Techniques de localisation et de résumé des données dans les systèmes P2P

Rabab Hayek

École Centrale de Nantes **Université de Nantes** **École des Mines de Nantes**

ÉCOLE DOCTORALE STIM

« SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DES MATÉRIAUX »

Année 2009

Data Localization and Summarization Techniques in P2P Systems

THÈSE DE DOCTORAT

Discipline : Informatique

Spécialité : Bases de Données

Présentée
et soutenue publiquement par

Rabab HAYEK

*Le 9 Janvier 2009 l'*UFR *Sciences & Techniques, Université de Nantes,*
devant le jury ci-dessous

Président	:	Pr. directeur directeur	Université de
Rapporteurs	:	Witold Litwin, Professeur	Université de Dauphine
		Mohand-Said Hacid, Professeur	Université de Lyon
Examinateurs	:	Guillaume Raschia, Maître de Conférence	Université de Nantes
		Noureddine Mouaddib, Professeur	Université de Nantes
		Stephane Gancarski, Maître de Conférence HDR	INRIA Lip6
		Patrick Valduriez, Directeur de Recherche	INRIA Nantes

Directeur de thèse : Pr. Patrick Valduriez
Co-encadrant : Guillaume Raschia

Laboratoire: **L**ABORATOIRE D'**I**NFORMATIQUE DE **N**ANTES **A**TLANTIQUE.
CNRS FRE 2729. 2, rue de la Houssinière, BP 92 208 – 44 322 Nantes, CEDEX 3.

DATA LOCALIZATION AND SUMMARIZATION TECHNIQUES IN P2P SYSTEMS

Techniques de localisation et de résumé des données dans les systèmes P2P

Rabab HAYEK

⋈

favet neptunus eunti

Université de Nantes

Rabab HAYEK

Data Localization and Summarization Techniques in P2P Systems

V+**??**+IV p.

Ce document a t prpar avec LaTeX2e et la classe `these-LINA` version v. 2.7 de l'association de jeunes chercheurs en informatique LOGIN, Universit de Nantes. La classe `these-LINA` est disponible l'adresse : `http://login.irin.sciences.univ-nantes.fr/`.
Cette classe est conforme aux recommandations du ministre de l'ducation nationale, de l'enseignement suprieur et de la recherche (circulaire n° 05-094 du 29 mars 2005), de l'Universit de Nantes, de l'cole doctorale « Sciences et Technologies de l'Information et des Matriaux » (ED-STIM).

Impression : thesisRabab.tex – 08/01/2009 – 15:40.

Résumé

Le but de cette thèse est de contribuer au développement des techniques de localisation et de description de données dans des environnements P2P. Au niveau de la couche application, nous nous concentrons sur l'exploitatoin des sémantiques qui peuvent être capturées à partir des données partagées. Ces sémantiques peuvent améliorer l'efficacité de recherche, ainsi que permettre des requêtes complexes. A cet effet, nous présentons une technique originale d'indexation de données dans les systèmes P2P qui se base sur les résumés linguistiques. Nos résumés sont des vues synthétiques et multidimensionnelles qui supportent la localisation des données pertinentes en se basant sur leur contenu. Plus intéressant, ils fournissent des représentations intelligibles de données, qui peuvent renvoyer des réponses approximatives à des requêtes d'utilisateur.

Au niveau de la couche réseau P2P, nous nous concentrons sur l'exploitation des caractéristiques de la topologie, à savoir les caractéristiques de leur regroupement (*clustering*). Des informations sur le clustering du réseau P2P peuvent être utilisées pour réduire le trafic de réseau produit par le mécanisme de flooding. Ceci permet d'améliorer l'exécution des systèmes P2P, indépendamment de l'emploi des index de données à la couche application, puisque le mécanisme de flooding représente toujours un bloc constitutif fondamental des systèmes non structurés P2P.

Dans cette thèse, nous présentons un bref état de l'art sur les systèmes P2P de partage de données P2P et nous nous concentrons sur l'évolution des systèmes simples de partages des fichiers vers des systèmes de gestion des données. En second lieu, nous proposons une solution pour la gestion des résumés de données dans des systèmes P2P. Nous définissons un modèle approprié et des techniques efficaces pour la création et la mise à jour des résumés. Nous discutons également le traitement des requêtes dans le cadre des résumés. Troisièmement, nous proposons une technique de recherche basée sur clustering implémentée au dessus d'un protocole de custering selon la connectivité des noeuds. Nous nous concentrons sur la reduction des messages de requêtes redondants qui surchargent inutilement le système. Nous avons validé nos solutions par la simulation et les résultats montrent une bonne performance.

Mots-clés : Systèmes Pair à Pair, Résumés de données, Organisation du réseau

Abstract

The goal of this thesis is to contribute to the development of data localization and summarization techniques in P2P environments. At the application layer, we focus on exploiting the semantics that can be captured from the shared data. These semantics can improve the search efficiency, and allow for more query facilities. To this end, we introduce a novel data indexing technique into P2P systems that relies on linguistic summarization. Our summaries are synthetic, multidimensional views that support locating relevant data based on their content. More interestingly, they provide intelligible data representations which may return approximate answers for user queries.

At the P2P network layer, we focus on exploiting the characteristics of the overlay topology, namely its clustering features, in order to reduce the traffic overhead generated by flooding-based mechanisms. This allows to improve the performance of P2P systems, irrespective of the employment of techniques relying on data semantics at the application layer. To this end, we define a cluster-based search technique which is implemented over a connectivity-based clustering protocol. A connectivity-based clustering protocol aims to discover the natural organization of nodes, based on their connectivity. Thus, it delimits the boundaries of non-overlapping subgraphs (i.e. clusters) which are loosely connected, and in which nodes are highly connected.

In this thesis, we first survey P2P data sharing systems. We focus on the evolution from simple file-sharing systems with limited functionalities, to Peer Data Management Systems (PDMSs) that support advanced applications with more sophisticated data management techniques. Second, we propose a solution for managing linguistic summaries in P2P systems. We define an appropriate summary model and efficient techniques for summary creation and maintenance. We also discuss query processing in the context of summaries. Third, we propose a cluster-based search technique on top of existing connectivity-based clustering protocols. We focus on reducing redundant query messages which unnecessarily overload the system. We validated our solutions through simulation and the results show good performance.

Keywords: P2P systems, Database summarization, Network Clustering

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In the last decade, the rise of P2P networks is attested by the increasing amount of interest in both commercial and academic areas. According to studies and statistics made on web traffic, P2P networks are becoming the killer application of the internet. P2P network architectures are generally characterized by the direct sharing of computer resources rather than requiring the intermediating of a centralized server. Initially, the P2P paradigm gained much popularity with systems and infrastructures designed for sharing digital media data (e.g. music and video files), such as Gnutella [4], and bittorrent [1].

At the same time, the database community has been slowly evolving toward a higher degree of distribution. The traditional architectures employed by distributed and parallel databases, and data integration systems rely on a centralized global schema and strong assumptions about the network. While these architectures have reached their maturity and only supported a limited number of users, the new class of P2P architectures starts to represent an interesting alternative to build large-scale distributed systems.

P2P systems allow to share data on a world-wide scale with many advantages such as decentralization, self-organization, autonomy, etc. However, their operation as distributed systems is constrained by the employment of efficient data management techniques. In distributed databases, the location of content is generally known, the query optimizations are performed under a central coordination, and answers to queries are expected to be complete. On the other side, the ad hoc and dynamic membership of participants in P2P systems makes it difficult to predict about the location and the quality of resources.

Initially, search in P2P systems relied on flooding mechanism. A peer sends a query message to its neighbors, which in turn forward the message to all their neighbors (except the sender) and so on, until a stop condition is satisfied (e.g. number of query hops, number of returned results). The main merits of this approach are its simplicity, reliability, and its high network coverage, i.e. a large number of nodes could be reached within a small number of hops. However, flooding suffers from high bandwidth consumption since it may produce a significant number of query messages. In fact, there are two major concerns with flooding-based mechanisms.

- **Search Blindness:** a peer forwards a query message to its neighbors without any information on how these neighbors may contribute to query answers. This results in a poor query recall, i.e. a large number of visited nodes do not store relevant data.
- **Message Redundancy:** a peer may receive the same query message multiple times. This is due to the ad hoc nature of P2P connections, i.e. the neighbor selection process is random and non-discriminant in P2P topologies. Due to the lack of structural information, a query message travels all the available paths from a given node, and thus may reach a same node through different paths.

In the P2P literature, many techniques have been proposed in order to improve the performance of P2P systems. Initial works have led to structured systems, which mainly act as global distributed indexes (e.g. [72], [131]). These systems address the problem of topology randomness by imposing a specific network structure, and remedy to the blindness problem by making a tight control on data (or data pointers) placement. Hence, these systems provide an efficient, deterministic search. However, they compromise node autonomy and may restrict search expressiveness since data are mainly accessed using numerical identifiers. Other data indexing techniques have been proposed in fully unstructured systems in order to support query routing with information about the location or the direction toward relevant data (e.g. [9], [34]). Each of these techniques achieves a different trade-off between the cost of maintaining indexes and the benefits obtained in query processing.

Another research axis has focused on network clustering which aims to introduce some structure to, or extract inherent structural patterns from fully unstructured P2P networks. A network clustering scheme consists in organizing the nodes into clusters, based on a given criterion. A clustering criterion could be a physical network metric (e.g. bandwidth, latency), some peer property/behavior (e.g. node connectivity/stability), or even defined at the application layer (e.g. similar interests). The motivation behind P2P clustering schemes is that efficient routing protocols could be defined by taking advantage of the clustering features of the underlying network [98], [49]. However, the existing works have focused on proposing clustering techniques that cope with the dynamic and autonomous nature of P2P systems. Less effort has been put on demonstrating the efficiency of cluster-based search techniques, i.e. *how restructuring unstructured P2P systems may contribute to enhance the search performance.*

As revealed by the large number of P2P research papers and surveys, data localization (or access) has been the main issue addressed in the P2P community. However, advanced data management techniques, which allow more than locating data, are strongly required to support database applications. These applications are facing the *information explosion* problem: a huge amount of information is generated and stored each day into data sources (e.g. biological, astronomical database applications). This ever increasing amount of available data, in addition to the high distribution of connected resources, makes search techniques no more sufficient for supporting P2P database applications. To illustrate, in a scientific collaborative application, a doctor may require information about patients

diagnosed with some disease, without being interested in individual patient records. Besides, in today's decision-support applications, users may prefer an approximate but fast answer, instead of waiting a long time for an exact one.

To address the problem of managing voluminous databases, the data summarization paradigm has emerged into the database field. The objective of data summarization is to synthesize the information which is disseminated within large datasets, in order to provide the *essential*. Obviously, data summarization is of higher importance in P2P systems where the participants share a global database, which is equivalent to the aggregation of all their local databases, and thus may become much more voluminous. *Reasoning on compact data descriptions that can return approximate answers like "dead Malaria patients are typically children and old" to queries like "age of dead Malaria patients", is much more efficient than retrieving raw records, which may be very costly to access in highly distributed P2P databases.*

1.2 Contribution

The objective of this thesis is to contribute to the development of both data localization and summarization techniques in P2P systems. Our main contributions are the following.

First, we survey P2P sharing systems [62]. All along, we focus on the evolution from simple file-sharing systems, with limited functionalities, to Peer Data Management Systems (PDMS) that support advanced applications with more sophisticated data management techniques. Advanced P2P applications are dealing with semantically rich data (e.g. XML documents, relational tables), using a high-level SQL-like query language. We start our survey with an overview over the existing P2P network architectures, and the associated routing protocols. Then, we discuss data indexing techniques based on their distribution degree and the semantics they can capture from the underlying data. We also discuss schema management techniques which allow integrating heterogeneous data. We conclude by discussing the techniques proposed for processing complex queries (e.g. range and join queries). Complex query facilities are necessary for advanced applications which require a high level of search expressiveness. This last part shows the lack of querying techniques that allow for an *approximate query answering*.

