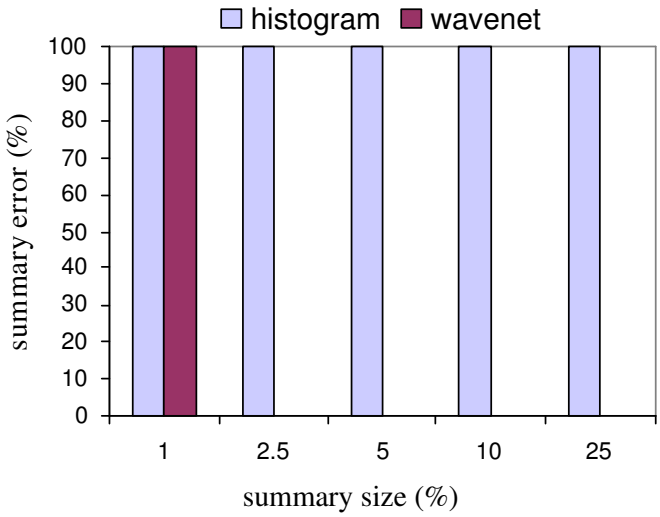
6.4.3.1 Summary Errors

In order to compare the summary errors introduced by histogram summary and localwavelet, we vary the summary size from 1% to 25%. In the experiment, we set $|A_1| = |A_2| = ... = |A_N| = t$. Thus, $|\bar{lA}| = t$.

Figure 6.8 shows the results under different data distribution (with skewness parameter α set to $1, 10$, respectively). The results under uniformly distributed item set ($\alpha = 0$) are not shown in the figure since the values are too small to be readable. When $\alpha = 0$, with even very small summary size (1%), both the localwavelet and histogram summary can approximate the item set pretty well (the summary error is below 0.1%). When the data distribution is relatively skewed ($\alpha = 1$), the summary errors introduced by localwavelet and histogram summary increase. In addition, we observe that larger summary errors are introduced by histogram summary. When the data distribution becomes more skewed ($\alpha = 10$), localwavelet with very small summary size can still approximate the item set well. For instance, with 2.5% summary size, the summary errors introduced by localwavelet is below 0.1%. On the other hand, even when the summary size is as large as 25%, the summary error introduced by histogram summary is still as high as 99.86%. This confirms our discussion in Section 6.3.1 that wavelet can compress both uniformly distributed item set and skewly distributed item set while histogram can not compress skewly distributed item set well.



(a) $\alpha = 1$



(b) $\alpha = 10$

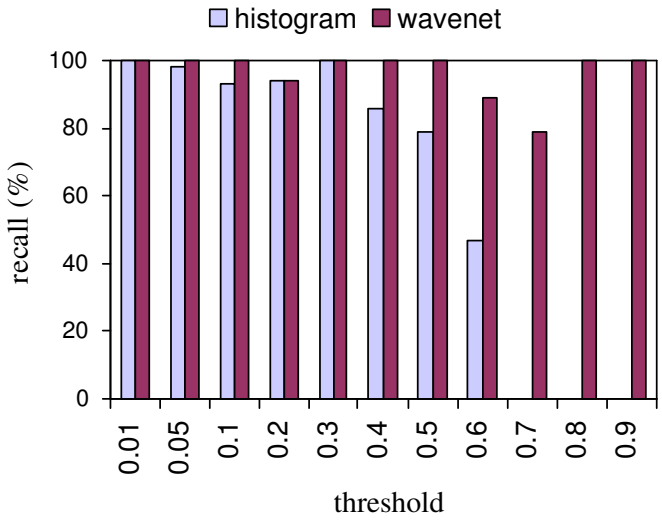
Fig. 6.8. Summary errors introduced by wavenet and histogram.

6.4.3.2 Effect of the Threshold

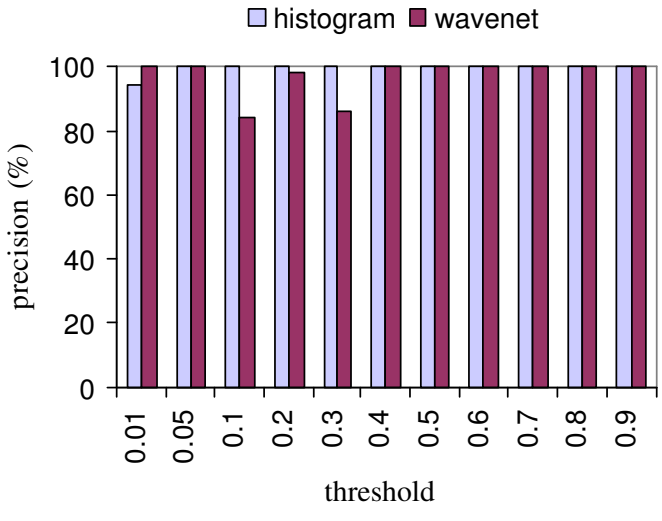
Small threshold requires the system to detect small changes on the data distribution, and large threshold requires the system to detect significant changes on the data distribution. Since wavelet can approximate both uniformly distributed item set and skewly distributed item set, we expect that wavenet performs well under different thresholds. On the other hand, histogram can not compress skewly distributed data set well (as confirmed in Section 6.4.3.1), and thus we expect that it can not detect significant changes on the data distribution as discussed in Section 6.3.1.1.

In this set of experiments, we vary the threshold from 0.01 to 1 (the summary size is set to 5%). Figure 6.9(a) and (b) show the recall and precision of wavenet and histogram, respectively. Since there is no true hit when the threshold is 1, we omit the results with threshold as 1 from the figure. Figure 6.9(a) shows that wavenet consistently achieves high recall (close to 100% in most cases). On the other hand, histogram has similar recall as wavenet does when the threshold is small, but has much lower recall than wavenet does when the threshold is large. For instance, with threshold as 0.6, the recall using histogram drops to 47%. When the threshold increases beyond 0.6, the recall using histogram drops to 0.

As shown in Figure 6.9(b), the precision using wavenet and histogram are close to 100% in majority of the tested cases. The high precision achieved by histogram is not surprising since histogram smoothes out the skewness in the item set and tends to have the estimated entropy lower than its exact value. Therefore, if histogram reports a hit, it is very likely that it is a true hit. As we discussed in Section 6.4.2, in practice,



(a) recall



(b) precision

Fig. 6.9. The effect of threshold on wavenet.

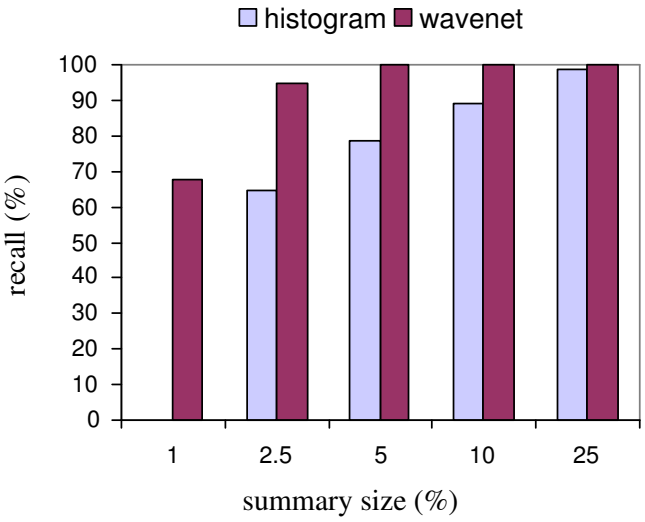
high recall is generally more critical than high precision. The consistently high recall (and high precision) achieved by wavenet confirms the practical value of our proposed technique.

6.4.3.3 Effect of the Summary Size

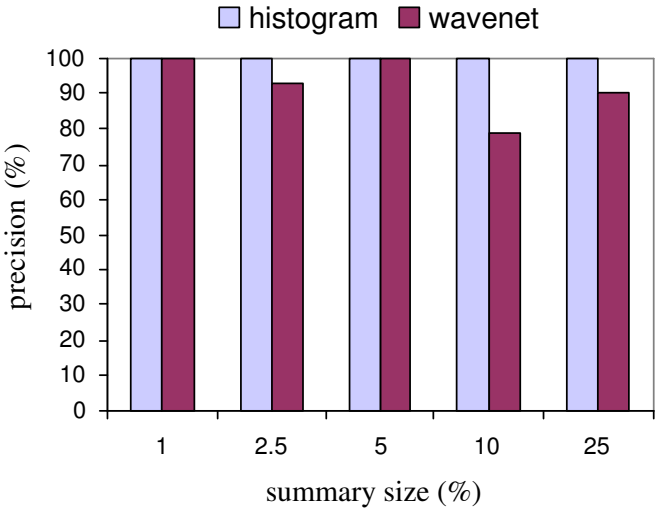
In this set of experiments, we vary the summary size from 1% to 25% and observe the recall and precision of wavenet and histogram, as depicted in Figure 6.10(a) and (b), respectively. From Figure 6.10(a), we can see that when the summary size is small (1%), histogram performs very poorly and has recall as 0%. On the other hand, even with such a small summary size, wavenet achieves 68% recall. When the summary size increases, the recall of both wavenet and histogram increases, which is expected. From Figure 6.10(b), we can see that the precision of both wavenet and histogram is sufficiently high ($\geq 80\%$).

6.4.3.4 Effect of Adaptive Monitoring

We evaluate the effect of adaptive monitoring by comparing the communication costs incurred with the local filters turned on and turned off in Figure 6.11 (the precision and recall are not affected by adaptive monitoring significantly and the results are omitted). In this set of experiments, we set the summary size to 5% and vary the threshold. When the local filters are turned off, the communication cost using both wavenet or histogram is equivalent to the summary size, i.e., 5% of cost incurred by the naive approach. In Figure 6.11, we show the ratio of the communication cost with the local filters turned on to the one with the local filters turned off under different thresholds.



(a) recall



(b) precision

Fig. 6.10. The effect of summary size on wavenet.

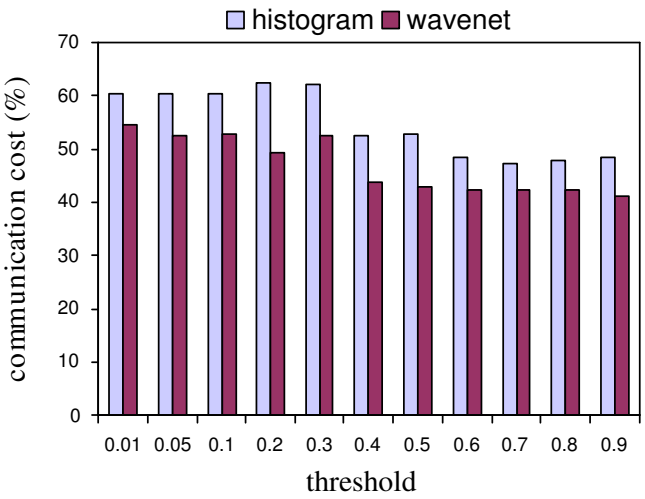


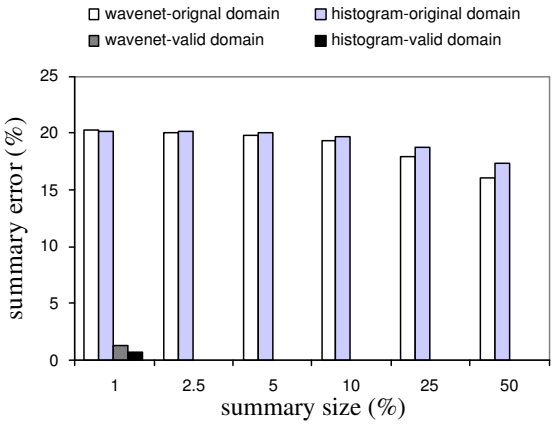
Fig. 6.11. The effect of adaptive monitoring on wavenet.

From this figure, we can see that when the threshold increases, the communication cost with local filters turned on decreases. This is self-explanatory since large threshold allows more room for changes on the values of items without incurring distributed changes with respect to ϵ. In addition, the saving on the communication cost using wavenet is larger than the saving using histogram.

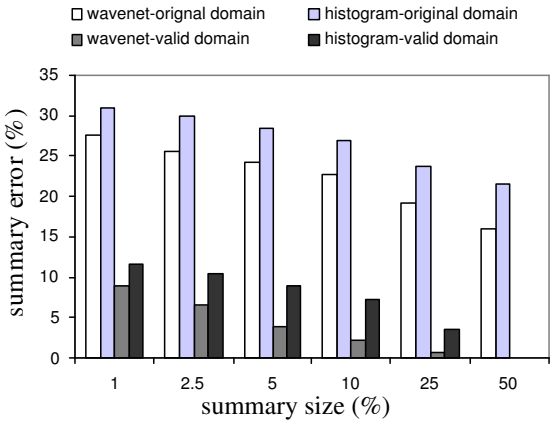
6.4.3.5 Effect of Localwavelet Refinement

We evaluate the effect of localwavelet refinement from two aspects. We first demonstrate the need for localwavelet refinement in the case when the data domain is sparsely populated by items. We then demonstrate the effectiveness of our proposed localwavelet refinement.