At the application layer, our second contribution consists in exploiting data semantics not only to improve the efficiency of exchanging data, but also to allow exchanging synthesized information represented in a higher level of abstraction. This is very interesting for collaborative and decision-support applications.

To this end, we introduce a data summarization technique into Peer Data Management Systems. Our approach for data summarization is based on SaintEtiQ: an online linguistic approach for database summarization [51], [94], [126]. The SaintEtiQ model has been designed by our team for managing voluminous databases, and has been extensively studied in centralized environments. It exhibits many salient features such as robustness against the potential imprecision and vagueness in raw data, and scalability in term of the amount of processed data. The produced summaries are synthetic, multidimensional

views over relational tables. The novelty of our proposal relies on the double exploitation of these summaries in distributed P2P systems. First, as semantic indexes, they support locating relevant nodes based on their data descriptions. Second, due to their intelligibility, these summaries can be directly queried and thus approximately answer a query without the need for exploring original data, which might be highly distributed.

Our work has evolved as follows. First, we attempt to study the feasibility of integrating our summarization technique into an existing PDMS architecture. So, we work in the context of APPA (Atlas Peer-to-Peer Architecture), a PDMS which has been developed by our team over the last years [93], [120]. The main objective of APPA is to provide high level services for advanced applications. To deal with semantically rich data, APPA supports decentralized schema management and uses novel solutions for complex query processing. It addresses the problem of data consistency and reliability through efficient solutions for persistent data management with updates, and data replication with semantic-based reconciliation.

In our work, we propose PeerSum, a new service for managing data summaries [58], [63]. First, we define a summary model that deals with the dynamic and autonomous nature of P2P systems. This model architecture is characterized by an incremental mechanism of summary construction. Each peer maintains a local summary LS of its own database. To this end, the summary process is integrated to each peer's DataBase Management System (DBMS). Then, peers which are willing to cooperate will exchange and merge summaries, in order to build a Global Summary GS over their shared data. Let summary "*coverage*" be the fraction of peers that own data described by that summary. According to our model, a global summary GS is characterized by a continuous evolution in term of coverage, i.e. the cooperation between two sets of peers, each having constructed a global summary, will result in a higher-coverage one. Second, we define efficient techniques for global summary creation and maintenance, based on APPA's services. In particular, we assume that a common storage for global summaries is provided by APPA.

The second part has been done in hierarchical P2P networks [60]. Here, we study how summaries can be efficiently distributed in P2P systems, which has been abstracted when relying on APPA's services. The idea is to exploit the node heterogeneity in hierarchical (superpeer) networks. Hence, the architecture of our summary model is characterized by 2-summary levels. The first level is provided by the set of local summaries maintained at different peers. The second is obtained by materializing a set of global summaries as follows. The network is organized into *domains*, where a domain is defined as being the set of a supernode and its associated leaf nodes. In a given domain, peers cooperate to maintain a global summary over their shared data, which is stored at the corresponding superpeer. The issue of organizing the network into domains, i.e. how the superpeers are selected and other peers are grouped around them in a fully decentralized manner, is discussed [59], [61]. Then, we present efficient algorithms for managing summaries in a given domain.

Finally, we propose a query processing mechanism which relies on the interrogation of available summaries. Our performance is evaluated through a cost model and a simulation

model. The simulation results have shown that our solutions allow to significantly reduce the query cost, without incurring high costs of summary updating.

At the P2P network layer, our third contribution consists in exploiting the characteristics of the overlay topology, namely its clustering features, in order to reduce the traffic overhead. This allows to improve the performance of P2P systems, irrespective of the employment of techniques relying on data semantics at the application layer. In fact, the efficiency of such techniques depends on the application and the nature of exchanged data. Besides, they generally have direct implications on the underlying network, such in data indexing schemes, or additional requirements for maintaining a new overlay built on top of the P2P network, as in semantic clustering schemes. A semantic clustering may propose the creation of new overlays in which semantically related peers are connected to each others. In other terms, such a clustering strives to introduce structure, which is different from discovering inherent structure of the P2P overlay, as we are considering in our work. On the other hand, in the context where such techniques are not employed, or their efficiency degrades (depending on the application), the solution is to resort to flooding-based techniques. In reality, the flooding approach is still a fundamental building block of unstructured P2P systems. It represents the natural way for exchanging messages between nodes which are connected in an ad hoc fashion.

In our work, we propose a search technique, which is implemented over a connectivity-based clustering protocol, in order to reduce the number of query messages generated by flooding-based algorithms. A connectivity-based clustering protocol aims to discover the natural organization of nodes, based on their connectivity. Thus, it delimits the boundaries of sub-graphs (i.e. clusters) which are loosely connected, and in which nodes are highly connected. In the P2P literature, two main protocols have been proposed, i.e. the Connectivity-based Distributed node Clustering (CDC) [121], and the SCM-based Distributed Clustering (SDC) [86]. These works have been introduced by arguing the benefits of such a clustering from the search performance point of view. However, none of them has proposed an appropriate routing protocol, or provided analytical models or experimental results on how their clustering schemes contribute to reduce the bandwidth consumption. The main focus was on the clustering scheme accuracy, i.e. using the Scale Coverage Measure (SCM), and its maintenance against node dynamicity.

Our Cluster-Based Search Technique (CBST) works as follows. Based on a local knowledge about the network clustering in its neighborhood, each node maintains information about *local links* to nodes in its cluster (intra-cluster information), as well as about *global links* connecting its cluster to other reachable clusters (inter-cluster information). The intra-cluster routing information is equivalent to a spanning tree, rooted at that node and covering its partners (i.e. the nodes that belong to its cluster). These information are efficiently gathered and maintained in a cluster-based routing table.

The benefits in query routing are two folds. First, a query Q is efficiently disseminated in a given cluster, using the spanning tree of the first node contacted in that cluster. Second, the query messages between clusters are restricted to those traversing the global links specified by the querying node. Extensive simulations have demonstrated the

efficiency of the *CBST* technique compared to pure flooding and random walk routing techniques.

1.3 Roadmap

The rest of this thesis is organized as follows. In Chapter 2, we present an overview of P2P data sharing systems. In Chapter 3, we present our solutions for summary management in P2P systems. Chapter 4 proposes CBST, a cluster-based search technique for unstructured P2P systems. Chapter 5 concludes and discusses future directions of research.

Chapter 2

Data Sharing in P2P Systems

The recent years have witnessed a paradigm shift in the design of internet-scale distributed systems, with widespread proliferation of peer-to-peer technologies. Nowadays, the P2P model is used for diverse applications and services–including content storage and sharing (file-sharing, content distribution, backup storage) and communication (voice, instant messages, multicast) to name a few.

But, what is the P2P paradigm?

From the application perspective, the P2P paradigm is a way to leverage vast amounts of computing power, storage, and connectivity from personal computers distributed around the world [46]. Thus, the P2P model allows distributed systems to scale up on a world wide scale without the need for an expensive infrastructure, like the one it would be incurred by a client-server model.

From the system perspective, the P2P paradigm is about managing autonomous, unreliable resources that connect to the system in order to provide together the desired objectives, which that system is supposed to achieve. The management of such resources should be done without any global information or central control.

In other words, the P2P model overcomes the limitations of centralized and client-server models by introducing symmetry in roles, where each node is both a client and a server. But unlike Grid systems, P2P networks do not arise from the collaboration between established and connected groups of systems. Instead, they are characterized by ad hoc connections between autonomous and dynamic resources. Thus, P2P systems pose new challenges including resource discovery, reliability and availability.

In late 1999, P2P systems gained much attention with Napster's support for music sharing, and then have became a very interesting medium through which users share huge amount of data. Popular examples of P2P data sharing systems (e.g. Gnutella, KaZaa) report millions of users, sharing petabytes of data. However, a key challenge is implementing efficient techniques for search and data retrieval, without which an enormous shared data collection remains useless.

In a P2P data sharing system, users should be able to locate relevant data in a resource-

efficient manner. Let us examine the generic architecture of a given peer, as shown by Figure 2.1. Queries are submitted through a user interface. Then, they are handled by a data management layer which includes several techniques for supporting an efficient distributed query processing. This data management layer has been enriched all along the evolution of P2P systems from simple file-sharing systems with limited functionalities, to Peer Data Management Systems (PDMSs) which are dealing with semantically rich data.

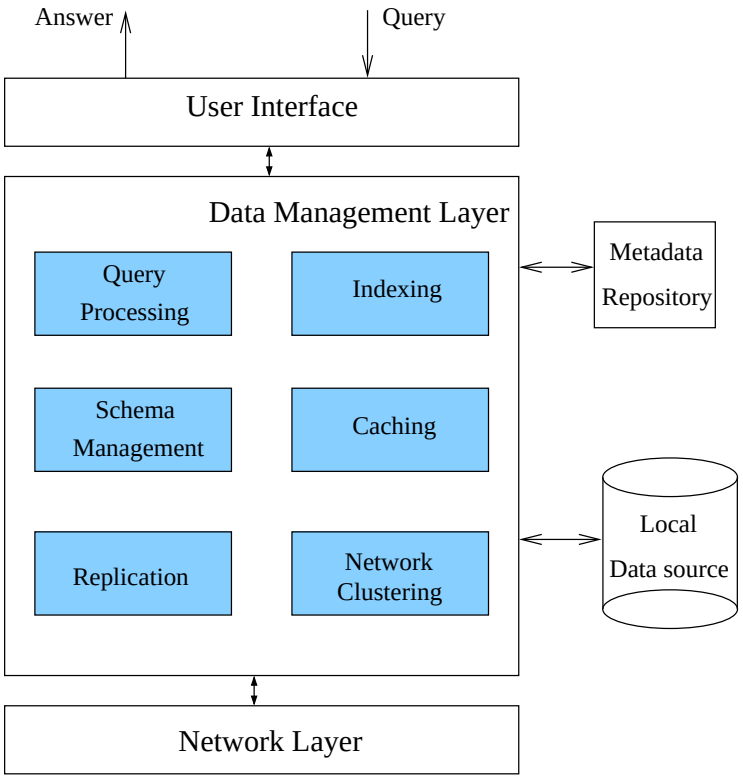

Figure2.1: Peer Generic Architecture

In early P2P file sharing systems, the data management layer lacks several components. At a given peer, filename-based queries are blindly broadcasted in the network in order to locate the requested files. Besides, all the visited peers locally evaluate the received queries and return results, if any, to be finally merged at the requesting peer. The performance of these systems is quite dependent of the topology of the *underlying network*, and the associated routing protocol. This is discussed in Section 2.1.

To enhance the search performance, P2P works started to employ *data indexing* techniques (e.g. [9], [131]). At a given peer, the index allows to select a set of relevant peers to which the query is directly sent (i.e. location indexes), or to determine the direction through which relevant peers may be located (i.e. forwarding indexes). Section 2.2 discusses the P2P indexing schemes.

Parallel works have focused on data *replication* and *caching* techniques in order to improve the availability and the consistency of data, against the dynamic and autonomous

nature of P2P systems. In this work, we do not detail these techniques, however good pointers can be found in [21, 74, 136].

In Schema-based P2P systems [149], each peer can provide its own database with its own schema, and may issue queries according to its local schema. In this case, irrespective of using data indexes, peers have to apply *schema management* techniques that provide a common ground for distributed query processing (e.g. [73], [137]). The basic idea is to identify content or structure similarities among peers. Semantic mappings are then defined to specify these similarities, and based on the semantic mapping definitions, queries are reformulated for each specific peer. Semantic mappings and other metadata are stored in a specific repository. The schema management techniques are presented in Section 2.3.