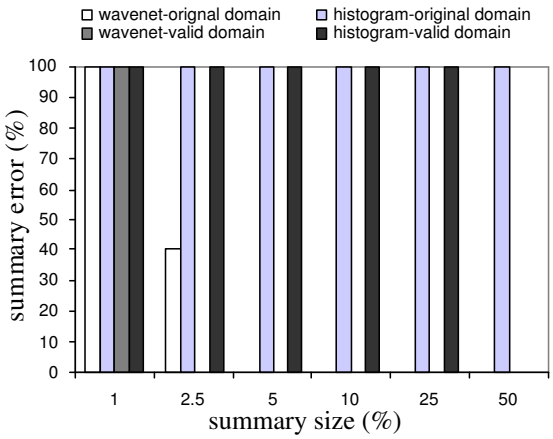
To demonstrate the need for localwavelet refinement, we compare the summary errors introduced by localwavelet and histogram summary constructed on the original sparsely populated data domain and on the valid item set. We set the original data domain (number of distinct items including valid items and invalid items) as 32000 and the number of valid items as 1000. In another words, only around 3% of the data domain is populated by items. We obtain the summary errors introduced by histogram summary and localwavelet in the original data domain (with 32000 items) and in the valid item set (with 1000 valid items), respectively. Figure 6.12 shows the results under different data distributions ($\alpha = 0, 1, 10$). The general trends we observe from this set of experiments is that the summary errors introduced by localwavelet and histogram summary constructed in the original sparsely populated data domain are much larger than the ones introduced by localwavelet or histogram summary constructed in the valid item set. For instance,



(a) $\alpha = 0$

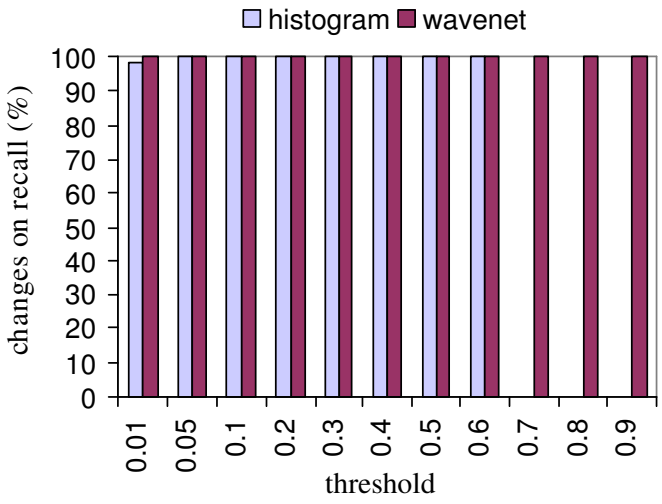


(b) $\alpha = 1$

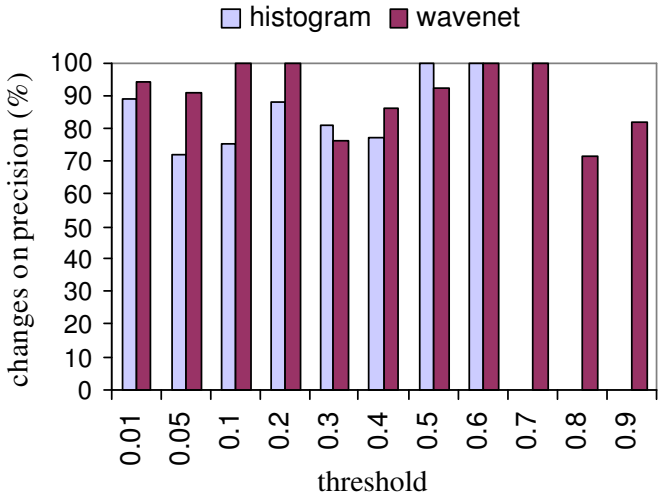


(c) $\alpha = 10$

Fig. 6.12. Comparing the summary errors introduced by summarization in the original data domain and the valid data domain.



(a) recall



(b) precision

Fig. 6.13. Results of localwavelet refinement.

when the data distribution is uniform ($\alpha = 0$), the summary errors introduced by a small sized histogram summary or localwavelet (with summary size set to 1%) constructed in the valid item set are very small (below 0.1%). However, even when the summary size is 50% of the size of local item set, the summary errors introduced by localwavelet and histogram summary constructed in the original data domain are still as high as 20%.

To demonstrate the effectiveness of our proposed localwavelet refinement, we compare the recall and precision obtained with localwavelet refinement enabled (i.e., the histogram summary and localwavelet are constructed in the virtual valid item set) with the ones obtained under the ideal case where the valid item set is known to all the host nodes (i.e., the histogram summary and localwavelet are constructed on the valid item set). The results are shown in Figure 6.13(a) and (b). The y-axis shows the ratio of the recall (precision) obtained with localwavelet refinement enabled to the recall (precision) obtained under the ideal case. From this figure, we can see that the recall and precision obtained with our localwavelet refinement technique are very close to the ones obtained under the ideal case. This confirms the effectiveness of our proposed localwavelet refinement technique. In addition, it also confirms that obtaining the approximate number of valid items in the system is sufficient for localwavelet refinement as pointed out earlier in Section 6.3.3.

6.5 Summary

A massive amount of data is collected and stored at a large number of host nodes connected via wired networks or wireless networks. Users are normally interested in some "interesting" information, such as anomaly or patterns, hidden among the data

rather than the raw data. In this study, we investigate monitoring changes on the data distribution in the networks (MCDN), which is a prevailing task in network anomalies detection, query optimization on networked data and various distributed data mining tasks. The unique characteristics of the data and system under consideration, such as decentralization, large scale, continuity, and real time, make this issue extremely challenging.

We propose a distributed monitoring framework, called wavenet, to summarize the data at local host nodes into compact yet accurate data summaries, called localwavelets, for communication with the coordinator. In addition, we propose efficient solutions to address various issues involved in the design of wavenet, i.e., localwavelet construction in a sparsely populated data domain and localwavelet propagation. Extensive evaluation demonstrates the efficiency of wavenet. According to the authors' best knowledge, this is the first study investigating MCDN.

In the future, we plan to investigate predicting the changes on the data distribution and incorporate prediction into the proposed monitoring framework. In addition, we plan to deploy wavenet and evaluate its performance in real world applications, e.g., network attack detection.

Chapter 7

Distributed Clustering

In this study, we investigate clustering, one of the most important data mining tasks, in P2P systems. The lack of a central control and the sheer large size of P2P systems make the existing clustering techniques not applicable here. We propose a fully distributed clustering algorithm, called peer density-based clustering (PENS), which overcomes various challenges raised in performing clustering in peer-to-peer environments, including cluster assembly and cluster membership storage. Additionally, an important feature of our proposal is incremental clustering. PENS consists of three components, 1) *hierarchical cluster assembly* which enables peers to collaborate in forming a global clustering model without requiring a central control or message flooding, 2) *multi-granularity based cluster membership storage* which facilitates easy access to cluster membership with reasonable maintenance and storage overheads, and 3) *incremental PENS* which enables incremental clustering upon data insertion/deletion in P2P systems. The correctness of the algorithm is proven and the complexity analysis of the algorithm demonstrates that PENS can discover clusters and noise efficiently in P2P systems.

7.1 Introduction

Clustering, which groups a set of data objects into clusters of similar data objects, can be applied in many different problem domains, such as spatial data analysis, scientific pattern discovery, document categorization, taxonomy generation, customer/market analysis, etc. As discussed in Chapter 2.1.2 and 2.3, various clustering techniques have been proposed for either centralized systems or distributed systems. However, these existing clustering techniques are not applicable to P2P systems. This study re-examines the problem of clustering in P2P systems, identifies critical challenges involved, and provides solutions to overcome these challenges.

There are a couple of challenging issues that need to overcome in order to efficiently support clustering in P2P systems.

- A general idea for clustering in P2P systems is that peers perform clustering on the data objects held locally and then collaboratively assemble the locally obtained clusters to form a global clustering model. The lack of a central site and infeasibility of network-wide flooding makes it extremely challenging for peers to conduct *cluster assembly*.

- The lack of a central site also makes it extremely difficult for storing the clustering results (cluster membership) in the system, i.e., *cluster membership storage*.

- While the data objects in P2P systems are dynamic in nature, it is essential to perform *incremental clustering* upon data insertion/deletion to avoid system-wide re-clustering.

We propose a fully distributed clustering algorithm, called peer density-based clustering (PENS) algorithm, that overcomes above-mentioned challenges. PENS follows the design principle of DBSCAN ([41, 42]), a well-known density-based clustering algorithm[1]. We illustrate our proposal in CAN [106], which is a well-accepted P2P overlay network supporting multi-dimensional data[2]. PENS consists of three components, i.e., *hierarchical cluster assembly (HCA)*, *multi-granularity based cluster membership storage (MGCMS)*, and *incremental PENS (increPENS)*. HCA forms a global clustering model through peer communications only without requiring a central site or message flooding. Peers collaboratively assemble clusters progressively along a hierarchy leveraged by the CAN overlay. MGCMS facilitates easy access to cluster membership yet avoids excessive maintenance and storage overheads by maintaining at each peer cluster membership of different granularities. IncrePENS enables efficient incremental clustering upon data insertion/deletion in P2P systems by systematically identifying and updating the clusters affected by the insertion/deletion of a data object. We also present optimization techniques applicable to PENS. The correctness of the proposed algorithm is proven. In addition, we analyze the message complexity of PENS. The analytic result indicates that PENS performs density-based clustering in P2P systems efficiently.

The primary contributions of this paper are three-fold:

[1]We select DBSCAN for investigation is because DBSCAN is efficient in very large databases. It requires minimum domain knowledge to determine input parameters and discovers clusters with arbitrary shapes.

[2]While our proposed algorithm can be applied to other P2P overlays supporting multi-dimensional data, e.g., SSW [84], we choose the simple overlay structure CAN for the clarity of presentation.

1. We identify critical research challenges involved in clustering on P2P systems. To the best of our knowledge, this is the first attempt to investigate the problem of *P2P clustering*.

2. We propose an efficient distributed clustering algorithm, PENS, to facilitate density-based clustering in P2P systems.

3. We prove that the proposed algorithm is correct. In addition, our analytic result demonstrates the efficiency of the algorithm.

The rest of this paper is structured as follows. Related works and background are provided in Section 7.2. We present the details of our proposal, PENS, in Section 7.3. The correctness and the complexity analysis of the proposal are shown in Section 7.4 and 7.5, respectively. Finally, we conclude this paper and outline directions for future research in Section 7.6.

7.2 Preliminaries

In this section, we review some works relevant to our study and provide backgrounds on DBSCAN clustering algorithm and CAN overlay.

7.2.1 Background

In this section, we review DBSCAN and CAN that is necessary for understanding of our proposal.

7.2.1.1 DBSCAN

DBSCAN is a representative algorithm for density-based clustering, which treats the regions in the data space that are densely populated by data as clusters. In the following, we give a short introduction of the basic DBSCAN algorithm and the incremental variant[3]. For details of the basic and incremental DBSCAN algorithms, please see [42] and [41], respectively.

Basic DBSCAN Algorithm: The key idea of basic DBSCAN algorithm is that if the neighborhood of a given radius (ε) for a data object has a cardinality exceeding a preset threshold value (T), this data object belongs to a cluster. In the following, we list the definitions of terminologies regarding to density-based clustering from [42] for convenience of presentation.

Definition 1: (directly density-reachable) [42] An object p is *directly density-reachable* from an object q wrt. ε and T in the set of objects D if 1) $p \in N_\varepsilon(q)$ ($N_\varepsilon(q)$ is the subset of D contained in the ε-neighborhood of q.) 2) $|N_\varepsilon(q)| \geq T$.

Objects satisfying Property 2 in above definition are called *core objects* and the property is called *core object property*.

Definition 2: (density-reachable) [42] An object p is *density-reachable* from an object q wrt. ε and T in the set of objects D, denoted as $p >_D q$, if there is a chain of objects $p_1, ..., p_n$, $p_1 = q$, $p_n = p$ such that $p_i \in D$ and p_{i+1} is directly density-reachable from p_i wrt. ε and T.

[3]We refer the DBSCAN algorithm without incremental clustering feature as the basic DBSCAN algorithm.

It is possible that two objects of a cluster residing along the cluster boundary might not be density-reachable from each other. However, there must exist a third object, from which these two objects are density-reachable. Therefore, the notion of density-connectivity is introduced.

Definition 3: (density-connected) [**42**] An object p is *density-connected* to an object q wrt. ε and T in the set of objects D if there is an object $o \in D$ such that both p and q are density-reachable from o wrt. ε and T in D.

While density-reachability is a transitive but not symmetric relation, density-connectivity is both a transitive and symmetric relation. Figure 7.1 depicts the relations on some sample points where ε is the radius of the circles and T is 5.

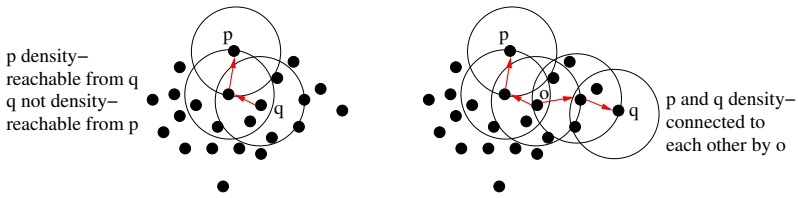


Fig. 7.1. Density-reachability and density-connectivity.

A cluster is defined as a set of density-connected objects which is maximal with respect to density-reachability and the noise is the set of objects not contained in any cluster.

Definition 4: (cluster) [**42**] Let D be a set of objects. A *cluster* C wrt. ε and T in D is a non-empty subset of D satisfying the following conditions: 1) Maximality: $\forall p, q \in D$: if $q \in C$ and $p >_D q$ wrt. ε and T, then also $p \in C$. 2) Connectivity: $\forall p, q \in C$: p is density-connected to q wrt. ε and T in D.

Definition 5: (noise) [**42**] Let $C_1, ..., C_k$ be the clusters wrt. ε and T in D. Then, the noise is defined as the set of objects in the database D not belonging to any cluster C_i, i.e., noise = $\{p \in D | \forall i$: $p \notin C_i\}$.

Note that not all objects in a cluster are core objects. The objects in a cluster that are not core objects are called *border objects*. Different from *noise objects* that are not density-reachable from any other objects, border objects are density-reachable from some core object in the cluster. In the following discussion, we omit "wrt. ε and T" when no confusion is caused.

The basic DBSCAN algorithm efficiently discovers clusters and noise in a dataset according to above definitions. According to [42], a cluster is uniquely determined by any of its core objects (Lemma 2 in [42]). Based on this fact, the basic DBSCAN algorithm starts from any arbitrary object p in the database D and obtains all data objects in D that are density-reachable from p through successive region queries, which return all data objects intersecting with the specified query regions. If p is a core object, the obtained set is a cluster in D containing p. On the other hand, if p is either a border object or noise, the obtained set is empty and p is assigned to the noise. The above procedure is then invoked with next object in D that has not been examined before. This process continues till all data objects are examined.

Incremental DBSCAN Algorithm: An insertion of a data object might expand a cluster or merge two or more clusters into one larger cluster, while a deletion of a data object might shrink a cluster or divide a cluster into two or more smaller clusters. The incremental DBSCAN algorithm identifies which part of the data set is affected by the insertion/deletion and updates the clusters accordingly.

According to the discussion in [41], the insertion/deletion a data object (p) only affects the neighborhood of p. Moreover, only the "seed" objects, denoted as *UpdSeed*, which are core objects in the ε-neighborhood of those objects in $N_{\varepsilon}(p)$ that change their core object properties as a result of the update, need to reapply DBSCAN. UpdSeed is defined as follows.

Definition 6: (seed objects for the update) **[41]** Let D be a set of objects and p be an object to be inserted or deleted. Then,

$UpdSeed\_ins = \{q | q$ is a core object in $D \cup \{p\}, \exists q'$: q' is core object in $D \cup \{p\}$ but not in D and $q \in N_{\varepsilon}(q')\}$.

$UpdSeed\_del = \{q | q$ is a core object in $D - \{p\}, \exists q'$: q' is core object in D but not in $D - \{p\}$ and $q \in N_{\varepsilon}(q')\}$.

UpdSeed is obtained as follows. A region query is first invoked to obtain the data objects in $N_{\varepsilon}(p)$. Then for each of the data object in $N_{\varepsilon}(p)$ with core object property changed, a region query is invoked to obtain the core data objects in its ε-neighborhood.

7.2.1.2 Content Addressable Network (CAN)

CAN organizes the logical data space as a k-dimensional Cartesian space and partitions the space into *zones*, each of which is taken charge of by one or more peers,

called as *zone owners*. When a new node joins, it obtains a random point in the space and joins the zone covering the random point. The owner of the zone to be joined splits its zone into two equal-sized sub-zones along the dimension chosen in round robin fashion. A data object is mapped to a point in the space and the index for the data object is stored at the peer whose zone covers the corresponding point. In addition to indexing data objects, peers maintain routing tables which consist of pointers to neighboring subspaces along each dimension. Routing from a source zone to a destination zone is performed greedily by always forwarding the message to the peer in the routing table that is closest to the destination zone.

CAN provides two fundamental operations: *PUT* and *GET*. When a peer has a new sharable data object a, it invokes PUT(a) to publish the index information for a to the owner of the zone covering a. When a peer wants to retrieve the index information for a data object a, it invokes GET(a) to obtain this information from the owner of the zone covering a.

7.2.2 System Model

Assume that there are N peer nodes in the system, where each peer node i has a dataset U_i ($i = 1, 2, ..., N$) consisting of n_i ($n_i \gg 1$) data objects represented by their k-dimensional feature vectors. Note that the number of data objects ($n = \sum_1^N n_i$) in the system is much greater than the number of nodes in the system, i.e., $n \gg N$. The union of the dataset U_i is U and the domain of U is $D \in R^k$, where R is the set of real numbers in the range of $\{0, 1\}$[4]. Each data object $x \in D$ is represented by a

[4]We can always normalize the attributes with domain not in the range of $\{0, 1\}$.

feature vector $x = \{x_1, x_2, ..., x_k\}$. Without loss of generality, we assume the distance (dissimilarity) between two data objects, $d(x, y)$ where $x, y \in D$, is their Euclidean distance, i.e., $\sqrt{\sum_{j=1}^{k}(x_j - y_j)^2}$.

A data object is mapped to a point in a k-dimensional Cartesian space. The N peer nodes form into a k-dimensional CAN overlay over this Cartesian space. We modify the CAN overlay slightly as follows. When a peer joins the system, the owner of the zone to be joined splits its zone into two only if the total number of data objects mapped to this zone exceeds some threshold value M. With this minor modification, a zone is not splitted unless it is overloaded with data objects. In following discussions, only the feature vectors of data objects are required for clustering. Therefore, for brevity, we refer the feature vector of a data object as the data object itself. In addition, we refer the data objects mapped to the zone of a peer as this peer's data objects whenever the context is clear.

7.3 Peer Density-based Clustering

We design a fully distributed density-based clustering algorithm in P2P systems, called peer density-based clustering (PENS). PENS addresses three challenging issues raised by clustering in P2P systems, i.e., cluster assembly, cluster membership storage, and incremental clustering, through HCA (hierarchical cluster assembly), MGCMS (multi-granularity based cluster membership storage), and increPENS (incremental PENS), respectively.

To perform clustering over the whole set of data objects stored in P2P systems, each peer first conducts clustering over the data objects mapped to its zone to obtain

local clusters or noise within its zone. Then peers invoke HCA to assemble local clusters progressively to form a global clustering model. The obtained clustering result, i.e., the cluster membership, is stored at peers in accordance with MGCMS. Upon the insertion/deletion of a data object, increPENS is invoked for incremental clustering.

In the following, we further introduce some definitions and terms used in the remaining discussions. We then present the details of HCA, MGCMS and increPENS in Section 7.3.1, 7.3.2 and 7.3.3, respectively. Lastly, we discuss some optimization technique applicable to PENS.

In the following, "region" refers to a portion of the data space. It may consist of one single zone or multiple neighboring zones. The precise definition of region is deferred to Section 7.3.1.1.

Definition 7: (cluster within Region i) Let D_i be the set of data objects mapped to Region i. A *cluster within Region i*, C, wrt. ε and T is a nonempty subset of D_i satisfying the following conditions: 1) Maximality: $\forall p, q \in D_i$: if $q \in C$ and $p >_{D_i} q$ wrt. ε and T, then also $p \in C$. 2) Connectivity: $\forall p, q \in C$: p is density-connected to q wrt. ε and T in D_i.

Definition 8: (noise within Region i) Let $C_1, ..., C_k$ be the clusters within Region i wrt. ε and T. Then, the *noise within Region i* is defined as the set of objects in the database D_i not belonging to any cluster C_i within Region i, i.e., noise = $\{p \in D_i | \forall j$: $p \notin C_j\}$.

Definition 9: (cluster covering Zone i) Let D_i be the set of data objects mapped to Zone i and D be the union of all D_i $(1 \leq i \leq N)$. Let C be a cluster in D. C is a *cluster covering Zone i* if $\exists p \in C$: $p \in D_i$.

Recall that to perform clustering in PENS, before invoking HCA, peers first perform clustering locally based on the basic DBSCAN algorithm as introduced in Section 7.2.1.1[5]. When peers finish clustering locally, the data objects in each zone are classified as either local clusters or noise within a zone. Each local cluster is given a unique LocalClusterID, denoted by <ZoneID, ClusterID>, where ZoneID is the unique identifier of a zone (how to obtain the ZoneID is described later) and ClusterID is the unique identifier of a local cluster within the zone. We also given a unique LocalClusterID to data objects that are considered noise locally.

7.3.1 Hierarchical Cluster Assembly

While a central site is not available and network-wide message flooding is impractical in P2P systems, it is extremely challenging to assemble local clusters and obtain a global clustering model. Hierarchical clustering assembly (HCA) is proposed to address this issue. HCA enables peers to collaboratively form a global clustering model through peer communication only. The basic idea of HCA is that the clusters within smaller regions (or zones) are assembled to form clusters within larger regions. These clusters are then assembled to form clusters within even larger regions. This hierarchical process continues until the clusters in the whole data space are formed. HCA consists of three tasks:

- **Hierarchy Formation:** To form a hierarchy in P2P systems which is used for cluster assembly.

[5]In this paper, we adopt the heuristics developed in [42] to obtain the setting for ε and T from the "thinnest" region of a cluster. For details on ε and T settings, please refer to [42].

- **Cluster Expansion Check**: To determine whether locally obtained clusters can be expanded to other zones and obtain the corresponding cluster expansion information.

- **Cluster Merging:** To merge clusters according to cluster expansion information along the hierarchy to obtain a global clustering model.

In the following, we address these three tasks in details.

7.3.1.1 Hierarchy Formation

We observe that the CAN overlay can be leveraged to form a hierarchical structure, which can be used for cluster assembly. We call this hierarchical structure as *virtual peer tree (VPtree)*. In the following, we introduce the concept of VPtree and some relevant terms.

The history of space partition during CAN overlay construction can be represented by a virtual binary tree. The root of the tree represents the initial data space and the two subtrees of each node represent the two subspaces generated by a partition event. We encode the binary tree by assigning bit 0 and 1 to the left and right edges of each tree node, respectively. A tree node obtains a tree label by concatenating the edge labels on the path from the root to itself. The tree label uniquely identifies a node in the tree. This encoded binary tree is the VPtree. Note that the VPtree is virtual only and it is not stored anywhere in the system.

The two equal-sized subspaces generated from a partition event are called *buddy regions*. We use *region* as a general term to represent the subspace consisting of one

single zone or two buddy regions. The regions that undergo x partitions correspond to the nodes at depth x in the VPtree, and they are called *level-x regions* (thus, the leaf nodes of VPtree are also zones). For instance, the whole data space is level-0 region and the two buddy regions generated from the first partition (corresponding to the two subtrees of the root node) are level-1 regions. The tree label for a node in the VPtree serves as the *RegionID* for the corresponding region of an internal tree node, or the *ZoneID* for the corresponding zone of a leaf node.

Figure 7.2 depicts the VPtree structure and the corresponding regions for a CAN in 2-dimension Cartesian space. The Cartesian space is partitioned along Dimension 1 and 2 (marked as $d1$ and $d2$ in the figure) in round robin fashion. In this example, level-5 Zones M (with ZoneID 11110) and N (with ZoneID 11111) are buddy zones. Similarly, level-4 Zone J (with ZoneID 1110) and the region (with RegionID 1111) composed of Zone M and N are buddy regions.

7.3.1.2 Cluster Expansion Check

The task of cluster expansion check is to determine whether and how local clusters within a zone can be expanded to other zones and obtain the corresponding cluster expansion information. In the following, we first discuss some observations, which facilitate cluster expansion check. We then explain how cluster expansion check is performed in details.

After careful examination, we have following two observations:

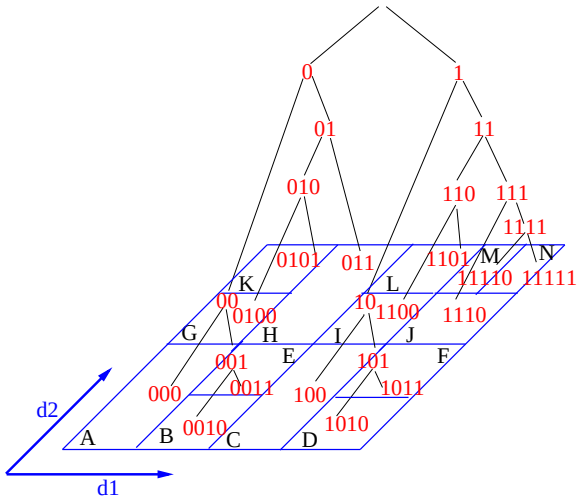


Fig. 7.2. Illustrative example of VPtree.