Network clustering has been also proposed as a viable solution to improve query processing in P2P systems (e.g. [86], [10]). Clustering techniques aim to organize the network into groups based on some criteria. A clustering criterion may be a physical network parameter (e.g. bandwidth), peer property/behavior (e.g. connectivity, stability), or application-dependent parameter (e.g. similarity of interests). Clustering in P2P networks will be addressed later in Chapter 4.

Finally, we note that the data management layer presented in Figure 2.1 may include additional components depending on other application requirements, such as trust and security.

All the above techniques have a common objective which is improving the efficiency of locating data. However, they should not restrict the search expressiveness. Certainly, the required level of search expressiveness is related to the data model used by the application. For instance, advanced P2P applications which are dealing with semantically rich data requires a higher expressiveness than key-based lookups or keyword searches. Processing complex queries in P2P systems is discussed in Section 2.4.

Note that throughout this thesis, the terms "node" and "peer" are used interchangeably to refer to the entities that are connected in a peer-to-peer network.

2.1 P2P Networks

P2P systems are application-level virtual networks with their own overlay topology and routing protocols. The overlay topology defines how the nodes are connected to each others, while routing protocols define how nodes can exchange messages in order to share information and resources. The network topology and the associated routing protocol have significant influence on application properties such as performance, scalability, and reliability. P2P network overlays can be classified into two main categories: *unstructured* and *structured*, based on their structure. By "*structure*" we refer to the control on overlay creation and data placement.

2.1.1 Unstructured

Most popular P2P applications operate on unstructured networks. In these networks, peers connect in an ad-hoc fashion and the placement of content is completely unrelated to the overlay topology. Although P2P systems are supposed to operate in a fully decentralized manner (i.e. fully decentralized routing mechanisms), in practice, unstructured networks with various degrees of centralization are encountered. Accordingly, three categories can be identified.

2.1.1.1 Hybrid Decentralized Architectures

In these networks, a central server facilitates the interaction between nodes by indexing all their shared files (Figure 2.2). Whenever a query is submitted, the central server is addressed to identify the nodes storing the requested files. Then, the file exchange may take place directly between two nodes. Certainly, this approach provides a very good search efficiency. However, the central server, which is a single point of failure, renders hybrid decentralized networks inherently unscalable and vulnerable to malicious attacks.

The class of P2P systems relying on such hybrid architectures, i.e. including a server (e.g. red node) and peers (e.g. blue nodes), is usually called the first generation of P2P systems ($1GP$). A well-known example is Napster [6].

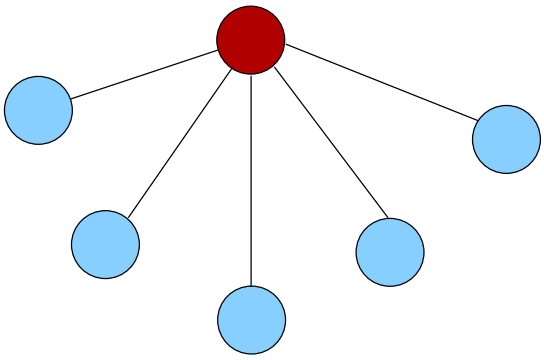

Figure2.2: Hybrid decentralized architecture

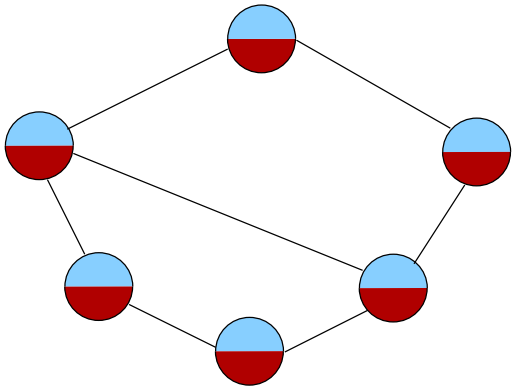

Figure2.3: Pure decentralized architecture

2.1.1.2 Pure Decentralized Architectures

In pure decentralized networks, there is a complete symmetry in node roles without any central coordination. Each node is both a client a server, i.e. each node may issue requests and serve/forward requests of other nodes (bi-colored nodes in 2.3). Hence, they exhibit high fault tolerance against node dynamicity and failure. However, resources are maintained locally and nodes have only limited knowledge. Thus, guarantees on lookup efficiency and content availability can not be provided. Here, search mechanisms range from brute flooding to more sophisticated mechanisms, such as random walks [136] and routing indices [9]. These mechanisms have direct implications on network scalability.

Representative examples of pure decentralized P2P systems are Gnutella [4], and FreeHaven [124].

2.1.1.3 Partially Decentralized Architectures

In these networks, there is a differentiation in roles between *supernode* and *leafnode*. Each supernode acts as a proxy for all its neighboring leaves by indexing their data and forwarding queries on their behalf. In practice, several supernodes are designated in the system to avoid all the problems associated to a single server (Figure 2.4). Like pure decentralized P2P networks, the set of supernodes can be organized in a P2P fashion and communicate with one another in sophisticated ways. They are dynamically assigned and, if they fail, the network will automatically take action to replace them with others.

Examples of partially decentralized P2P systems are KaZaa [5], Gnutella2 [3], and Edutella [137]. Note that partially decentralized networks are also referred as *hierarchical* networks, while pure decentralized ones are referred as *flat* networks. Both categories represent the so-called, second P2P generation ($2GP$).

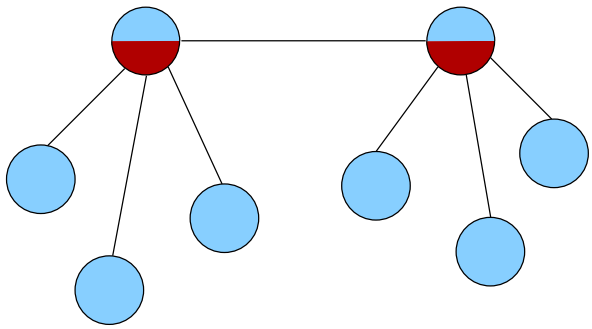

Figure2.4: Partially decentralized architecture

2.1.2 Structured

In an attempt to remedy the scalability problem of unstructured systems, some works have focused on introducing "*structure*" into network topologies. The topology overlay is tightly controlled, and the content may be distributed according to specific rules. These works led to the third generation of P2P systems ($3GP$), i.e. structured systems. Aiming basically to act as a decentralized index, structured overlays provide a mapping between content (e.g. file identifier) and location (e.g. node address), in the form of a distributed routing table.

Structured networks consist in partitioning a key space among peers, so that each peer is responsible for a specific key space partition, i.e. it should store all the resources (or pointers) which are mapped into keys, which are in the respective key-space partition. Then a routing algorithm is defined to allow a deterministic search based on key content. A representative class of structured overlays are the *Distributed Hash Tables DHT*s (e.g.

Structure	Decentralization		
	Hybrid	*Partial*	*Full*
Unstructured	Napster Publius	KaZaa Morpheus Gnutella2 Edutella	Gnutella FreeHaven
Structured Infrastructures			Chord CAN Trapestry Pastry
Structured Systems			OceanStore Mnemosyne Scan, Past Kademlia

Table2.1: A classification of P2P Systems and Infrastructures Based on Network Structure, and Degree of Decentralization [19]

[72], [131]). Freenet [70] is often qualified as a loosely structured system because the nodes of its P2P network can produce an estimate (not with certainty) of which node is most likely to store certain data. They use a chain mode propagation approach, where each node makes a local decision about which node to send the request message next.

Table 2.1 summarizes the P2P categories we outlined, with examples of P2P systems and infrastructures. P2P infrastructures do not constitute working applications, but provide P2P based-services (e.g. location and routing, anonymity, reputation management) and application frameworks. The infrastructures listed here are location and routing infrastructures. Note that according to the centralization criteria, all structured systems and infrastructures rely on pure decentralized topologies where all participants have equal roles.

2.1.3 Unstructured vs. Structured: Competition or Complementarity?

An important question is: should the P2P overlay be "Structured" or "Unstructured"? Are the two approaches competing or complementary?

Some have considered unstructured and structured routing algorithms as competing alternatives. When generic key lookups are required, structured routing schemes guarantee locating relevant nodes within a bounded number of hops, based on strong theoretical foundations. The routing unstructured approaches, however, may have large costs or fail to find available data (in particular unpopular data). Despite of the lookup efficiency of structured overlays, several research groups are still leveraging unstructured P2P schemes. In fact, there are two main criticisms for structured systems [155]. First, the strict network structure imposes high overhead for handling node join and leave, although some works

have defended performance during churn (e.g. [37]). Second, the lookup efficiency of these systems is limited to exact-match queries. Their ability to implement keyword searches and more complex queries is still an open issue. Therefore, given a P2P application, the best suited network overlay depends on that application functionalities and performance metrics.

Recently, some have started to justify that unstructured and structured approaches are complementary, not competing. The approach presented in [97] improves the unstructured Gnutella network by adding structural components. The motivation behind is that unstructured routing mechanisms are inefficient for data that are not highly replicated across the P2P network, while structured key-based lookup performs efficiently, irrespective of replication. In [29], the authors leverage the idea of cohabiting several P2P overlays on a same network, so that the best overlay could be chosen depending on the application. The distinctive feature of this proposal is that, in the *joint overlay*, the cohabiting overlays share information to reduce their maintenance cost while keeping the same level of performance.

Finally, we agree with the statement saying that the "unstructured vs. structured" taxonomy is becoming less useful, for two reasons. First, almost no network topologies are truly "unstructured". Unstructured P2P proposals, which used initially blind flooding and random walks, have evolved to exploit inherent structure (e.g. small world and scale-free features), or to incorporate structure through clusters and superpeers. Second, a new class of *schema-based P2P* systems, also called Peer Data Management systems PDMSs, has emerged [149]. Examples of such systems combine approaches from P2P research as well as from the database and semantic web research areas. These systems allow the aggregation and integration of data from autonomous, distributed data sources. They are dealing with heterogeneity of nodes and structure within data.

Following this statement, some studies have adopted database taxonomy rather than networking taxonomy (e.g. [27], [64]) in order to categorize P2P search networks. The structure is implicitly determined by the type of the employed index. In the following section, we discuss the different data indexing schemes that have been proposed in the P2P literature.

2.2 Data Indexing in P2P Systems

P2P search techniques rely basically on data indexes. A data indexing scheme should take into account the following requirements. First, the creation/maintenance of indexes should not overload either the nodes by an extensive usage of their resources, or the network by a large bandwidth consumption. Second, the mechanism of maintaining indexes should not restrict peer autonomy. Instead, it should recover from node leave and join in a resource-efficient manner.

Obviously, the use of indexes should contribute to enhance the efficiency of searches made in the system. For instance, this efficiency can be quantified by the rate of successful searches (a search is *successful* if it locates, at least, one replica of the requested object),

the response time, the number of returned results, the number of hops made to find a first query matching, and the number of messages exchanged in the network, which is an important metric from the system point of view.

The related trade-offs between index update cost, efficiency of the associated search technique, and peer churn are critical to evaluate a P2P indexing scheme.

2.2.1 Index Types

A P2P index can be local, centralized or distributed according to where it is maintained in the system, and to the distribution of data which it refers to.

2.2.1.1 Local Index

A node only keeps references to its own data, without obtaining any information about data stored at other nodes. The very early Gnutella design [4] adopted the *local-index* approach. This approach enables rich queries, but also generates huge traffic overhead since the query needs to be flooded widely in the network. Furthermore, any guarantees on search success can not be provided.

Considering that the key part of P2P searching approaches is an efficient routing mechanism, the local-index approaches can be seen as *index-free*, since they do not support query routing with any *forwarding* or *location* hints [154]. A forwarding index allows to reach the requested object within a varying number of hops (with the network size), while a location index allows to reach the target in a single hop. Based on the same reasoning, the search techniques that have been proposed to improve the performance of index-free systems, are referred as *blind* search techniques [140].

Breadth First Search (BFS). The originally Gnutella algorithm uses flooding (BFS traversal of the underlying graph) for object discovery, and contacts all accessible nodes within a Time-To-Leave (TTL) value (Figure 2.5). Small TTL values reduce the network traffic and the load at peers, but also reduce the chances of a successful search.

Modified BFS [143] is a variation of the BFS scheme in which the peers randomly choose only a ratio of their neighbors to forward the query to (Figure 2.6). This approach reduces the number of messages needed for query routing at the cost of loosing available query answers, which might be found by the original BFS.

Iterative Deepening. In [136], the idea of iterative deepening has been borrowed from artificial intelligence and used in P2P searching. This method is also called *expanding ring*. The querying node periodically issues a sequence of BFS with increasing TTL values. The query terminates when sufficient number of results is found, or the predefined maximum value of TTL is reached. Iterative deepening is tailored to applications where the initial number of results found at peers that are closer to the query originator is important. In this case, it achieves good performance gains compared to the original BFS. In other cases, its overhead and response time may be much higher.

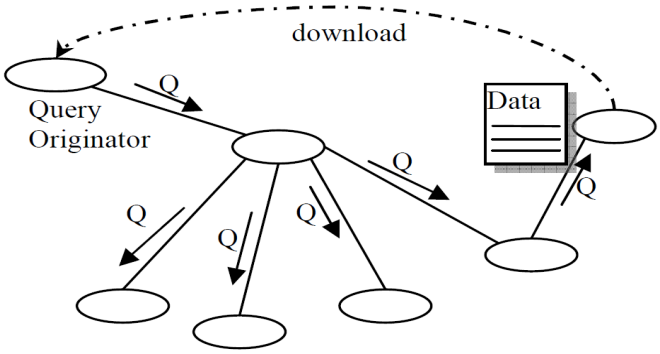

Figure2.5: Example of BFS: the received query is forwarded to all the neighbors

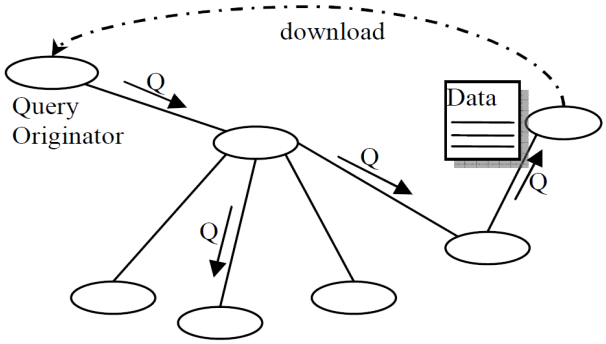

Figure2.6: Example of Modified BFS: the received query is forwarded to a randomly selected set of neighbors

Random Walks. In the *standard random walk* algorithm, the querying node forwards the query message to one randomly chosen neighbor. This neighbor randomly selects one of its neighbors and forwards the query to that neighbor, and so on until there is a query match. This algorithm indeed reduces the network traffic, but massively increases the search latency.

In the *k-walker random walk* algorithm [136], the query is replicated at the originator, so it sends k query messages to an equal number of randomly chosen neighbors. Each of these messages follows its own path, having intermediate nodes forward it to a randomly chosen neighbor at each step. These query messages are also known as *walkers*. When the TTL of a walker reaches zero, it is discarded.

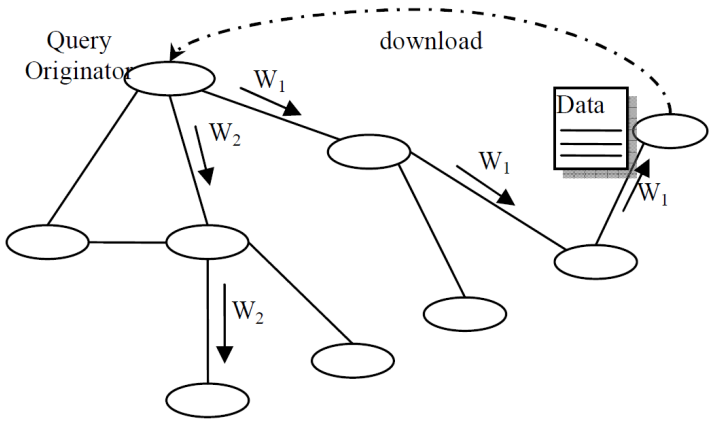

Figure2.7: Example of Random Walks: each received walk is forwarded to only one neighbor

The algorithm's most important advantage is the significant message reduction it achieves. It produces $k*TTL$ messages in the worst case, a number which seldom depends on the underlying network. It also achieves some kind of local "load balancing", since no nodes are favored in the forwarding process over others. However, the most serious problem of this algorithm is its highly variable performance. Success rates and number of

hits vary greatly depending on network topology and the random choices made. Another drawback is its inability to adapt to different query loads.

Adamic et al [81] addressed the first problem of random walks by recommending that instead of using purely random walks, the search protocol should bias its walks toward high-degree nodes (i.e. nodes with large number of connections). They assume that high-degree nodes are also capable of higher query throughputs. Certainly, the relevance of such assumption is constrained by the design of balancing rules to avoid overloading high-degree nodes, which may not have the capacity to handle a large number of queries.

Finally we note that, in spite of their name, the *Local Indices* proposed in [34] do not belong to this type of indexes. An index is locally maintained at a given node, however, it refers to remote data stored at other nodes.

2.2.1.2 Centralized Index

The index is centralized at dedicated servers, but the described data is distributed. In fact, the centralized schemes [6] were the first to demonstrate the P2P scalability that comes from separating data index from the data itself. The centralized index is a location (non-forwarding) index that allows to locate relevant data within one hop, which is very efficient. However, the central servers are single points of failure which renders the system inherently unscalable and vulnerable to malicious attack.

The P2P research community has rapidly turned its back on centralized architectures. Furthermore, P2P systems that only use local indexes are becoming rare, since routing the query in a blind manner is still providing a poor trade-off between the traffic overhead and the lookup efficiency. In practice, all current P2P systems are implementing distributed indexes.

2.2.1.3 Distributed Index

The index refers to data from distributed sources, and is itself distributed across the network. Here, we are talking about the global index, which is (may be virtually) obtained from the set of indexes materialized in the network. A hybrid decentralized approach consists in distributing such global index among some specialized nodes (e.g. supernodes and ultrapeers). A pure decentralized approach distributes the index among all participants, that is, each node in the system maintains a part of that index.

An early P2P proposal for a distributed index was Freenet [70]. Freenet uses a hash function to generate keys, by which the shared files are identified. Each node maintains a dynamic routing table containing the addresses of other nodes and the file keys they are thought to hold. To search for a file, the user sends a request message specifying the *key* and a TTL value. Upon receiving a query message, a node checks its local table for either a match or another node with keys close to the target. If the file is eventually found at a certain node (before exceeding TTL), the query response traverses the successful query path in reverse, adding a new routing table entry (the requested key and the file

provider) at each peer. A subsequent request with the same key will be served with this cached entry. The request will be forwarded directly to the node that had previously provided the data. Freenet allows to significantly reduce the traffic overhead in the system. However, it only supports exact-match queries, and only one result is returned. Another limitation is that Freenet takes time to build an efficient index upon the arrival of a new node.

As said before, almost all of the current P2P proposals rely on distributed indexes, which can range from simple forwarding hints to exact object locations. These indexes can be distinguished according to whether they are semantic-free, or they capture data semantics. The semantic index is human-readable. For example, it might associate information with keywords, document names, or database keys. A free-semantic index typically corresponds to the index by a hash mechanism, i.e. the DHT schemes.

2.2.2 Semantic-free Index: DHT

Structured systems have emerged mainly in an attempt to address that scalability problem of Gnutella-like systems. They use the Distributed Hash Table (DHT) as a substrate, in which the overlay topology and the data placement are tightly controlled.

Various DHT schemes differ in the topologies, routing protocols, fault tolerance, and resilience to churn. In the following, we first briefly discuss the origin of DHTs. Then, we present the main geometries (i.e. the topology and the associated routing strategies) used for DHT-based systems, and discuss their search efficiency and robustness.

2.2.2.1 Origins of DHTs

Distributed Hash Tables (DHTs) have been proposed to provide semantic-free, data-centric references. DHTs allow to find an object's persistent key in a very large, changing set of hosts. Three contributions from the 1990s are at the origin of DHTs: Plaxton Tree [111], Consistent Hashing [77], and Scalable Distributed Data Structure (SDDS) [87]. The Plaxton Tree [111] influenced the design of many P2P structured systems and infrastructures, such as Pastry [20], Tapestry [36], and OceanStore [75]. The main value of Plaxton Tree is that it can locate objects using fixed-length routing tables. Objects and nodes are assigned semantic-free addresses, and each node is effectively the root of a spanning tree corresponding to a given object. A message routes toward an object by matching longer address suffixes. This approach has several limitations, including that an object's root node is a single point of failure. This issue has been addressed while designing the Tapestry infrastructure.

Consistent Hashing [77] has been first introduced in the context of distributing objects across a network of caches. Unlike normal hashing, consistent hashing provides a smooth object references redistribution, when caches are added or deleted. It also ensures that the total number of caches responsible for a particular object is limited. However, there is an open Consistent Hashing problem concerning the fraction of items that should be

redistributed when a node is inserted.

A Scalable Distributed Data Structure (SDDS) [87] is specifically designed for multicomputers. An SDDS file can span over the storage of many computers linked through a high-speed network. An SDDS satisfies three design requirements: files grow to new servers only when existing servers are well loaded, there is no centralized directory, and the basic operations like insert, search and split never require atomic updates to multiple clients. Although the SDDS has not been significantly explored in structured P2P designs, the authors have continuously worked on elaborating it to be adapted for P2P environments [89], [139].

2.2.2.2 Tree

Tree is the first geometry which is used for organizing the peers of a DHT and routing queries among them. In this approach, nodes and objects are assigned unique identifiers (e.g. 160-bit key). The leaf nodes of the binary tree represent the key-space partitions (peer's identifiers). The depth of that tree is $log(n)$, where n is the number of peers. The responsible for a given object key is the peer whose identifier has the highest number of prefix bits which are common with the key. A search is routed toward the requested object based on longest prefix matching at each intermediate peer until reaching the responsible peer. The distance between two peers is then the height of the smallest common subtree. Tapestry [36] uses similar prefix matching in order to forward query messages. To avoid the problem of single point of failure that root nodes constitute in the Plaxton Tree model, Tapestry assigns multiple roots to each object. Such approach allows reliability at the cost of redundancy.

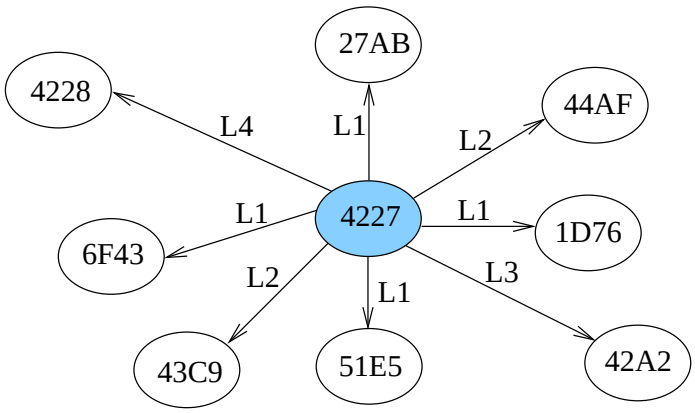

Figure2.8: Tapestry routing mesh from the perspective of a single node. Outgoing neighbor links point to nodes with a common matching prefix. Higher level entries match more digits. Together, these links form the neighbor map [36].