- If all data objects of a cluster within Zone i are at least ε away from the boundary of Zone i, this cluster is also a cluster in the whole data space, i.e., this cluster is *non-expandable*.

- If a cluster within Zone i can be expanded to include other data objects outside of Zone i, i.e., this cluster is *expandable*, there exists at least one data object in this cluster whose ε-neighborhood contains some data objects outside of Zone i.

We call the region inside Zone i that is within ε to the boundary of Zone i as *ε-inner-boundary* of Zone i, and the region outside of Zone i that is within ε to the boundary of Zone i as *ε-outer-boundary* of Zone i. Observation 1 implies that if no data object of a cluster within a zone resides in the ε-inner-boundary of the zone, this cluster is not expandable. Observation 2 implies that if a cluster within a zone can be expanded to

other zones, there must exist some data objects of this cluster in the ε-inner-boundary of this zone whose ε-neighborhood contains some data objects in the ε-outer-boundary of this zone. In the following discussions, we refer the ε-inner-boundary of a zone as the ε-inner-boundary and ε-outer-boundary of a zone as the ε-outer-boundary whenever the context is clear. Figure 7.3 illustrates examples of an expandable cluster (A2) and a non-expandable cluster (A1) within Zone A (depicted by the rectangle). The shaded areas indicate the ε-inner-boundary and ε-outer-boundary of Zone A, respectively.

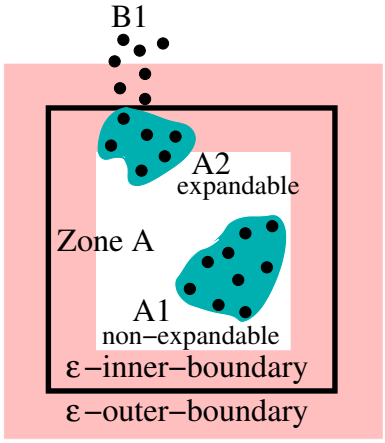


Fig. 7.3. Expandable or non-expandable clusters.

The above two observations can be generalized to following two formal theorems:

THEOREM 7.1. *Let D, D_i and D_{i_in} be the set of data objects in the whole data space, the set of data objects mapped to Zone i, the set of data objects mapped to the ε-inner-boundary, respectively. Let C be a cluster within Zone i. If $\forall p \in C$: $p \in D_i - D_{i\_in}$, C is a cluster in D.*

Proof: We prove Theorem 7.1 by contradiction. Suppose that Cluster C is not a cluster in D, i.e., C does not satisfy maximality or connectivity in D.

From the definition of clusters within a region (Definition 7), we know that C satisfies connectivity in D_i, from which we can derive that C must satisfy connectivity in D. Thus, if C is not a cluster in D, C should not satisfy maximality in D. In the following, we prove this is impossible, i.e., C must also satisfy maximality in D.

If C does not satisfy maximality in D (C satisfies maximality in D_i according to Definition 7), there exists two data objects p and q, $p \notin D_i$, $q \in D_i$ and q is core object in C such that p is density-reachable from q in D. Thus, there exists a chain of data object $p_1, ..., p_n$, $p_1 = q$ and $p_n = p$ such that p_{i+1} is directly density-reachable from p_i in D. Among $p_1, ..., p_n$, there exists x $(1 \leq x \leq n-1)$ such that $p_1, ..., p_x$ are inside Zone i and $p_{x+1}, ..., p_n$ are outside of Zone i. Since p_{x+1} is directly density-reachable from p_x, p_{x+1} is in the ε-neighborhood of p_x. Therefore, p_x must be located no greater than ε from the boundary of Zone i, i.e., p_x is in D_{i_in}. Since p_x is density-reachable from q, p_x belongs to Cluster C. This contradicts with the fact that no data objects of C are in D_{i_in}. This proves Theorem 7.1. □

THEOREM 7.2. *Let D_{i_in} and D_{i_out} be the set of data objects mapped to the ε-inner boundary and ε-outer-boundary of Zone i, respectively. Let C be a cluster within Zone*

i. If C can be expanded to other zones, there exists a pair of data objects p and q where $q \in C$, $q \in D_{i\_in}$ and $p \in D_{i\_out}$ such that p is directly density-reachable from q.

Proof: We prove Theorem 7.2 by contradiction. Suppose that there is no data object in D_{i_out} that is directly density-reachable from any data object of C in D_{i_in}.

In order for a data object p outside of Zone i to be density-reachable from a data object q of C in D_{i_in}, there should exist a chain of data objects $p_1, ..., p_n$, $p_1 = q$ and $p_n = p$ such that p_{i+1} is directly density-reachable from p_i in D. In addition, there should exist x $(1 \leq x \leq n-1)$ such that $p_1, ..., p_x$ are inside Zone i ($p_1, ..., p_x$ are in C) and $p_{x+1}, ..., p_n$ are outside of Zone i . If there is no data object in D_{i_out} that is directly density-reachable from any data object of C in D_{i_in}, we couldn't find such x. Thus, no data object outside of Zone i is density-reachable from any data object of C in D_{i_in}. Since among all data objects in Zone i, only those data objects in D_{i_in} might contain some data objects outside of Zone i in their ε-neighborhood, we can then derive that no data objects outside of Zone i is density-reachable from any data object of C. Thus C can not be expanded outside of Zone i, which contradicts with the fact. This proves the theorem. □

According to Theorem 7.2, in order to determine whether a local cluster within a zone can be expanded, we only need to examine whether this cluster is affected by the data objects in the ε-outer-boundary. In addition, only the data objects in the ε-inner-boundary might be affected, i.e., change their core object property (mentioned in Section 7.2.1.1), by these data objects in the ε-outer-boundary. Therefore, cluster

expansion check proceeds as follows. A peer first obtains the data objects in the ε-outer-boundary by a region query. Then through local computation, the peer examines the ε-neighborhood for each of the data objects in the ε-inner-boundary. There are four possible cases for each of such data object p:

1. Noise: $|N_{\varepsilon}(p)| < T$.
2. New cluster: $|N_{\varepsilon}(p)| \geq T$, and all data objects in $N_{\varepsilon}(p)$ are previous noise objects within zones.
3. Cluster extension: $|N_{\varepsilon}(p)| \geq T$, and some data objects in $N_{\varepsilon}(p)$ belong to one cluster within a zone, while others are noise objects within zones.
4. Cluster merging: $|N_{\varepsilon}(p)| \geq T$, and some data objects in $N_{\varepsilon}(p)$ belong to two or more clusters within zones while others are noise objects within zones.

Figure 7.4 illustrates the four cases. Note that different cases might coexist in some scenario. For instance, in Figure 7.4(d) the two clusters $A2$ and $B1$ are merged. In addition, some noise object is added to the newly merged cluster as well.

For the first case, we do not need to do anything. For the latter three cases, the owner of the zone needs to record cluster expansion information, formally denoted as *cluster expansion set (CES)*, which indicates how local clusters within a zone can be expanded outside of the zone. CES consists of cluster expansion entries for each local cluster that can be expanded to other zones. Each cluster expansion entry consists of two parts, i.e., *present coverage (Pcoverage)* and *expandable coverage (Ecoverage)*, where Pcoverage includes the LocalClusterID for a local expandable cluster within the

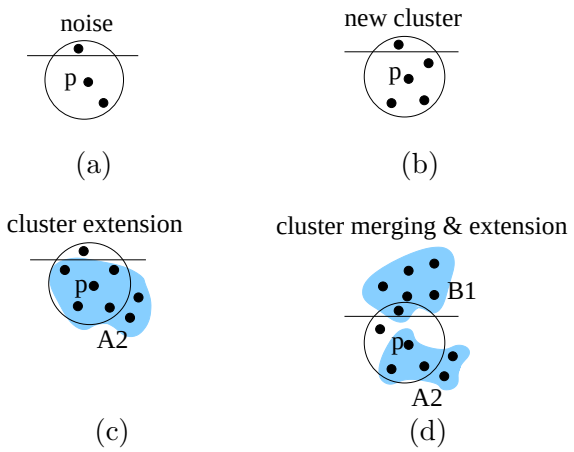


Fig. 7.4. Illustrative examples for cluster expansion check.

zone, and Ecoverage includes the LocalClusterIDs for the clusters or noise objects in the ε-outer-boundary that this local cluster can be expanded to.

7.3.1.3 Cluster Merging

Cluster merging assembles local clusters according to CESs obtained previously along the VPtree to form a global clustering model. To preform cluster merging, the CESs of two buddy zones are first delivered to their parent in the VPtree where the clusters within the two buddy zones are assembled to form larger clusters and the two CESs are merged to form a new CES for the region consisting of the two buddy zones. The CESs of two buddy regions are then delivered to their parent where the cluster assembly and CES merge take place accordingly. This process continues till the root of VPtree (corresponding to the whole data space) is reached. In order to perform above process, we need to address following four questions:

- Who should be the *arbiter* acting as the parent for merging clusters in two buddy regions?
- Who should be the *representative* to deliver the relevant information to the parent along the VPtree?
- How should the CESs be merged at an arbiter?
- What information should be forwarded along the VPtree?

In the following, we address these four questions and discuss the algorithmic details of cluster merging.

To minimize communication overheads, the arbiter for a region is chosen as the peer in charge of the zone within the specified region that is closest to the center of data space. The representative for a region is chosen among the arbiter for this region and the two representatives representing the two corresponding buddy regions. The selection is made according to the current processing load, bandwidth, etc., at these three peers (when one of the representatives is chosen as the arbiter, one message instead of two is incurred to forward CES to the arbiter). Once the arbiter receives CESs from the two representatives of the corresponding buddy regions, it merges them to form a new set of CES (to be detailed shortly).

Figure 7.5 illustrates one example for arbiter selection in the top right region from Figure 7.2. The owner of Zone M acts as the arbiter for the Region 1111 (consisting of the two buddy zones M and N) since Zone M is closer to the center of data space. Thus, the owner of Zone N forwards its CES to the owner of Zone M. Between the owners of Zone M and N, one peer is selected as the representative for Region 1111 to forward

the merged CES to the owner of Zone J, who is the arbiter for Region 111 (consisting of Zone M, N and J). Similarly, the owner of Zone L forwards its CES to the owner of Zone I, who is the arbiter for Region 110. Then, the CESs of Region 111 and 110 are merged at the owner of Zone I, who is the arbiter for Region 11.

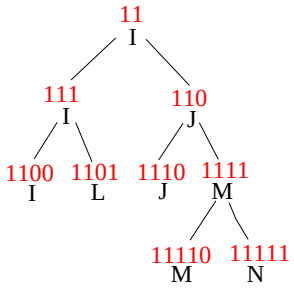


Fig. 7.5. Illustrative example for arbiter selection during cluster merging.

We now proceed to the details of how CESs are merged and corresponding local clusters are assembled at an arbiter. We assume that the two buddy regions are Region A and B and the corresponding CESs are Set A and B, respectively. The merged result is stored back to Set B. Recall that each entry in Set A and B indicates how a local cluster in Region A and B can be expanded. Without loss of generality, we start to examine Set A first to determine whether a cluster in Region A can be expanded to Region B. For each entry i in A (A_i), the arbiter checks whether there is some overlap between Ecoverage of A_i and Pcoverage of an entry j from B (B_j). An non-empty overlapping set between these two indicates that the local cluster (or noise objects) in

Region A corresponding to A_i can be combined with the local cluster (or noise objects) in Region B corresponding to B_j. In this case, these two clusters are merged as follows. Pcoverage of A_i is added to Pcoverage of B_j. The overlapping set is then removed from Ecoverage of A_i. In addition, the updated Ecoverage of A_i is added to Ecoverage of B_j. If Ecoverage of A_i becomes empty, the corresponding cluster can not be expanded any more. Thus, the examination for A_i terminates and A_i is removed from A. After all cluster expansion entries in A are examined, if A is not empty at this moment, the remaining entries in A are added to B.

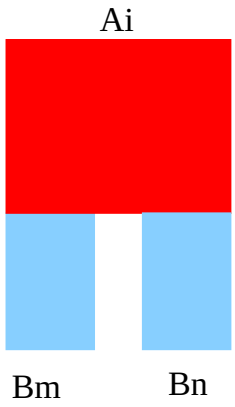


Fig. 7.6. Illustrative example for one cluster to be merged with two clusters.