For each level in a tree topology there are several choices to select routing table entries. To illustrate, each Tapestry node maintains a neighbor map as shown in Figure 2.8. The neighbor map has multiple levels, each level l containing pointers to nodes whose identifier

must be matched with l bits. For instance, the node in figure 2.8 maintains at the third level of its routing table one pointer to one node matching his identifier with 3 digits.

The tree geometry has good neighbor selection flexibility, i.e. each peer has $2^i - 1$ options in choosing a neighbor at a level i. However, it has no flexibility for message routing: there is only one neighbor which the message must be forwarded to, i.e. this is the neighbor that has the most common prefix bits with the given key. Several applications have been designed on the top of Tapestry, such as OceanStore [75]. Pastry [20] is a scheme similar to Tapestry, however, it differs in the approach to achieving network locality and object replication. It is employed by the PAST large-scale persistent P2P storage utility [21].

2.2.2.3 Ring

The Ring geometry is based on a one dimensional cyclic space such that the peers are ordered on the circle clockwise with respect to their keys. Chord [72] is the prototypical DHT ring. Chord supports one main operation: find a peer with the given *key*. The keys are assigned both to data and peers by means of a variant of Consistent Hashing [77]. Each key on the key-space is mapped to the peer with the least identifier greater or equal to the key, and this peer is called the key's *successor*. Thus to say, this peer is responsible for the corresponding data. The use of consistent hashing tends to balance load, as each node receives roughly the same number of keys.

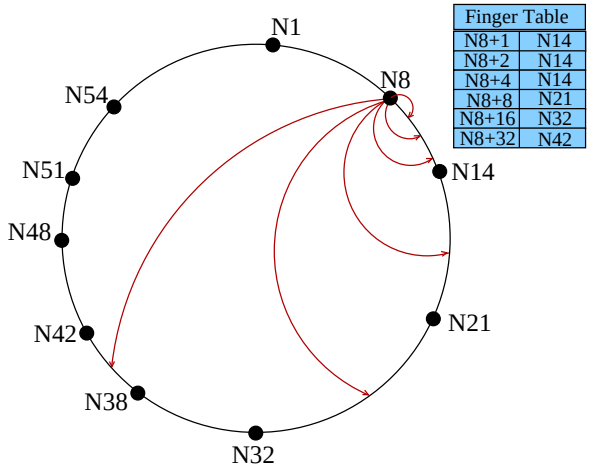

Figure2.9: The *finger table* at node 8 on a Chord ring of 10 nodes, $m = 6$ [72].

In Chord, a peer needs to track the addresses of only m other peers, not all peers such in the original Consistent Hashing proposal. Each peer p maintains a *"finger table"* containing $m = log(n)$ entries such that the i^{th} entry provides the address of the peer whose distance from p clockwise in the circle is $2^i - 1$ *mod* n (see Figure 2.9). Hence, any peer can route a given key to its responsible in $logn$ hops because each hop reduces the distance to the destination by half. In Chord, a peer needs to track the addresses of

only $m = O(logn)$ other peers, not all peers such as in the original Consistent Hashing proposal.

The correctness of the Chord routing protocol relies on the fact that each peer is aware of its successors. When peers fail, it is possible that a peer does not know its new successor, and that it has no chance to learn about it. To avoid this situation, peers maintain a successor list of size r, which contains the peer's first r successors. When the successor peer does not respond, the peer simply contacts the next peer on its list.

2.2.2.4 Hypercube

The Hypercube geometry is based on partitioning a d-dimensional space into a set of separate zones and attributing each zone to one peer. Peers have unique identifiers with $log\ n$ bits, where n is the total number of peers Each peer p has $log\ n$ neighbors such that the identifier of the ith neighbor and p differ only in the ith bit. Thus, there is only one different bit between the identifier of p and each of its neighbors. The distance between two peers is the number of bits on which their identifiers differ. Query routing proceeds by greedily forwarding the given key via intermediate peers to the peer that has minimum bit difference with the key. Thus, it is somehow similar to routing on the tree. The difference is that the hypercube allows bit differences to be reduced in any order while with the tree bit differences have to be reduced in strictly left-to-right order.

The number of options for selecting a route between two peers with k bit differences is $(log\ n) * (log\ n - 1) * \cdots * (log\ n - k)$, i.e. the first peer on the route has $log\ n$ choices, and each next peer on the route has one choice less than its predecessor. Thus, in the hypercube, there is great flexibility for route selection. However, each node in the coordinate space does not have any choice over its neighbors coordinates since adjacent coordinate zones in the coordinate space can not change. The high selection flexibility offered by the Hypercube is at the price of poor neighbor selection flexibility.

The routing geometry used in CAN [131] resembles a hypercube geometry. CAN uses a d-dimensional coordinate space which is partitioned into n zones and each zone is occupied by one peer (see Figure 2.10). When $d = log\ n$, the neighbor sets in CAN are similar to those of a $log\ n$ dimensional hypercube.

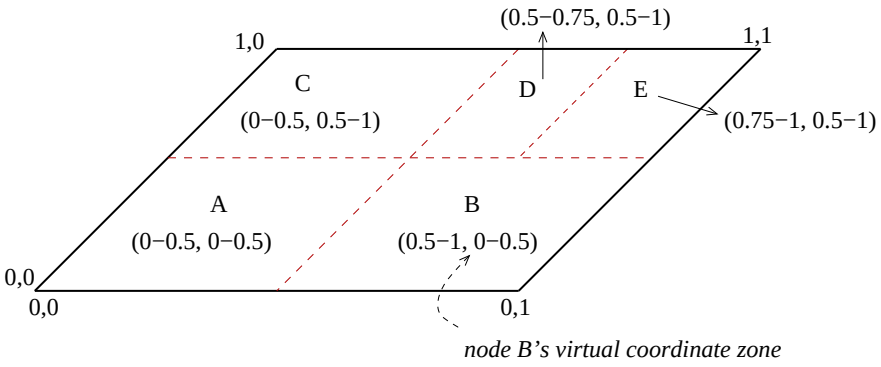

Figure2.10: 2-dimensional $[0; 1]\times[0; 1]$ coordinate space partitioned between 5 CAN nodes [131].

Other DHTs geometries are the *Butterfly* geometry which is used in Viceroy [45], and the XOR geometry which is used by Kademlia [112]. Certainly, two or more geometries can be combined together to provide a hybrid geometry that satisfies better the DHT requirements. To illustrate, Pastry [20] combines the tree and ring geometries in order to achieve more efficiency and flexibility.

2.2.3 Semantic Index

The initial unstructured file sharing P2P systems offered a filename-based search facility, while the DHT-based systems offered only a key-based lookup. However, as stated before, the P2P systems should be able to do more than "finding" things, i.e. to capture data semantics and to allow for rich, complex queries. Works on both P2P networks, unstructured and structured, have been started in order to support P2P applications with higher levels of search expressiveness. First enhancements to existing file sharing P2P systems have early provided keyword search facilities. Later, providing large-scale Information Retrieval (IR), e.g. for searching the world wide web, becomes an appealing application for P2P networks. Consequently, the well known IR techniques have been brought into the context of P2P networks, in order to support a decentralized document management (e.g. storing, clustering, indexing) and retrieval.

Recently, the Database and P2P paradigm have meet. The former was slowly moving toward a higher degree of distribution, and thus requiring a new class of scalable, distributed architecture. The latter has started to explore more expressiveness infrastructures in order to extend the representation and query functionalities it can offer to advanced applications. The P2P Data Management Systems (PDMS) are the point where the two paradigms meet.

As the P2P networks are going to be adaptable, i.e. to support a wide range of applications, they need to accommodate many search types. Index engineering has been always at the heart of P2P search methods. In the following, we introduce the various types of semantic indexes employed by current P2P systems. Then, the query capabilities will be discussed in Section 2.4.

2.2.3.1 Keyword Lookup

Gnutella [4] provides a simple keyword match. Queries contain a string of keywords and peers answer when they have files whose names contain all that keywords. In its first version, Gnutella was a local-index system. Queries were flooded in the entire network and peers only used their local indexes for filename matches.

As a way to improve the performance of unstructured Gnutella-like systems, the notion of *ultrapeer* was introduced, so that the peer are organized into a hierarchical network overlay. In [23], each peer maintains an index of filename keywords, called the Query Routing Table (QRT), and forwards it to its ultrapeer. Upon receiving a query, the latter sends the query only to leaves which have a match based on their QRTs. Later, there has been a proposal to exploit the network hierarchy in order to build a hierarchical index.

Aggregated QRTs are distributed amongst the ultrapeers to improve the query forwarding from an ultrapeer to another.

In other approach, [34] suggested the *local indices*: data structures where each node maintains an index of the data stored at nodes located within a radius r from itself. The query routing is done in a BFS-like way, except that the query is processed only at the peers that are at certain hop distances from the query originator. To minimize the overhead, the hop distance between two consecutive peers that process the query must be $2 * r + 1$. In other words, the query must be processed at peers whose distance from the query originator is $m * (2 * r + 1)$ for $m = 1, 2, \ldots$. This allows querying all data without any overlap. The processing time of this approach is less than that of standard BFS because only a certain number of peers process the query. However, the number of routing messages is comparable to that of standard BFS. In addition, whenever a peer joins/leaves the network or updates its shared data, a flooding with $TTL = r$ is needed in order to update the peers' indices, so the overhead becomes very significant for highly dynamic environments.

Routing Indices [9] have been proposed to support query routing with information about "direction" towards data, rather than providing its actual location. Documents are assumed to fall into a number of topics, and queries request documents on particular topics. Routing Indices (RIs) store information about the approximate number of documents from every topic that can be retrieved through each outgoing link (i.e. not only from that neighbor but from all nodes accessible from it).

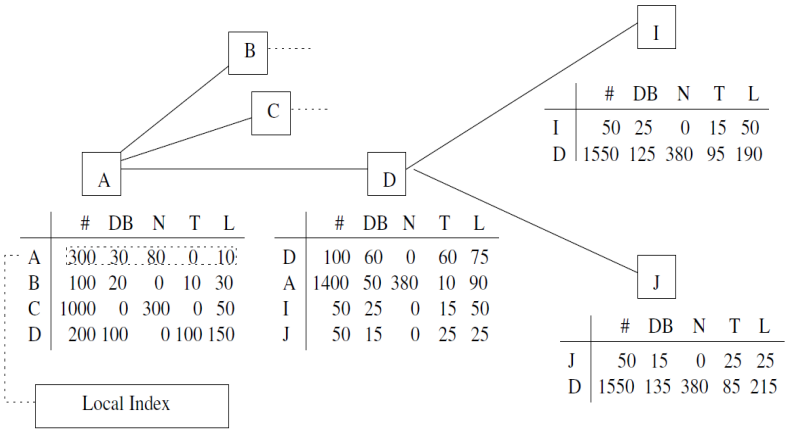

Figure2.11: Example of Routing Indices

Figure 2.11 shows an example of a P2P network with RIs built over four topics of interest. The first row of each RI contains the summary of the local index presented before (i.e. radius $r = 2$). In particular, the summary of A's local index shows that A has 300 documents: 30 about databases, 80 about networks, none about theory, and 10 about languages. The rest of the rows represent a compound RI. In the example, the RI shows that node A can access 100 database documents through D (60 in D, 25 in I, and 15 in J).

Given a query, the termination condition relates to a minimum number of hits. A node that can not satisfy the query stop condition with its local repository will forward it to the neighbor with the highest "goodness" value. Three different functions which rank the out-links according to the expected number of documents that could be discovered through them are proposed. The routing algorithm backtracks if more results are needed. A limitation of this approach is that RIs require flooding in order to be created and updated, so they are not suitable for highly dynamic networks. Moreover, stored indices can be inaccurate due to topic correlations, over-counts or under-counts in document partitioning and network cycles.

2.2.3.2 Peer Information Retrieval

The amount of data published in the internet and its amazing growth rate become beyond centralized web search engines. Recently, P2P systems start to represent an interesting alternative to build large-scale, decentralized Information Retrieval systems.

IR systems define representations of both documents and queries. They may only support a boolean retrieval model, in which documents are indexed and a document can match or not a given query. Note that the local and routing indices described in the above section allow for such a retrieval model. Current IR systems are supporting the retrieval model with a ranking function that quantifies the order amongst the documents matching the query. This becomes essential in the context of large document collections, where the resulting number of matching documents can far exceed the number a user could possibly require. To this end, the IR system defines relationships between document and query representations, so that a score can be computed for each matching document, w.r.t. the query at hand.

A P2P system differs from a distributed IR system in that it is typically larger, more dynamic with node lifetimes measured in hours. Furthermore, a P2P system lacks the centralized mediators found in many IR systems that assume the responsibility for selecting document collections, rewriting queries, and merging ranked results [25]. In the following, we first introduce the main IR techniques used for indexing documents. Then, we present the P2P IR systems that have been proposed in the literature.

Inverted Index The inverted index, or sometimes called inverted file, has became the standard technique in IR. For each term, a list that records which documents the term occurs in is maintained. Each item in the list is conventionally called a *posting*. The list is then called a *postings list* (or inverted list), postings list and all the postings lists taken together are referred to as the postings.

Vector Space Model The representation of the set of documents and queries as vectors in a common vector space is known as the Vector Space Model (VSM) and is fundamental to support the operation of scoring documents relative to a query. Each component of the vector represents the importance of a *term* in the document or query. The weight of a

component is often computed using the *Term Frequency * Inverse Document Frequency* (TF*IDF) scheme.

- *Term Frequency*: the frequency of each term in each document.
- *Inverse Document Frequency*: the document frequency df_t is the number of documents, in a collection of N documents, that contain a term t. The Inverse Document Frequency of term t is given by: $log(N/df_t)$.

Viewing a collection of N documents as a collection of vectors leads to a natural view of a collection as a *term-document matrix*: this is an $M \times N$ matrix whose rows represent the M terms (dimensions) of the N columns, each of which corresponds to a document.

Latent Semantic Index Latent Semantic Index (LSI) uses Singular Value Decomposition (SVD) to transform and truncate the term-document matrix computed from VSM. This allow to discover the semantics underlying terms and documents. Intuitively, LSI transforms a high-dimensional document vector into a medium-dimensional semantic vector by projecting the former into a medium-dimensional semantic subspace. The basis of the semantic subspace is computed using SVD. Semantic vectors are normalized and their similarities are measured as in VSM.

Several solutions for text-based retrieval in decentralized environments have been proposed in the literature.

PlanetP [48] is a publish-subscribe service for P2P communities, supporting content ranking search. PlanetP maintains a detailed inverted index describing all documents published by a peer locally (i.e. a local index). In addition, it uses gossiping to replicate a *term-to-peer* index everywhere for communal search and retrieval. This term-to-peer index contains a mapping $t \rightarrow p$ if term t is in the local index of peer p. PlanetP approximate $TF * IDF$ by dividing the ranking problem into two stages. In first, peers are ranked according to their likehood of having relevant documents. To this end, PlanetP introduces the *Inverse Peer Frequency (IPF)* measure. Similar to IDF, the idea behind is that a term is of less importance if it is present in the index of every peer. Second, PlanetP contacts only the first group of m peers from the top of the peer ranked list, to retrieve a relevant set of documents. It stops contacting peers when the top-k document ranking becomes stable, where k is specified by the user. A primary shortcoming of PlanetP is the large amount of metadata that should be maintained, which restricts its scalability.

The PeerSearch system [42] proposes another approach that places documents onto a DHT network according to their semantic vectors produced by Latent Semantic Indexing (LSI) in order to reduce document dimensionality and guarantee solution scalability. However, as semantic vectors have to be defined a priori, the method cannot efficiently handle dynamic scenarios and adapt to changing collections.

A query-driven indexing method has recently been proposed in [54]. However, the solution is based on single-term indexing and does not consider indexing with term combinations. A recent work proposes the AlvisP2P search engine [138], which enables

retrieval with multi-keywords from a global document collection available in the P2P network. One of the merits of the proposed approach is that indexing is performed in parallel with retrieval. However, a main limitation is that the quality of the answer obtained for a given query depends on the popularity of the term combinations it contains.

2.2.3.3 Peer Data Management

While existing architectures for distributed systems have been reaching their maturity (e.g. distributed database systems, data integration systems), the P2P paradigm has emerged as a promising alternative to provide a large-scale decentralized infrastructure for resource sharing. Grible et al. have addressed an important question "*how data management can be applied to P2P, and what the database community can learn from and contribute to the P2P area?* [53]. The P2P paradigm has gained much popularity with the first successful file sharing systems (e.g. Gnutella, KaZaa) because of the ease of deployment, and the amplification of the desired system properties as new nodes join (i.e. this is aligned with the definition of the P2P paradigm). However, the semantics provided by these systems is typically weak. So far in this report, we have reviewed P2P systems that support key lookups or keyword search. In order to support advanced applications which are dealing with structured and semantically rich data, P2P systems must provide more sophisticated data access techniques. The overlapping of P2P and database areas has lead to a new class of P2P systems, called Peer Data Management Systems (PDMS) or schema-based P2P systems (see Figure 2.12).

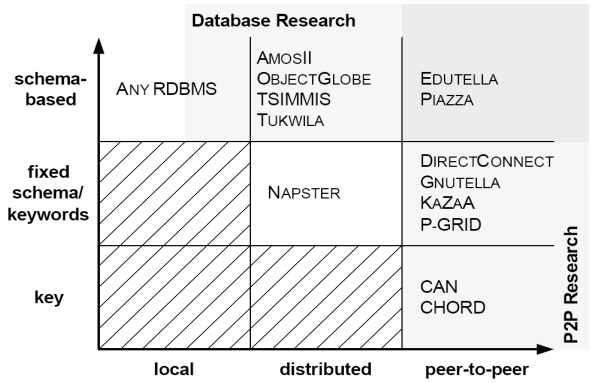

Figure2.12: Schema capabilities and distribution [149]

In distributed databases, the location of content is generally known, the query optimizations are performed under a central coordination, and answers to queries are expected to be complete. On the other side, the ad-hoc and dynamic membership of participants in P2P systems makes difficult to predict about the location and the quality of resources, and to maintain globally accessible indexes which may become prohibitive as the network size grows.

The work that has been done in PDMSs mainly addresses the information integration issue. In fact, the potential heterogeneity of data schemas makes sharing structured data in

P2P systems quite challenging. This issue will be discussed in Section 2.3. Besides, PDMSs have started to study the design and the implementation of complex query facilities (e.g. join and range queries). This is a fundamental building block of a given PDMS which attempts to be a fully distributed data system, with a high level of query expressiveness. Processing complex queries requires the employment of data access techniques which deal with the structure and semantics within data. Section 2.4 discusses complex queries in P2P systems.

2.3 Schema Management in P2P Systems

Semantic heterogeneity is a key problem in large scale data sharing systems [92]. The data sources involved are typically designed independently, and hence use different schemas. To be able to allow meaningful inter-operation between different data sources, the system needs to define schema mappings. *Schema mappings* define the semantic equivalence between relations and attributes in two or more different schemas.

The traditional approach for querying heterogeneous data sources relies on the definition of *mediated schema* between data sources [55] (see Figure 2.13). This mediated schema provides a global unified schema for the data in the system. Users submit their queries in terms of the mediated schema, and schema mappings between the mediated schema and the local schemas allow the original query to be reformulated into subqueries executable at the local schemas. There is a wrapper close to each data source that provides translation services between the mediated schema and the local query language [142].

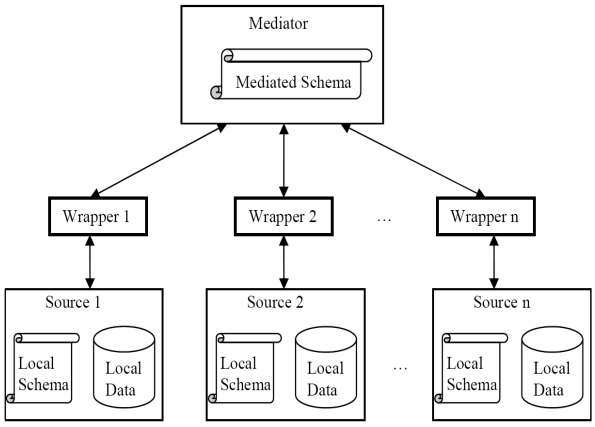

Figure2.13: Schema Mapping using a Global Mediated Schema

In data integration systems, there are two main approaches for defining the mappings: Global-as-view (GAV) which defines the mediated schema as a view of the local schemas, and Local-as-View (LAV) which describes the local schemas as a view of the mediated schema [85]. In GAV, the autonomy of data sources is higher than LAV because they can define their local schemas as they want. However, if any new source is added to a system that uses the GAV approach, considerable effort may be necessary to update the

mediator code. Thus, GAV should be favored when the sources are not likely to change. The advantage of a LAV modeling is that new sources can be added with far less work than in GAV. LAV should be favored when the mediated schema is not likely to change, i.e. the mediated schema is complete enough that all the local schemas can be described as a view of it.

Given the dynamic and autonomous nature of P2P systems, the definition of a unique global mediated schema is impractical. Thus, the main problem is to support decentralized schema mapping so that a query on one peer's schema can be reformulated in a query on another peer's schema. The approaches which are used by P2P systems for defining and creating the mappings between peers' schemas can be classified as follows: pairwise schema mapping, mapping based on machine learning techniques, common agreement mapping, and schema mapping using IR techniques.

2.3.1 Pairwise Schema Mappings

In this approach, the users define the mapping between their local schemas and the schema of any other schema which is interesting for them. Relying on the transitivity of the defined mappings, the system tries to extract mappings between schemas which have no defined mapping.

Piazza [73] follows this approach (see Figure [73]). In Piazza, the data are shared as XML documents, and each peer has a schema, expressed in XMLSchema, which defines the terminology and the structural constraints of the peer. When a new peer (with a new schema) joins the system for the first time, it maps its schema to the schema of some other peers of the network. Each mapping definition begins with an XML template that matches some path or sub-tree of an instance of the target schema, i.e. a prefix of a legal string in the target DTD's grammar. Elements in the template may be annotated with query expressions (in a subset of XQuery) that bind variables to XML nodes in the source.

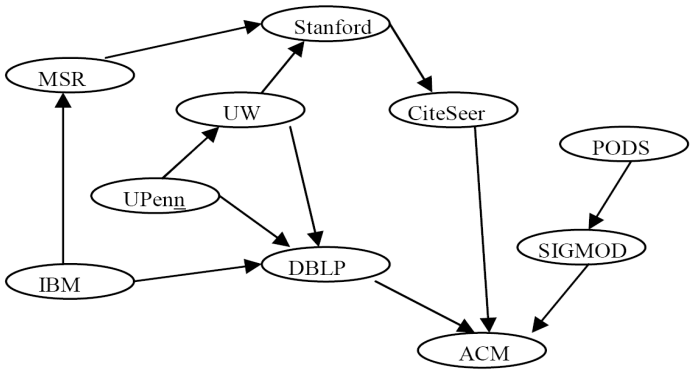

Figure2.14: An Example of Pairwise Schema Mapping in Piazza

The Local Relational Model (LRM) [107] is another example that follows this approach. LRM assumes that the peers hold relational databases, and each peer knows

a set of peers with which it can exchange data and services. This set of peers is called p's *acquaintances*. Each peer must define semantic dependencies and translation rules between its data and the data shared by each of its acquaintances. The defined mappings form a semantic network, which is used for query reformulation in the P2P system.