Up to now, we have merged a cluster in Region A with a cluster in Region B. In some cases, it is possible that a cluster in Region A, e.g., A_i, can be merged with two or more different clusters in Region B, B_m, B_n as shown in Figure 7.7. Using the

procedure described above, A_i and B_m are merged into one larger cluster, and A_i and B_n are merged into another larger cluster. However, in this case, B_m, B_n and A_i should be merged (note that the symmetric case where a cluster in Region B can be merged with two or more clusters in Region A is already handled, since whenever a cluster in A is merged with B, the corresponding cluster expansion entry in B is expanded with the corresponding entry from A). To achieve this, we examine whether two clusters in set B have some overlap in their Pcoverage. If there are, merging is performed by combining their Pcoverage and Ecoverage. The algorithm for merging CESs is illustrated in Algorithm 10.

In order to determine which information to be forwarded along VPtree, the arbiter checks the merged CES to see whether there are some clusters that can not be expanded out of current region any more (i.e., empty Ecoverage). For those clusters, there is no need to forward the corresponding cluster expansion entry any further (this information is retained at this peer for cluster membership propagation to be discussed shortly). However, in order to obtain certain global clustering information, e.g., the total number of clusters, we need to indicate the existence of such clusters in the messages forwarded up to the root of VPtree. Therefore, we represent such a cluster by *CRegionID*, the RegionID of the smallest region enclosing the cluster, to indicate the existence and the coverage of this cluster (the coverage information is for cluster membership propagation to be discussed shortly). CRegionID replaces the cluster expansion entry for this cluster and is propagated up along the VPtree. Once the arbiter for the whole data space (the root of VPtree) receives messages from the corresponding representatives, in addition to

Algorithm 10 Algorithm for merging CESs at an arbiter.

Merging CESs (A and B are the two CESs to be merged. The merged results are stored in B.)

```
1: for i = 1 to |A| do
2:    for j = 1 to |B| do
3:       U = A_i.Ecoverage ∩ B_j.Pcoverage
4:       if U ≠ φ then
5:          B_j.Pcoverage = B_j.Pcoverage ∪ A_i.Pcoverage
6:          A_i.Ecoverage = A_i.Ecoverage − U
7:          B_j.Ecoverage = B_j.Ecoverage ∪ A_i.Ecoverage
8:          if A_i.Ecoverage = φ then
9:             A = A − {Ai}
10:         end if
11:      end if
12:   end for
13: end for
14: if A ≠ φ then
15:    B = B ∪ A
16: end if
17: i = 1
18: while i < |B| do
19:    for j = i + 1 to |B| do
20:       U = B_i.PCoverage ∩ B_j.Pcoverage
21:       if U ≠ φ then
22:          B_i.Pcoverage = B_i.Pcoverage ∪ B_j.Pcoverage
23:          B_i.Ecoverage = B_i.Ecoverage ∪ B_j.Ecoverage
24:          B = B − {Bj}
25:       else
26:          i = i + 1
27:       end if
28:   end for
29: end while
```

merging the two corresponding CESs, it also obtains the total number of clusters and assigns each cluster a unique numeric GlobalClusterID.

Note that even though the clusters are assembled along a hierarchy (VPtree), each node in the hierarchy only forwards one message and receives two messages. In addition, the message size is not necessarily monotonically increasing since the clusters that can not be expanded further are simply represented by their CRegionIDs. Thus, HCA does not impose high processing load at the root or nodes at the high level of the hierarchy. Potential single point of failure at the root or higher level of the hierarchy can be addressed by extending our current HCA algorithm to have multiple arbiters and representatives for each region, which is left for future exploration.

7.3.2 Cluster Membership Storage

Without a centralized server, the cluster membership storage is a non-trivial issue. While replicating the complete cluster membership at every peer can make the access to the cluster membership easy, this incurs excessive maintenance overheads as well as storage burden. Here, we propose a multi-granularity based cluster membership storage (MGCMS), which facilitates easy access to cluster membership with reasonable maintenance and storage overheads. In the following, we first explain the concept of MGCMS. We then explain how the cluster membership at different granularities is obtained.

In MGCMS, we distinguish the cluster membership at three granularities: *the global level*, *the zone level*, and *the object level*. At the global level, a peer maintains very coarse cluster membership, i.e., the total number of clusters. At the zone level, for each cluster that covers a peer's zone, this peer maintains the ZoneIDs of the other zones that

this cluster also covers if there are any. At the finest level, for each data object mapped to its zone, the peer records the detailed cluster membership (GlobalClusterID for data objects in a cluster or NULL for noise objects). MGCMS provides the peer a finer clustering picture for data objects mapped to its zone and a coarser clustering summary for data objects at other zones. Compared to full replication as described above, MGCMS incurs reasonable maintenance/storage overheads yet does not comprise the accessibility to the cluster membership. Table 7.1 summarizes the cluster membership maintained by a peer (i).

Table 7.1. Cluster membership maintained at Peer i in PENS

Storage Granularity	**Entry**
Global level	Number of clusters
Zone level	Every cluster covering i's zone: GlobalClusterID, {ZoneIDs}
Object level	Every data object at i's zone: GlobalClusterID or NULL

We now discuss how the cluster membership at different granularities is obtained. Recall that after performing HCA, the root of the VPtree has the following information: the total number of clusters, the coverage of each cluster (either the Pcoverage if a cluster spans the two buddy regions of the root node, or CRegionID otherwise). This aggregated clustering information is disseminated to relevant peers in accordance with MGCMS as follows. The cluster membership at the global level, i.e., the number of

clusters, is propagated to each peer along the VPtree. The cluster membership at the zone level is propagated along the VPtree down to relevant peers in accordance with the coverage of a cluster. The coverage of a cluster is represented by CRegionID before the peer who acts as the arbiter for the smallest region enclosing the cluster, i.e., the region with its RegionID as CRegionID, is reached. The coverage of a cluster is later represented by Pcoverage of this cluster once such peer is reached (since this peer retains Pcoverage for this cluster as mentioned in Section 7.3.1.3). The cluster membership at the zone level is then propagated down the VPtree as follows. If a cluster covers a region corresponding to a subtree branch, the cluster membership is propagated to the branch recursively till the owners of the zones covered by the cluster are reached. Zone owners then update their local clusters accordingly, i.e., recording the cluster membership at the zone level for each of the cluster covering its zone, labeling each of the data objects mapped to its zone with a GlobalClusterID if the data object belongs to a cluster or NULL otherwise.

7.3.3 Incremental Clustering

We propose increPENS to facilitate incremental clustering upon the insetion/deletion of a data object. increPENS first systematically identifies the clusters affected by an insertion/deletion, then updates the corresponding cluster membership accordingly. In the following, we discuss these two steps in details.

Assume that the data object to be inserted or deleted is p. As proven in [41], examining UpdSeed, the core data objects in the ε-neighborhood of these data objects within $N_{\varepsilon}(p)$ which have the core object properties changed as a result of the update, is

sufficient for incremental DBSCAN algorithm to identify the clusters and data objects affected by the insertion/deletion of a data object. This still holds in our case. In order to detect the data objects with core object properties changed, each data object is associated with the number of data objects in its ε-neighborhood (the incremental DBSCAN also makes similar optimization). This information is readily obtained during local clustering or cluster expansion check. Thus, upon insertion, only a data object p with $|N_{\varepsilon}(p)| = T - 1$ might change its core object property. On the other hand, upon deletion, only a data object p with $|N_{\varepsilon}(p)| = T$ might change its core object property.

We now proceed to the details of obtaining UpdSeed. The owner of the zone covering p, called as *update initiator*, first issues a region query to obtain the data objects in the ε-neighborhood of p. Then a peer sets the region that is the union of the ε-neighborhood of these data objects in $N_{\varepsilon}(p)$ with core object property changed as the query range and issues one single region query to obtain UpdSeed. Note that different from incremental DBSCAN algorithm where an individual region query is issued for each of the data object in $N_{\varepsilon}(p)$ with core object properties changed, increPENS avoids examining the same region covered by the ε-neighborhood of these data objects multiple times.

Similar to cluster expansion check, by examining data objects in UpdSeed, we have four possible cases upon insertion of a data object: noise, new cluster, cluster extension, and cluster merging, and four possible cases upon deletion of a data object: noise, cluster disappearance, cluster shrinking, and cluster splitting, where each of the

four cases for the deletion is contrary to the corresponding case for the insertion[6]. For the latter three cases, the update initiator informs the owners of the zones, which have their cluster membership at object level changed, to update the corresponding cluster membership accordingly. If the cluster membership at the zone level is changed (more or less zones are covered by the cluster), the update initiator informs the owners of the zones covered by this cluster to update their cluster membership at the zone level accordingly. Lastly and also less frequently, the number of clusters might change and the update initiator informs all peers to update their cluster membership at the global level accordingly. Note that for cluster membership update at the zone level (and at the object level in some cases), the update initiator needs to inform the owners for a specific subset of zones (covered by a cluster). In order to inform these peers, the coordinates of the corresponding zones need to be known for routing. While the update initiator only has the ZoneIDs for the zones covered by a cluster, it can figure out the coordinates of these zones easily (since a region is partitioned in the middle along a dimension chosen in round robin fashion). Once the coordinates of these zones are obtained, the update is propagated to these corresponding zones by following the neighbor pointers in a peer's routing table recursively.

7.3.4 Optimization Techniques

We observe that region queries are issued to obtain CES (cluster expansion information) in HCA and UpdSeed in increPEN, respectively. In HCA, the query region is the

[6]Determining which case that a data object belongs to according to UpdSeed is similar to [41]. Thus, it is not presented here for brevity. Interested readers please see [41].

ε-outer-boundary of a zone (Section 7.3.1.2). In increPENS, to obtain UpdSeed which are data objects in the ε-neighborhood of data objects that are in the ε-neighborhood of the data object to be inserted/deleted (Section 7.3.3), the query region is the 2ε-neighborhood of the data object to be inserted/deleted. One optimization is to let a peer store the index information for data objects mapped to the 2ε-outer-boundary of its zone (in addition to storing the index information for data objects mapped to its zone). In this case, a peer can obtain CES and UpdSeed through local computation only. We analyze in Section 7.5 that when 2ε is small and the data dimensionality is high, this optimization could result in large saving in message overheads with marginal extra maintenance and storage overheads.

7.4 Correctness of PENS

In this section, we prove that PENS can correctly discover clusters and noise in P2P systems. While we adopt the design principle of basic DBSCAN algorithm for local clustering and incremental DBSCAN algorithm for increPENS, respectively, peers correctly discover clusters and noise within their zones when performing local clustering according to [42], and increPENS correctly discovers clusters and noise incrementally according to [41]. Therefore, the focus of this section is on proving the correctness of HCA, i.e., the clusters obtained through HCA satisfy maximality and connectivity (as defined in Definition 4).

THEOREM 7.3. *Maximality: Let D be the set of data objects in the whole data space, and C be a cluster in D formed through HCA. $\forall p, q \in D$: if $q \in C$ and $p >_D q$, then also $p \in C$.*

Proof: We prove the maximality by induction. Given that clusters within level-(i+1) regions satisfy maximality within their regions, we need to prove that the clusters within level-i regions satisfy maximality within their regions.

When we merge two level-(i+1) buddy regions A and B, we have (and only have) following four cases when a cluster in a level-(i+1) region can be expanded (depicted in Figure 7.7(a–d), respectively):

1. A cluster (or noise object) in Region A, i.e., A_i, can be expanded to a cluster (or noise object) in Region B, i.e., B_j.

2. Multiple clusters (or noise objects) in Region A, i.e., A_{i1}, ..., A_{im}, can be expanded to a cluster (or noise object) in Region B, i.e., B_j.

3. A cluster (or noise object) in Region A, i.e., A_i, can be expanded to multiple clusters (or noise objects) in Region B, i.e., $B_{j1}, ..., B_{jk}$.

4. Multiple clusters (or noise objects) in Region A, i.e., $A_{i1}, ..., A_{im}$, can be expanded to multiple clusters (or noise objects) in Region B, i.e., $B_{j1}, ..., B_{jk}$.

If we prove that the clusters within the two level-(i+1) buddy regions are merged into one larger cluster within level-i region in all above four cases, the maximality of clusters within level-i region is proven to be satisfied given the maximality of clusters

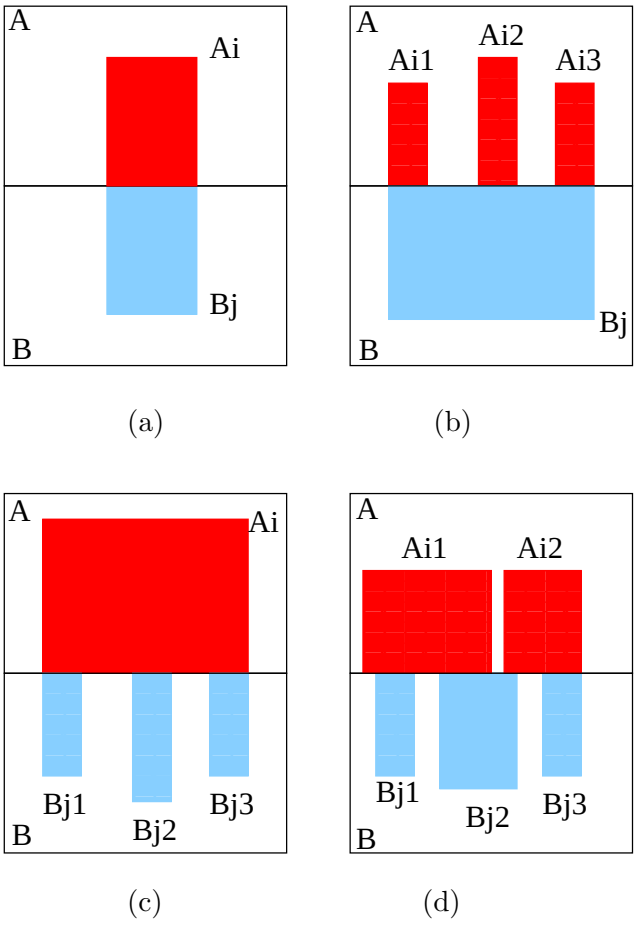


Fig. 7.7. Illustrative examples for cluster merging.

within level-(i+1) region is satisfied. Since Case 4 is a combination of Case 2 and 3, for presentation clarity, we do not prove maximality under case 4 explicitly.

Case 1: According to Line 5–7 in Algorithm 10, A_i and B_j are merged into one larger cluster.

Case 2: According to Line 5–7 in Algorithm 10, when A_{i1} is examined, A_{i1} and B_j are merged into one larger cluster, AB_{i1j}. Later on, when A_{i2} is examined, A_{i2} and AB_{i1j} are merged into one larger cluster, AB_{i12j}. Following this process, lastly, A_{im} and $AB_{i1(m-1)j}$ (consisting of clusters $A_{i1}, ..., A_{i(m-1)}$ and B_j) are merged into one larger cluster consisting of A_{i1}, ..., A_{im} and B_j.

Case 3: According to line 5–7 in Algorithm 10, A_i and B_{j1} are merged into one larger cluster, AB_{ij1}, A_i and B_{j2} are merged into one larger cluster, AB_{ij2}, ..., and A_i and B_{jk} are merged into one larger cluster, B_{ijk}. According to line 22–23 in Algorithm 10, AB_{ij1} and AB_{ij2} are merged into one larger cluster AB_{ij12}. Later on, AB_{ij12} and AB_{ij3} are merged into one larger cluster AB_{ij13}. Following this process, lastly, $AB_{ij1(k-1)}$ (consisting of A_i and B_{j1}, ..., $B_{j(k-1)}$) and AB_{ijk} (consisting of A_i and B_{jk}) are merged into one larger cluster consisting of A_i, B_{j1}, ..., B_{jk}. □

THEOREM 7.4. *Connectivity: Let D be the set of data objects in the whole data space, and C be a cluster in D formed through HCA. $\forall p, q \in C$: p is density-connected to q in D.*

Proof: We prove connectivity by induction. Given that clusters within level-(i+1) regions satisfy connectivity within their regions, we need to prove that the clusters within level-i regions satisfy connectivity within their regions.

Let C_i be a cluster in a level-i region R. Without loss of generality, we assume C_i is composed of two clusters C_{i1} and C_{i2}, which are clusters in the two level-(i+1) buddy regions $R1$ and $R2$, respectively (according to the transitivity of connectivity, the connectivity can be proven similarly for other cases when C_i consists of more than two clusters within level-($i+1$) regions). From Line 3–4 and Line 20–21 in Algorithm 10, C_{i1} and C_{i2} are merged only when there are some data objects x and y, where $x \in C_{i1}$ and $y \in C_{i2}$, are density-connected to each other in Region R. Since every data object in C_{i1} is density-connected to x in Region $R1$ and every data object in C_{i2} is density-connected to y in Region $R2$, every data object in C is density-connected to each other in R. □

7.5 Analysis of PENS

We analyze the message complexity of basic PENS algorithm and increPENS algorithm (for presentation clarity, we refer PENS without incremental clustering feature as basic PENS algorithm). Table 7.2 lists the symbols used in this section. For presentation clarity, we assume that the average side length of zones[7], r, is not less than 2ε, and the clusters are randomly distributed in the data space.

Recall that the basic PENS algorithm discovers clusters and noise through local clustering and HCA (hierarchical cluster assembling). While local clustering doesn't incur any message communication, cluster expansion check and cluster merging in HCA incur message communication. To conduct cluster expansion check, a peer issues a

[7]This is a practical assumption and we can guarantee that this condition is satisfied in most cases by requiring the threshold value M (introduced in Section 7.2.2), the minimum number of data objects contained in a zone to invoke zone splitting, is not less than $e \cdot (2\varepsilon)^k$.

Table 7.2. Symbols used in the analysis of PENS.

Symbols: Descriptions
N: Number of nodes in the network
k: Dimensionality of data objects
r: Average side length of a zone
e: Average data density
α: Average percentage of clusters affected by an insertion/deletion

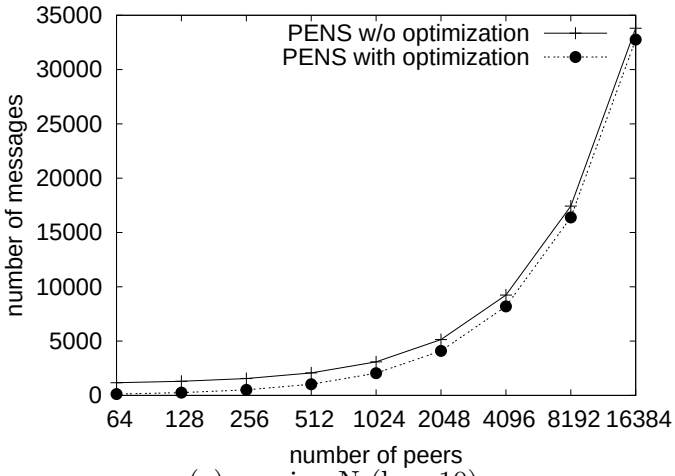
region query to obtain the data objects in the ε-outer-boundary of its zone. For a k-dimensional Cartesian space, the ε-outer-boundary of a zone intersects with the two neighboring zones (abutting all but one dimension) along each dimension, i.e., $2k$ zones in total, and other neighboring zones sharing each of the vertex, i.e., 2^k zones in total. Therefore, cluster expansion check incurs $(2k + 2^k)$ messages. During cluster merging, at most one message is forwarded along each edge in the VPtree. For a tree with N leaf nodes, there are $(N-1)$ internal nodes, thus, there are $(2N-2)$ edges in the tree connecting these $(2N - 1)$ nodes (N leaf nodes and $(N-1)$ internal nodes). Therefore, at most $(2N-2)$ messages are incurred during cluster merging. In summary, the message complexity incurred by the basic PENS algorithm is $(2k + 2^k + 2) \cdot N-2 = O(2^k \cdot N)$.

IncrePENS consists of obtaining UpdSeed and updating the corresponding cluster membership. The maximum region that need to be examined for obtaining UpdSeed is the 2ε-neighborhood of p as discussed in Section 7.3.4. The maximum number of zones that intersect with this region (excluding the zone A covering p) is $2k + 2^k$. This occurs when p is located around the center of Zone A. Moving p to other places within Zone A reduces the number of zones intersecting with the 2ε-neighborhood of p (since $r \geq 2\varepsilon$).

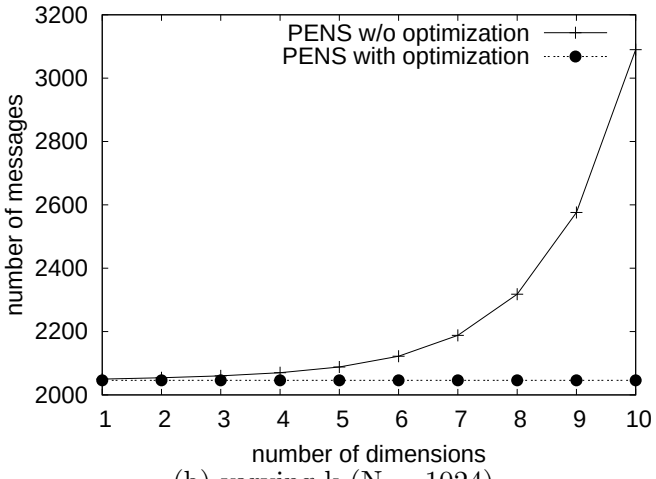
Therefore, the average message complexity is $2k + 2^k + \alpha \cdot N = O(2^k + \alpha \cdot N)$ (where $\alpha \cdot N$ indicates the number of peers managing the zones covering the affected clusters). In certain cases when the cluster membership at the global level, i.e., the number of clusters, is changed, this change needs to be propagated to the whole network, incurring $O(N)$ more messages. However, we expect this occurs fairly infrequently.

Using the optimization as mentioned in Section 7.3.4, no messages are required for cluster expansion check in basic PENS algorithm or obtaining UpdSeed in increPENS algorithm. Thus, the message complexity is reduced to $2N-2$ ($O(N)$) and $\alpha \cdot N$, respectively. Compared to the case without optimization, the relative maintenance and storage overhead is proportional to $\frac{e \cdot (2\varepsilon + r)^k}{e \cdot r^k}$, where r^k and $(2\varepsilon + r)^k$ denote the volume of the zone and the volume of the region consisting of this zone and its 2ε-outer-boundary, respectively. When 2ε is much smaller than r and k is large, the extra maintenance and storage overheads of this optimization technique are marginal.

Figure 7.8 displays the plots of message complexity incurred by clustering through basic PENS algorithm with and without optimization under different network size and data dimensionality. In Figure 7.8(a), the data dimensionality is set to 10 and the network size is varied from 64 to 16384 peers. From this figure, we can see that the number of messages grows linearly with the network size. The number of messages incurred by the basic PENS algorithm with optimization is smaller than the one without optimization. In Figure 7.8(b), the network size is set to 1024 and the data dimensionality is varied from 1 to 10. This figure indicates that when the dimensionality increases, the message complexity of the basic PENS algorithm with optimization is much smaller compared to the message complexity of the basic PENS algorithm without optimization.

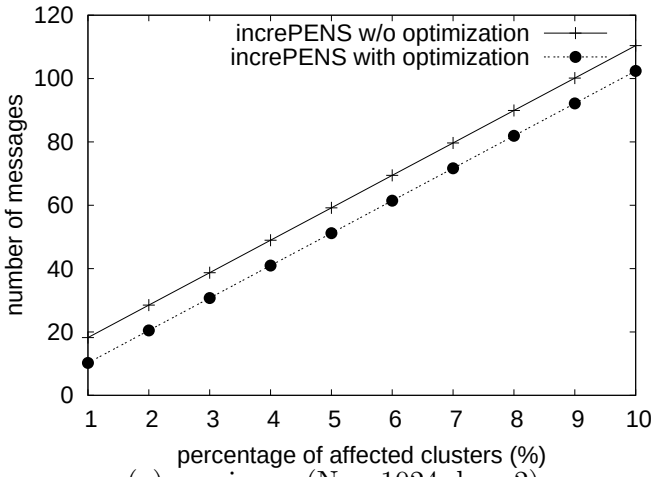


(a) varying N (k = 10)

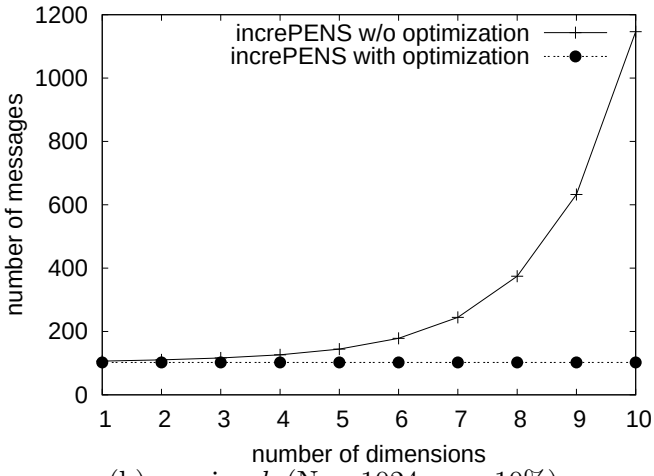


(b) varying k (N = 1024)

Fig. 7.8. Message complexity incurred by clustering through basic PENS algorithm.



(a) varying α (N = 1024, k = 2)



(b) varying k (N = 1024, $\alpha = 10\%$)

Fig. 7.9. Message complexity incurred by incremental clustering through increPENS algorithm.

Figure 7.9 displays the plots of message complexity incurred by incremental clustering through increPENS with and without optimization under different percentage of affected clusters and data dimensionality. In Figure 7.9(a), the data dimensionality is set to 2, the network size is set to 1024, and the percentage of affected clusters is varied from 1% to 10%. This figure indicates that the number of messages grows linearly with the percentage of affected clusters. In Figure 7.9(b), the percentage of the affected clusters is set to 10% and the data dimensionality is varied from 1 to 10. Similar to Figure 7.8(b), this figure indicates that with increasing dimensionality, the benefit of the optimization technique on incremental clustering becomes more significant.