PGrid also assumes the existence of pairwise mappings between peers, initially constructed by skilled experts [135]. Relying on the transitivity of these mappings and using a gossiping algorithm, PGrid extracts new mappings that relate the schemas of the peers between which there is no predefined schema mapping.

2.3.2 Mapping based on Machine Learning Techniques

This approach is usually used when the shared data is defined based on ontologies and taxonomies as proposed in the Semantic Web [8]. It uses machine learning techniques to automatically extract the mappings between the shared schemas. The extracted mappings are stored over the network, in order to be used for processing future queries.

GLUE [11] uses this approach. Given two ontologies, for each concept in one, GLUE finds the most similar concept in the other. It gives well founded probabilistic definitions to several practical similarity measures. It uses multiple learning strategies, each of which exploits a different type of information either in the data instances or in the taxonomic structure of the ontologies. To further improve mapping accuracy, GLUE incorporates commonsense knowledge and domain constraints into the schema mapping process. The basic idea is to provide classifiers for the concepts. To decide the similarity between two concepts A and B, the data of concept B is classified using A's classifier and vice versa. The amount of values that can be successfully classified into A and B represent the similarity between A and B.

2.3.3 Common Agreement Mapping

In this approach, the peers that have a common interest agree on a common schema description for data sharing. The common schema is usually prepared and maintained by expert users. APPA [120] makes the assumption that peers wishing to cooperate, e.g. for the duration of an experiment, agree on a Common Schema Description (CSD). Given a CSD, a peer schema can be specified using views. This is similar to the LAV approach in data integration systems, except that, in APPA, queries at a peer are expressed in terms of the local views, not the CSD. Another difference between this approach and LAV is that the CSD is not a global schema, i.e. it is common to a limited set of peers with common interest (see Figure 2.15). Thus, the CSD makes no problem for the scalability of the system. When a peer decides to share data, it needs to map its local schema to the CSD. In APPA, the mappings between the CSD and each peer's local schema are stored locally at the peer. Given a query Q on the local schema, the peer reformulates Q to a query on the CSD using locally stored mappings.

AutoMed [113] is another system that relies on common agreements for schema mapping. It defines the mappings by using primitive bidirectional transformations defined

in terms of a low-level data model.

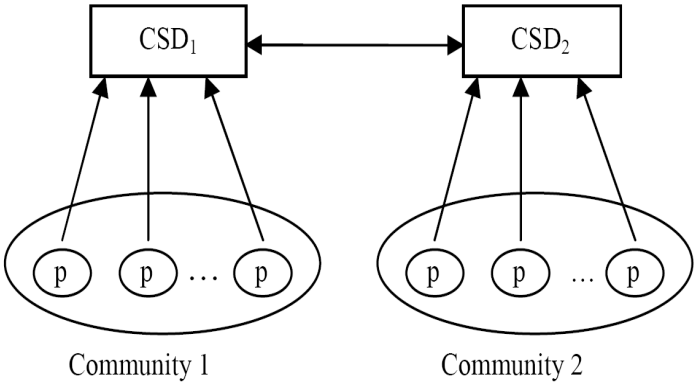

Figure2.15: Common Agreement Schema Mapping in APPA

2.3.4 Schema Mapping using IR Techniques

This approach extracts the schema mappings at query execution time using IR techniques by exploring the schema descriptions provided by users. PeerDB [102] follows this approach for query processing in unstructured P2P networks. For each relation which is shared by a peer, the description of the relation and its attributes is maintained at that peer. The descriptions are provided by users upon creation of relations, and serve as a kind of synonymous names of relation names and attributes. When a query is issued, some agents are flooded to the peers to find out potential matches and bring the corresponding meta-data back. By matching keywords from the meta-data of the relations, PeerDB is able to find relations that are potentially similar to the query relations. The found relations are presented to the user who has issued the query, and she decides on whether or not to proceed with the execution of the query at the remote peer which owns the relations.

Edutella [137] also follows this approach for schema mapping in super-peer networks. Resources in the Edutella are described using the RDF metadata model, and the descriptions are stored at superpeers. When a user issues a query at a peer p, the query is sent to p's super-peer where the stored schema descriptions are explored and the address of the relevant peers are returned to the user. If the super-peer does not find relevant peers, it sends the query to other super-peers such that they search relevant peers by exploring their stored schema descriptions. In order to explore stored schemas, super-peers use the RDF-QEL query language. RDF-QEL is based on Datalog semantics and thus compatible with all existing query languages, supporting query functionalities which extend the usual relational query languages.

Independently of the approach used to implement the schema mappings, P2P systems attempt to exploit the transitive relationships among peer schemas to perform data sharing and integration [156]. While in traditional distributed systems, schema mappings form a semantic tree, in P2P systems the mappings form a *semantic graph*. By traversing

semantic paths of mappings, a query over one peer can obtain relevant data from any reachable peer in the network. Semantic paths are traversed by reformulating queries at a peer into queries on its neighbors.

2.4 Querying in P2P Systems

The support of a wider range of P2P applications motivates the evolution of current P2P technologies in order to accommodate many search types. As said before, a P2P system should support the operating application with an appropriate level of query expressiveness. Advanced applications which are dealing with semantically rich data require an expressiveness level higher than filename-based or key-based lookup. In the following, we discuss the different techniques used for processing complex queries in P2P systems. These querying techniques can be distinguished according to:

- *Search completeness:* the network is entirely covered by the search mechanism. Relaxing the search completeness leads to *partial lookup*.
- *Result completeness:* the found result set is entirely returned to the user. Relaxing the result completeness leads to *partial answering*.
- *Result granularity:* generally, the returned results are retrieved from, and thus have the same type as, the original queried data (e.g. music files, XML documents, database tuples). Returning results at a different level of granularity (by making data abstraction) leads to *approximate answering*. The term "approximate" may still be ambiguous, due to its wide employment in query processing proposals. However, the following sections tend to give a precise definition of what we are referring to by "approximate answers".

2.4.1 Partial Lookup

The advantages of P2P data sharing systems, like scalability and decentralization, do not come for free. In large-scale dynamic systems, it is nearly impossible to guarantee a complete search. Let Q be a query issued by a peer p in the system, and P_Q the set of relevant peers, i.e. the peers that store, at least, one query result. Q is said to be a *total-lookup* query if it requires *all* the results available in the system. Here, the set P_Q should be entirely visited. In the case where Q requires *any* k results, Q is said to be a *partial-lookup* query.

The impracticality of an exhaustive flooding, the limited knowledge provided by indexes and the errors they may contain, and the incorrect semantic mappings are reasons among others for considering that all queries in P2P systems are in reality processed as being partial-lookup queries. In other terms, the query Q issued by peer p in a P2P network of size N will be routed in a subnetwork of size N', and thus a subset $P'_Q \subseteq P_Q$ can be targeted. The filename-based, key-based and keyword-based searches have been presented

earlier in this chapter. Here, we discuss four types of complex queries: range, multi-attribute, join, and fuzzy queries. The importance of these queries has been recognized in many distributed environments (e.g. parallel databases, Grid resource discovery) since they significantly enhance the application ability to precisely express its interests.

2.4.1.1 Range Queries

Range queries are issued by users to find all the attribute values in a certain range over the stored data.

Several systems have been proposed to support range queries in P2P networks. The query processing in these systems rely on underlying DHTs, or other indexing structures. Some argue that DHTs are not suited to range queries [12]. The hash functions used to map data on peers achieve good load balancing, but do not maintain data proximity, i.e. the hash of two close data may be two far numbers. Despite of this potential shortcoming, there have been some range query proposals based on DHTs.

Instead of using uniform-hashing techniques, Gupta *et al* [12] employ locality sensitive hashing to ensure that, with high probability, similar ranges are mapped to the same peer. They propose a family of locality sensitive hash functions, called min-wise independent permutations. The simulation results show good performance of the solution. However, there is the problem of load unbalance for large networks. In [105] the authors extend the CAN protocol using the Hilbert space-filling curve and load balancing mechanisms. Nearby ranges map to nearby CAN zones, and if a range is split into two sub-ranges, then the zones of the sub-ranges partition the zone of the primary range. Thus, the one-dimensional space of data items is mapped to the multi-dimensional CAN zones. Conversely, multi-dimensional data items are mapped to data points in one-dimensional space through the space-filling curve in [41]. Such a construction gives the ability to search across multiple attributes.

Some works rely on Skip list data structure which, unlike DHT, does not require randomizing hash functions and thus can support range queries. SkipNet [103] is a lexicographic order-preserving DHT that allows data items with similar values to be placed on contiguous peers. It uses names rather than hashed identifiers to order peers in the overlay network, and each peer is responsible for a range of strings. This facilitates the execution of range queries. However, it is not efficient because the number of peers to be visited is linear in the query range.

Other proposals for range queries avoid both DHT and Skip list structures. P-Grid [135] is based on a randomized binary prefix tree. One limitation is that P-Grid considers that all nodes in the system have a fixed capacity, and content is heuristically replicated to fill all the node capacity. However, there is no formal characterization of either the imbalance ratio guaranteed, or the data-movement cost incurred.

BATON [65] is a balanced binary search tree with in-level links for efficiency, fault-tolerance, and load-balancing. VBI-tree [66] proposes a virtual binary overlay which is an enhancement of BATON, and focuses on employing multi-dimensional indexes to support more complex range query processing. Common problems to balanced tree overlay

structures is that peer joining or leaving can cause a tree structural change, and the update strategies may get prohibitive under a high churn environment.

The work on Scalable Distributed Data Structures (SDDSs) has progressed in parallel with P2P work and has addressed range queries. Like DHTs, the early SDDS Linear Hashing (LH*) schemes were not order preserving [87]. To facilitate range queries, a range partitioning variant, RP* has been proposed in [88]. In [139], the authors investigated TCP and UDP mechanisms by which servers could return range query results to clients.

2.4.1.2 Join Queries

Distributed data among peers could be seen, in some cases, as a set of large relational tables fragmented horizontally. Running efficient join queries over such massively dispersed fragments is a challenging task. Two research teams have done some initial works on P2P join operations.

In [125], the authors describe a three layer architecture of the PIER system and implement two equi-join algorithms. In their design, a key is constructed from a "namespace" (relation) and a "resourceID" (primary key by default). Queries are multicast to all peers in the two namespaces to be joined. The first algorithm is a version of the symmetric hash join algorithm [WA91]. Each peer in the two namespaces finds the relevant tuples and hashes them to a new query namespace. The resource ID in the new namespace is the concatenation of join attributes. The second algorithm, called "fetch matches", assumes that one of the relations is already hashed on the join attributes. Each peer in the second namespace finds tuples matching the query and retrieves the corresponding tuples from the first relation. The authors leverage two other techniques, namely the symmetric semi-join rewrite and the Bloom filter rewrite, to reduce the high bandwidth overheads of the symmetric hash join. For an overlay of $10,000$ peers, they evaluated the performance of their algorithms through simulation. The results show good performance of the proposed algorithms. However, for the cases where the join relations have a large number of tuples, this solution is not efficient, especially in terms of communication cost.

In [116], the authors considered multicasting to a large number of peers inefficient. Thus, they propose using a set of dedicated peers called range guards to monitor partitions of join attributes. Join queries are therefore sent only to range guards which decide the peers that should be contacted to execute the query.

2.4.1.3 Multi-Attributes Queries

There has been some work on multi-attribute P2P queries. The Multi-Attribute Addressable Network (MAAN) [96] is built on top of Chord to provide multi-attribute and range queries. They use a locality preserving hash function to map attribute values to the Chord identifier space, which is designed with the assumption that the data distribution could be known beforehand. Multi-attribute range queries are executed based on single-attribute resolution in $O(logn + n * s_{min})$ routing hops, where n is the number of peers of the DHT and s_{min} is the minimum range selectivity across all attributes. The range

selectivity is defined to be the ratio of the query range to the entire attribute domain range. However, the authors notice that there is a query selectivity breakpoint at which flooding becomes better than their scheme. Another drawback of MAAN is that it requires a fixed global schema which is known in advance to all peers. The authors followed up with the RDFPeers system to allow heterogeneity in peers schemas [95]. Each peer contains RDF based data items described as triples $\langle subject, predicate, object \rangle$. The triples are hashed onto MAAN peers. The experimental results show improvement in load balance, but no test for skewed query loads was done.