In summary, the number of messages incurred by the basic PENS algorithm is proportional to the network size (N) and is not related to the dataset size (n), where N is expected to be much smaller than n. This indicates that our PENS is efficient. In fact, for any clustering algorithm in P2P systems, in order to form a global clustering model, any peer has to communicate with other peers at least once to exchange clustering information. Thus $O(N)$ message complexity incurred by the basic PENS algorithm with optimization is also the lower bound on messages complexity for any clustering algorithms in a P2P system with size N. For increPENS algorithm, the message complexity is related to the size (in terms of number of peers managing the zones) of the affected clusters, which confirms to the rational of incremental clustering.

7.6 Summary

We identify the challenges involved in performing density-based clustering on P2P systems and propose peer density-based clustering (PENS) algorithm, which overcomes

these challenges to facilitate clustering in P2P systems. PENS consists of three components, hierarchical cluster assembly (HCA), multi-granularity based cluster membership storage (MGCMS) and incremental PENS (increPENS). HCA enables peers to collaboratively form a global clustering model through peer communication only without requiring either a central site or message flooding. MGCMS stores cluster membership of multiple granularities at peers to facilitate easy access with reasonable maintenance and storage overhead. IncrePENS facilitates incremental clustering by systematically identifying and updating the clusters affected by the insertion/deletion of a data object. We also present optimization techniques applicable to PENS. We prove that the proposed algorithm can discover clusters and noise correctly. In addition, we provide complexity analysis on the messages incurred by PENS, which indicates that PENS can efficiently cluster data objects in peer-to-peer systems.

We are currently implementing the proposed algorithm and verify its efficiency through experiments. In addition, we plan to explore other clustering algorithms and data mining tasks in P2P systems.

Chapter 8

Conclusion

A massive amount of information is stored and shared in a large number of host nodes connected as large scale dynamic networks. Efficiently discovering and retrieving information and knowledge of interest from this decentralized gigantic information store is an essential operation for a wide spectrum of applications. The unique characteristics of these systems, such as large scale, lack of centralized administration, and high degree of dynamics, make this issue extremely challenging.

To address this challenge, we carefully examine various resource/knowledge discovery tasks in this dissertation. In the following, we summarize the major contributions made by this study. We then outline the future research directions, which all fall under the umbrella of the aforementioned challenge.

8.1 Summary of Contributions

This dissertation consists of five studies, which constitute the following major contributions.

- The studies on the design of efficient, robust and distributed information management infrastructures (SSW and DPTree) provide a solid foundation for exploring various data management tasks in the network systems.

- The proposed algorithms to process range queries and K nearest neighbor queries in P2P system paves the way to support other complex queries in LSDNs.

- The studies on the design of distributed mechanisms to efficiently identify frequent items and monitor changes on the data distribution in the networks are the first studies of their kind and provide profound insights on exploiting the vast amount of data for different applications, e.g., system performance tuning, network attack detection, market analysis.

- The study on clustering in P2P systems is also the first study of its kind and opens the new research direction on distributed data mining in LSDNs.

It is expected that this study will have a deep impact on the deployment of various applications that mandate efficient management and mining of the vast amount of data distributed in the network systems, including system performance tuning, network attack detection, smart query answering, scientific exploration, and market analysis.

8.2 Future Direction

In this study, we have addressed a couple of complex queries, i.e., range query and k nearest neighbor queries. There are some other complex queries that are worthy of further investigation. First, reverse nearest neighbor query, returning the set of data objects whose nearest neighbor is a given data object, is an important type of query appearing in location-based services as well as in some data mining tasks, such as outlier detection. Second, join among multiple data sets is another important query to be explored. In certain applications, the data sets might be distributed among the host

nodes in an attribute-based fashion (different attributes are distributed to different host nodes). For instance, a tuple consists of three attributes (ID, location, price), and the attribute pair (ID, location) and (ID, price) are two different data sets (A and B) stored in the network. A user might issue a query like "return the cheapest objects that are within 5 miles", which requires a join between data set A and B.

In the study of PENS, we take an initial step to investigate clustering, an important data mining tasks in P2P systems. Another important data mining task is association rule mining, which finds the set of data objects that frequently appear together (in a transaction). Mining association rules can reveal important user/system behavior in large scale dynamic networks, such as the sets of files downloaded together by users or the set of users interested in the same set of files, which are very useful in understanding/optimizing system performance.

In our two studies on network monitoring, we focus on monitoring/detecting interesting events. One interesting issue to explore in this area is to predict the future values/trends for data objects according to the patterns or periodicity existing among the data objects in the temporal and/or spatial domain. Integrating prediction and monitoring is expected to improve the performance of network monitoring in terms of both communication cost and latency.

Other network environments, such as mobile ad hoc networks, and wireless sensor networks, are facing the similar issues of resource/knowledge discovery. In addition, their extreme constraint on resources makes it even more challenging to address these issues. We have taken an preliminary step to tackle the issue of resource discovery in mobile ad hoc networks [83]. Nevertheless, more efforts are needed in this direction.

We have primarily targeted efficiency, scalability, and adaptivity on supporting resource/knowledge discovery in large scale dynamic networks. Other important system issues, such as consistency, availability, and security, incubate abundant research opportunities to be explored. In addition, sharing/accesing the information distributed in the networks raises many interesting political, social, and legal issues, which might have significant influence on the techniques to be designed.

In summary, efficiently managing the massive amount of information available in the networks is of critical need. The large scale of networks, extremely high volume of data, high degree of dynamics, constrains in the resources (mainly in bandwidth), and various political, social and privacy issues raise many challenges in resource/knowledge discovery in large scale dynamic networks for researchers to explore for years to come.

References

[1] Freenet website. http://www.freenet.com.

[2] Gnutella website. http://gnutella.wego.com.

[3] Morhpeus website. http://www.musiccity.com.

[4] Text retrieval conference web site. http://trec.nist.gov/.

[5] Top applications (bytes) for subinterface: Sd-nap traffic, 2002. www.caida.org/analysis/workload/byapplication/sdnap.

[6] True picture of p2p filesharing. http://www.cachelogic.com/home/pages/studies/2004_02.php.

[7] Pac nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *Proceedings of the 16th International Conference on Data Engineering (ICDE)*, pages 244–255, March 2000.

[8] Karl Aberer. P-Grid: a self-organizing access structure for P2P information systems. In *Proceedings of International Conference on Cooperative Information Systems (CoopIS)*, pages 179–194, September 2001.

[9] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman. Search in power-law networks. *Physics Review E*, 64:46135–46143, 2001.

[10] Ian Foster Adriana Iamnitchi, Matei Ripeanu. Small-world file-sharing communities. In *Proceedings of Infocom*, page to appear, March 2004.

[11] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of SIGMOD*, pages 94–105, June 1998.

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

[13] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of SIGMOD*, pages 49–60, June 1999.

[14] James Aspnes and Gauri Shah. Skip graphs. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384 – 393, January 2003.

[15] Brian Babcock and Chris Olston. Distributed top-k monitoring. In *Proceedings of ACM SIGMOD*, pages 28–39, June 2003.

[16] Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski, and Uwe Thaden. Progressive distributed top k retrieval in peer-to-peer networks. In *Proceedings of International Conference on Data Engineering (ICDE)*, pages 174–185, April 2005.

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

[18] S. Bandyopadhyay, C. Gianella, U. Maulik, H. Kargupta, K. Liu, and S. Datta. Clustering Distributed Data Streams in Peer-to-Peer Environments. *Information Science Journal (In Press)*, 2005.

[19] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. In *Proceedings of International Conference on World Wide Web (WWW)*, pages 651–660, May 2005.

[20] Mayank Bawa, Hector Garcia-Molina, Aristides Gionis, and Rajeev Motwani. Estimating aggregates on a peer-to-peer network. Technical report, Computer Science Department, Stanford University, April 2003.

[21] Mayank Bawa, Gurmeet S Manku, and Prabhakar Raghavan. SETS: Search enhanced by topic segmentation. In *Proceedings of ACM SIGIR conference on Research and Development in Information Retrieval*, pages 306–313, July 2003.

[22] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972.

[23] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[24] Stefan Berchtold, Christian Böhm, and Hans-Peter Kriegel. The pyramid-technique: Towards breaking the curse of dimensionality. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 142–153, 1998.

[25] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of International Conference on Very Large Databases(VLDB)*, pages 28–39, 1996.

[26] Michael W. Berry, Zlatko Drmac, and Elizabeth R. Jessup. Matrices, vector spaces, and information retrieval. *Society for Industrial and Applied Mathematics (SIAM) Review*, 41(2):335–362, 1999.

[27] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of ACM SIGCOMM*, pages 353–366, August 2004.

[28] Yatin Chawathe, Sriram Ramabhadran, Sylvia Ratnasamy, Anthony LaMarca, Scott Shenker, and Joseph Hellerstein. A case study in building layered DHT applications. In *Proceedings of ACM SIGCOMM*, pages 97–108, August 2005.

[29] Peter Cheeseman and John Stutz. Bayesian classification (autoclass): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180. AAAI/MIT Press, 1996.

[30] Graham Cormode and Minos Garofalakis. Sketching streams through the net: distributed approximate query tracking. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 13–24, August/September 2005.

[31] Graham Cormode, Minos Garofalakis, S. Muthukrishnan, and Rajeev Rastogi. Holistic aggregates in a networked world: distributed tracking of approximate quantiles. In *Proceedings of International Conference on Management of Data (SIGMOD)*, pages 25–36, June 2005.

[32] Graham Cormode and S. Muthukrishnan. What's new: Finding significant differences in network data streams. In *Proceedings INFOCOM*, April 2004.

[33] Graham Cormode, S. Muthukrishnan, and Wei Zhuang. What's different: Distributed, continuous monitoring of duplicate-resilient aggregates on data streams. In *Proceedings of International Conference on Data Engineering (ICDE)*, pages 20–31, April 2006.

[34] Adina Crainiceanu, Prakash Linga, Johannes Gehrke, and Jayavel Shanmugasundaram. Querying peer-to-peer networks using P-trees. In *Proceedings of International Workshop on the Web and Databases (WebDB)*, pages 25–30, June 2004.

[35] Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings Of IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 23–34, July 2002.

[36] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of ACM symposium on Operating Systems Principles (SOSP)*, pages 202–215, October 2001.

[37] Abhinandan Das, Sumit Ganguly, Minos N. Garofalakis, and Rajeev Rastogi. Distributed set expression cardinality estimation. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 312–323, August/September 2004.

[38] Inderjit S. Dhillon and Dharmendra S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Proceedings of Workshop on Large-Scale Parallel KDD Systems (in conjunction with SIGKDD)*, pages 245–260, August 1999.

[39] Richard Duda and Peter Hart. *Pattern classification and scene analysis.* John Wiley & Sons, New York, 1973.

[40] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *Proceedings of SIGCOMM*, pages 323–336, August 2002.

[41] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of VLDB*, pages 323–333, August 1998.

[42] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of Knowledge Discovery in Database (KDD)*, pages 226–231, 1996.

[43] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing iceberg queries efficiently. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 299–310, August 1998.

[44] Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[45] Philippe Flajolet and G. Nigel Martin. Probabilistic counting. In *Proceedings of Annual Symposium on Foundations of Computer Science (FOCS)*, pages 76–82, November 1983.

[46] George Forman and Bin Zhang. Distributed data clustering can be efficient and exact. *SIGKDD Explorations*, 2(2):34–38, 2000.

[47] Volker Gaede and Oliver Gűnther. Multidimensional access methods. *ACM Computer Surveys*, 30(2):170–231, 1998.

[48] Prasanna Ganesan, Mayank Bawa, and Hector Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 444–455, August 2004.

[49] Prasanna Ganesan, Beverly Yang, and Hector Garcia-Molina. One torus to rule them all: Multidimensional queries in P2P systems. In *Proceedings of International Workshop on the Web and Databases (WebDB)*, pages 19–24, June 2004.

[50] Jun Gao and Peter Steenkiste. An adaptive protocol for efficient support of range queries in DHT-based systems. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, pages 239–250, October 2004.

[51] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, pages 518–529, September 1999.

[52] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of SIGMOD*, pages 73–84, June 1998.

[53] Abhishek Gupta, Divyakant Agrawal, and Amr El Abbadi. Approximate range selection queries in peer-to-peer systems. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research(CIDR)*, January 2003.

[54] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of ACM SIGMOD*, pages 47–54, 1984.

[55] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. SkipNet: A scalable overlay network with practical locality properties. In *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2003.

[56] Michael E. Houle and Jun Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In *Proceeding of ICDE*, page to appear, 2005.

[57] Adriana Iamnitchi, Matei Ripeanu, and Ian T. Foster. Locating data in (small-world?) peer-to-peer scientific collaborations. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 232–241, March 2002.

[58] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In *Proceedings of VLDB*, pages 275–286, August 1998.

[59] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Quang Hieu Vu, and Rong Zhang. Speeding up search in peer-to-peer networks with a multi-way tree structure. In *Proceedings of ACM SIGMOD*, pages 1–12, June 2006.

[60] H. V. Jagadish, Beng Chin Ooi, Quang Hieu Vu, Zhang Rong, and Aoying Zhou. VBI-Tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In *Proceedings of Internatianal Conference on Data Engineering (ICDE)*, page to appear, April 2006.

[61] Navendu Jain, Praveen Yalagandula, Mike Dahlin, and Yin Zhang. Insight: a distributed monitoring system for tracking continuous queries. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, pages 1–7, october 2005.

[62] Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle. DBDC: Density based distributed clustering. In *Proceedings of International Conference on Extending Database Technology (EDBT)*, pages 88–105, March 2004.

[63] Hai Jin, Xiaomin Ning, and Hanhua Chen. Efficient search for peer-to-peer information retrieval using semantic small world. In *Poster Proceedings of International Conference on World Wide Web (WWW)*, pages 1003–1004, May 2006.

[64] Erik L. Johnson and Hillol Kargupta. Collective, hierarchical clustering from distributed, heterogeneous data. In *Proceedings of Workshop on Large-Scale Parallel KDD Systems (in conjunction with SIGKDD)*, pages 221–244, August 1999.

[65] David R. Karger and Matthias Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proceedings of International Conference on Peer-to-Peer Systems (IPTPS)*, pages 131–140, February 2004.

[66] Srinivas Kashyap, Supratim Deb, K.V.M. Naidu, Rajeev Rastogi, and Anand Srinivasan. Efficient gossip-based aggregate computation. In *Proceedings of ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, June 2006.

[67] Norio Katayama and Shin'ichi Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 369–380, 1997.

[68] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Proceedings of Symposium on Foundations of Computer Science (FOCS)*, pages 482–491, October 2003.

[69] Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *Proceedings of ACM SIGMOD*, pages 289–300, June 2006.

[70] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 180–191, September 2004.

[71] Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of USENIX Security Symposium*, pages 271–286, August 2004.

[72] Min Sik Kim, Taekhyun Kim, YongJune Shin, Simon S. Lam, and Edward J. Powers. A wavelet-based approach to detect shared congestion. In *Proceedings of SIGCOMM*, pages 293–306, August/September 2004.

[73] Jon Kleinberg. Navigation in a small world. *Nature*, 406(845), August 2000.

[74] Jon Kleinberg. The small-world phenomenon: an algorithm perspective. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing(SOTC)*, pages 163–170, May 2000.

[75] Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley Professional, 1998.

[76] John Kubiatowicz et al. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 190–201, Novermber 2000.

[77] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *Proceedings of SIGCOMM*, pages 217–228, August 2005.

[78] Chen Li, Edward Chang, Hector Garcia-Molina, and Gio Wiederhold. Clustering for approximate similarity search in high-dimensional spaces. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):792–808, 2002.

[79] Mei Li, Guanling Lee, Wang-Chien Lee, and Anand Sivasubramaniam. PENS: An algorithm for density-based clustering in peer-to-peer systems. In *Proceedings of the 1st International Conference on Scalable Information Systems (INFOSCALE 2006)*, May/June 2006.

[80] Mei Li and Wang-Chien Lee. netFilter: An in-network filtering technique to identify frequent items in P2P systems. *Under review.*

[81] Mei Li, Wang-Chien Lee, and Anand Sivasubramaniam. Supporting complex queries for high dimensional data in P2P systems. *Under review.*

[82] Mei Li, Wang-Chien Lee, and Anand Sivasubramaniam. Neighborhood signatures for searching P2P networks. In *Proceedings of the 7th International Database Engineering and Application Symposium (IDEAS 2003)*, pages 149–158, July 2003.

[83] Mei Li, Wang-Chien Lee, and Anand Sivasubramaniam. Efficient peer-to-peer information sharing over mobile ad hoc networks. In *Proceedings of the 2nd Workshop on Emerging Applications for Wireless and Mobile Access (MobEA 2004), in conjunction with the World Wide Web Conference (WWW)*, May 2004.

[84] Mei Li, Wang-Chien Lee, and Anand Sivasubramaniam. Semantic small world: An overlay network for peer-to-peer search. In *Proceedings of the 12th International Conference on Network Protocols (ICNP 2004)*, pages 228–238, October 2004.

[85] Mei Li, Wang-Chien Lee, and Anand Sivasubramaniam. DPTree: A balanced tree based index framework for peer-to-peer systems. In *Proceedings of the 14th International Conference on Network Protocols (ICNP 2006)*, pages 12–21, November 2006.

[86] Mei Li, Wang-Chien Lee, and Anand Sivasubramaniam. DPTree: Growing balanced trees in peer-to-peer systems. *Under review for IEEE Transactions on Parallel and Distributed Systems (TPDS)*, January 2007.

[87] Mei Li, Wang-Chien Lee, Anand Sivasubramaniam, and Dik Lun Lee. A small world overlay network for semantic based search in P2P systems. In *Proceedings of the 2nd Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGrid 2004), in conjunction with the World Wide Web Conference (WWW)*, May 2004.

[88] Mei Li, Wang-Chien Lee, Anand Sivasubramaniam, and Jing Zhao. Semantic small world: A small world based overlay for peer-to-peer search. *Under review for IEEE Transactions on Parallel and Distributed Systems (TPDS)*, September 2006.

[89] Mei Li, Ping Xia, and Wang-Chien Lee. Wavenet: A wavelet-based approach to monitor changes on the networked data. *Under review.*

[90] King-Ip Lin, H. V. Jagadish, and Christos Faloutsos. The TV-tree: An index structure for high-dimensional data. *International Journal on Very Large Databases*, 3(4):517–542, 1994.

[91] Bing Liu, Wang-Chien Lee, and Dik Lun Lee. Supporting complex multi-dimensional queries in P2P systems. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, pages 155–164, June 2005.

[92] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings Of the 16th ACM International Conference on Supercomputing*, pages 84–95, June 2002.

[93] John MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[94] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. In *Proceedings of Symposium on USENIX Operating System Design and Implementation (OSDI)*, December 2002.

[95] Amit Manjhi, Suman Nath, and Phillip B. Gibbons. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In *Proceedings of ACM SIGMOD*, pages 287–298, June 2005.

[96] Amit Manjhi, Vladislav Shkapenyuk, Kedar Dhamdhere, and Christopher Olston. Finding (recently) frequent items in distributed data streams. In *Proceedings of International Conference on Data Engineering (ICDE)*, pages 767–778, April 2005.

[97] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: Distributed hashing in a small world. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, March 2003.

[98] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. Wavelet-based histograms for selectivity estimation. In *Proceedings of SIGMOD*, pages 448–459, June 1998.

[99] Stanley Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.

[100] Wolfgang Müller and Andreas Henrich. Fast retrieval of high-dimensional feature vectors in P2P networks using compact peer data summaries. In *Proceedings of ACM SIGMM International Workshop on Multimedia Information Retrieval (MIR)*, pages 79–86, November 2003.

[101] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 250–262, November 2004.

[102] Wolfgang Nejdl, Martin Wolpers, Wolf Siberski, Christoph Schmitz, Mario T. Schlosser, Ingo Brunkhorst, and Alexander Löser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *Proceedings of the Twelfth International World Wide Web Conference (WWW)*, pages 536–543, May 2003.

[103] Cheuk Hang Ng, Ka Cheung Sia, and Chi Hang Chang. Advanced peer clustering and firework query model in the peer-to-peer network. In *Proceedings of the 12th World Wide Web Conference(WWW) (Poster)*, May 2003.

[104] Chris Olston, Jing Jiang, and Jennifer Widom. Adaptive filters for continuous queries over distributed data streams. In *Proceedings of ACM SIGMOD*, pages 563–574, June 2003.

[105] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load balancing in structured P2P systems. In *Proceedgins of International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 68–79, February 2003.

[106] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172, August 2001.

[107] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Application-level multicast using content-addressable networks. In *Networked Group Communication*, pages 14–29, 2001.

[108] Patrick Reynolds and Amin Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of International Middleware Conference*, pages 21–40, June 2003.

[109] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP)*, pages 188–201, October 2001.

[110] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings Of the 18th IFIP/ACM International Conference on Distributed Systems Platforms(Middleware)*, pages 329–350, November 2001.

[111] Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Proceedings of International Workshop on Networked Group Communication (NGC)*, pages 30–43, November 2001.

[112] Ozgur Sahin, Abhishek Gupta, Divyakant Agrawal, and Amr El Abbadi. A peer-to-peer framework for caching range queries. In *Proceedings of International Conference on Data Engineering (ICDE)*, page to appear, March 2004.

[113] Nagiza F. Samatova, George Ostrouchov, Al Geist, and Anatoli V. Melechko. RACHET: An efficient cover-based merging of clustering hierarchies from distributed datasets. *Distributed and Parallel Databases*, 11(2):157–180, 2002.

[114] Scott Schenker. Peer-to-peer computing: from exciting social revolution to boring academic research. Presentation at the Jon Postel Distinguished Lecture Series, UCLA, 2001.

[115] Robert Schweller, Ashish Gupta, Elliot Parsons, and Yan Chen. Reversible sketches for efficient and accurate change detection over network data streams. In *Proceedings of ACM SIGCOMM Conference on Internet Measurement (IMC)*, pages 207–212, October 2004.

[116] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. In *Proceedings of VLDB*, pages 428–439, August 1998.

[117] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, August 2001.

[118] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann, 1996.

[119] Jeremy Stribling, Isaac G. Councill, Jinyang Li, M. Frans Kaashoek, David R. Karger, Robert Morris, and Scott Shenker. Overcite: A cooperative digital research library.

[120] Sharmila Subramaniam, Themis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 187–198, September 2006.

[121] Chunqiang Tang, Melissa J. Buco, Rong N. Chang, Sandhya Dwarkadas, Laura Z. Luan, Edward So, and Christopher Ward. Low traffic overlay networks with large routing tables. In *Proceedings of ACM SIGMETRICS*, pages 14–25, June 2005.

[122] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of ACM SIGCOMM*, pages 175–186, August 2003.

[123] Egemen Tanin, Deepa Nayar, and Hanan Samet. An efficient nearest neighbor algorithm for P2P settings. In *Proceedings of National Conference on Digital Government Research*, pages 21–28, May 2005.

[124] Wei Wang, Jiong Yang, and Richard R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proceedings of VLDB*, pages 186–195, August 1997.

[125] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, June 1998.

[126] David A. White and Ramesh Jain. Similarity indexing with the SS-tree. In *Proceedings of IEEE International Conference on Data Engineering(ICDE)*, pages 516–523, 1996.

[127] Xiaowei Xu, Jochen Jäger, and Hans-Peter Kriegel. A fast parallel clustering algorithm for large spatial databases. *Data Mining and Knowledge Discovery*, 3(3):263–290, 1999.

[128] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings Of IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 5–14, July 2002.

[129] Haruo Yokota, Yasuhiko Kanemasa, and Jun Miyazaki. Fat-Btree: An update-conscious parallel directory structure. In *Proceedings of International Conference on Data Engineering (ICDE)*, pages 448–457, March 1999.

[130] Cui Yu, Beng Chin Ooi, Kian-Lee Tan, and H. V. Jagadish. Indexing the distance: An efficient method to KNN processing. In *The VLDB Journal*, pages 421–430, 2001.

[131] Matei Zaharia and Srinivasan Keshav. Gossip-based search selection in hybrid peer-to-peer networks. In *Proceedings of the Fifth International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2006.

[132] Chi Zhang, Arvind Krishnamurthy, and Randolph Y. Wang. Brushwood: Distributed trees in peer-to-peer systems. In *IPTPS*, Feburuary 2005.

[133] Hui Zhang, Ashish Goel, and Ramesh Govindan. Using the small-world model to improve Freenet performance. In *Proceedings of INFOCOM*, June 2002.

[134] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of SIGMOD*, pages 103–114, June 1996.

[135] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: an infrastructure for fault-tolerant wide-area location and routing. Technical Report UCS/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.

[136] Shelley Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 11–20, June 2001.

Vita

Mei Li received her B.S. in Biology from Sichuan University (China), her M.S. in Biochemistry from Chinese Academy of Sciences, and her M.S. in Computer Science from Pennsylvania State University. She enrolled in the Ph.D. program at the Department of Computer Science and Engineering at the Pennsylvania State University, Pennsylvania, USA in August 2002. She received her Doctor of Philosophy degree in Computer Science and Engineering from the Pennsylvania State University in May 2007. Mei Li's research interests are in the areas of distributed systems, network, database, data mining with special interest in information management and knowledge discovery in networks. Ms. Li is the inventor of a US patent (pending). She has received many awards, including a travel grant from National Science Foundation to attend Computing Research Association's Academic Careers Workshop in 2006, Excellent Thesis Award of Chinese Academy of Sciences in 1998, Dean's Scholarship of Chinese Academy of Sciences in 1998, Di'Ao scholarship of Chinese Academy of Sciences in 1996 and 1997, Excellent Student Scholarship from 1991 to 1995. She is currently a member of IEEE.