2.4.1.4 Fuzzy Queries

Information Need vs Query: In information systems, the *information need* is what the user (or group of users) desires to know from the stored data, to satisfy some intended objective (e.g. data analysis, decision making). However, the *query* is what the user submits to the system in an attempt to get that information need.

Precision vs Accuracy: Let us examine what is the relation between *precise* query statements and the *accuracy* of the returned results according to the information need. Consider the following relational table (Table 2.2) that maintains some patient records in a given hospital[1]. Suppose now that a doctor requires information about young patients

Table2.2: Patient Table

Id	Age	Sex	BMI	Disease
t_1	36	$female$	17	$Malaria$
t_2	23	$male$	20	$Malaria$
t_3	45	$female$	16.5	$Anorexia$
t_4	33	$female$	23	$Malaria$
t_5	55	$female$	21	$Rheumatism$
t_6	19	$male$	18	$Malaria$

diagnosed with Malaria (i.e. the information need). In a conventional SQL query, we must decide what are the ages of people considered as young. In the case where such age values fall into the $[21, 35]$ range, the SQL query is written as follows:

```
Select all From Patient Where age in [21, 35] and Disease = Malaria
```

The query above will return two tuples: t_2 and t_4. Unlike other contexts where the *young* term is well defined, such in banking applications where the age of clients precisely decides of the advantages they may benefit from, this term may not have a precise definition in biological contexts. For instance, the tuple t_6 may bring additional information to the doctor, affecting its analysis or decision. From this point of view, we say that the query results are not accurate, although the query has been precisely stated. One way to include tuple t_6 in the result set is to expand the scope of the selection predicate in order to encompass more data. Thus, the previous query is modified as follows:

[1]Body Mass Index (BMI) attribute: patient's body weight divided by the square of the height.

```
Select all From Patient Where age in [18, 35] and Disease = Malaria
```

Although it selects more tuples, the query still fails to find tuples lying just outside the explicit range of the selection predicate (e.g. tuple t_1). This is due to the crisp boundaries of the search range. Furthermore, there is no measure of inclusion, i.e. there is no way to know which tuples are strongly satisfying the information need and which are weakly satisfying it. **Introducing fuzziness into user queries is a viable solution for that problem, i.e. introducing some imprecision in query statements may in some cases improve accuracy**.

A fuzzy set is a class with *unsharp* boundaries. The grade of membership of an object in a fuzzy set is a number in the unit interval or, more generally, a point in a partially ordered set [82].

The application of gradual predicates, such as the YOUNG predicate presented in Figure 2.16, results in associating membership degrees to tuples in a PATIENT relational table. For example, a tuple whose attribute value $t.age$ is equal to 21 will be associated with a membership degree of 0.5 according to the YOUNG predicate. Hence, tuples can "partially" belong to the result set depending on how well they fit the information need.

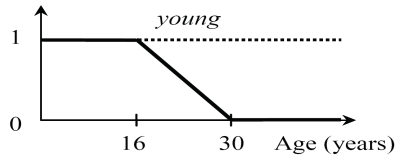

Figure2.16: Gradual predicate on attribute AGE

In [122], fuzzy techniques have been used in the design of P2P reputation systems based on collecting and aggregation peers' opinions. Characterizing peer's reputation by either "bad" or "good" based on some defined threshold is not adequate, as it would characterize in the same way a positive reputation produced by the collection of only positive opinions by many users and a reputation built with a limited number of heterogeneous opinions that produce a value immediately above the threshold; the same reasoning can be applied to negative reputations.

A recent work tends to introduce fuzziness into the BestPeer platform [150]. In [110], the authors propose FuzzyPeer, a generic P2P system which supports similarity queries. An image retrieval application is implemented as a case study. Fuzzy queries like "find the top-k images which are similar to a given sample" are very common in such applications because it is difficult for humans to express precisely an image's content in keywords or using precise attribute values. The authors investigate the problem of resolving similarity queries. The approach that consists in setting a similarity threshold and accepting objects only above this value is rejected. In fact, choosing the threshold value is not trivial given that the interpretation of an image depends on the user's perception of the domain. The approach proposed in [110] is based on the following observation: if two queries are similar, the top-k answers for the first one may contain (with high probability) some of the answers for the second query. In FuzzyPeer some of the queries are paused (i.e. they are

not propagated further) and stay resident inside a set of peers. We use the term *frozen* for such queries. The frozen queries are answered by the stream of results that passes through the peers, and was initiated by the remaining running queries. Then, the authors propose distributed optimization algorithm in order to improve the scalability and the throughput of the system.

2.4.2 Partial Answering

As seen before, a first repercussion of the scale of P2P systems on query processing is that all queries can *partially* search the network. Another issue is that the amount of available data in P2P systems is dramatically increasing. More specifically, it becomes difficult to retrieve a few data items within a large structured data set in current PDMSs. Consider that a user issues the following query Q: *select hotels in Nice where price < 100(euros) and proximity < 8(km)*. The set R_Q of results returned by the set of relevant peers P'_Q may include a number of hotels that is so far from the one required by the user. Therefore, rank-aware queries like top-k and skyline queries started to emerge in order to provide a partial result subset R'_Q, with the k results having the highest grades of membership to the result set R_Q, i.e. $R'_Q \subseteq R_Q$. Indeed, the user is interested in the most relevant available results, which may be specified in the query as follows: *select hotels with cheap price, and yet close to the beach*. The degree of relevance (score) of the results to the query is determined by a scoring function.

Ranking results in a distributed manner is difficult because ranking is global: *all* results (matching a query) have to be ranked w.r.t. each other. In a completely distributed system, the results returned for identical queries should ideally be the same, which is not an issue in a centralized implementation. In a large-scale P2P system, the lack of a central location to aggregate global knowledge makes the problem of ranking challenging.

2.4.2.1 Top-k Queries

Given a dataset D and a scoring function f, a top-k query retrieves the k data items in D with the highest scores according to f. The scoring function is specified by the user according to its criteria of interests.

In unstructured P2P systems, one possible approach for processing top-k queries is to route the query to all peers, retrieve all available answers, score them using the scoring function, and return to the user the k highest scored answers. However, this approach is not efficient in terms of response time and communication cost. Top-k is a popular aspect of IR. As mentioned before, PlanetP [48] supports content ranking search in Peer IR systems. The top-k query processing algorithm works as follows. Given a query Q, the query originator computes a relevance ranking of peers with respect to Q, contacts them one by one from top to bottom of ranking and asks them to return a set of their top-scored document names together with their scores. To compute the relevance of peers, a global fully replicated index is used that contains term-to-peer mappings. This algorithm has very good performance in moderate-scale systems. However, in a large P2P system,

keeping up-to-date the replicated index is a major problem that hurts scalability.

In the context of APPA, a fully distributed solution is proposed to execute top-k queries in unstructured P2P systems [118]. The solution involves a family of algorithms that are simple but effective. It executes top-k queries in completely distributed fashion and does not depend on the existence of certain peers. It also addresses the volatility of peers during query execution and deals with situations where some peers leave the system before finishing query processing.

In [147], the authors leverage the usage of super-peer networks, and propose an algorithm for distributed processing of top k queries on the top of Edutella [137]. In Edutella, a small percentage of nodes are super-peers and are assumed to be highly available with very good computing capacity. The super-peers are responsible for top-k query processing and other peers only execute the queries locally and score their resources. A limitation of this framework is that it assumes a global shared schema as well as consistent ranking methods employed at peers.

As for other complex queries, processing top-k queries in DHTs is quite challenging. A solution is to store all tuples of each relation by using the same key (e.g. relation's name), so that all tuples are stored at the same peer. Then, top-k query processing can be performed at that central peer using well-known centralized algorithms. However, the central peer becomes a bottleneck and single point of failure. In the context of APPA, a recent work has proposed a novel solution for Top-k query processing in DHT systems [119]. The solution is based on the TA algorithm [FLN03, GKB00, NR99] which is widely used in distributed systems. The solution is based on a data storage mechanism that stores the shared data in the DHT in a fully distributed fashion, and avoids skewed distribution of data among peers.

2.4.2.2 Skyline Queries

Top-k queries are sometimes difficult to define, especially if multiple aspects (i.e. scoring functions) have to be optimized. It is often not clear how to weight these aspects in order to obtain a global rank. Given such a multi-preference criteria, the concept of skyline queries provide a viable solution by finding a set of data points that are not *dominated* by any other points in a given data set. A point dominates another point if it is no worse in all concerning dimensions and better in at least one dimension according to user preferences. Objects belonging to skyline are precisely those objects that could be the best under some monotonic scoring functions. Most existing studies have focused mainly on centralized systems, and resolving skyline queries in a distributed environment such as a P2P network is still an emerging topic.

[117] is the first attempt on progressive processing of skyline queries on a P2P network such as CAN [131]. The authors present a recursive region partitioning and a dynamic region encoding method to enforce a partial order over the CAN's zones, so that all the participating machines can be correctly pipelined for query execution. During the query propagation, data spaces are dynamically pruned and query results are progressively generated. Therefore, users do not have to wait for query termination to receive partial

results, substantially reducing the query response time. However, this work focuses only on *constrained skyline queries* [47] where users are only interested in finding the skyline points among a subset of data items that satisfies multiple *hard* constraints. Besides, it suffers from workload imbalance caused by skewed query ranges.

A more recent work [134] has proposed an efficient solution for skyline query processing in the context of BestPeer. BestPeer [150] is a P2P platform that supports both structured and unstructured overlays. The solution proposed in [134] is called *Skyline Space Partitioning* (SSP), and is implemented in the BestPeer's structured network, called BATON [65]. It supports processing *unconstrained* skyline queries, which search skyline points in the whole data space. This work deals with the issue of imbalanced query load.

2.4.3 What about Approximate Answering?

To fix the ideas previously presented, and to eliminate any ambiguity, we precise here that "*approximate answering*",

- *Is not only about approximating the search space:* In Section 2.4.1, the fact of relaxing search completeness, due either to the limited coverage of routing protocols or to the inaccuracy of data indexes, has been referred as *"partial lookup"*.

- *Is not only about introducing flexibility into user's queries:* By flexible queries we refer to queries that may contain keywords, wildcards, ranges, or include user's preferences (e.g. top-k, skyline queries) or user's perception of the queried domain (e.g. fuzzy queries). This flexibility certainly supports users with more facilities to express their interests.

- *Is not only about approximating query evaluation techniques:* Query evaluation techniques, which are initially defined in centralized environments, can be only approximated in the context of P2P systems. Examples are [78] in which the notion of *relaxed skyline* is introduced, and [69] in which the well-known TA algorithm [141] is extended to adapt to P2P scenarios. For more illustration, the top-k answers returned to a user in a P2P system do not exactly match the set of top-k answers which would be obtained if all data were available and processed under a central coordination. This is considered as a natural repercussion of the nature of P2P networks on any computation method requiring some global information.

- *It is about returning approximate results, represented at a different level of abstraction:* As P2P systems start getting deployed in e-business and scientific environments, the vast amount of data within P2P databases poses a different challenge that has not been intensively researched until recently. In collaborative and decision support applications, a user may prefer an *approximate* but fast answer. Approximate answers do not belong to the original result set R_Q. However, they provide data descriptions $\tilde{R}_Q$, which may be queried or used as an alternative dataset for other operations input, including querying, browsing, or data mining.

Aggregation Queries. Aggregation queries have the potential of finding applications in decision support, data analysis and data mining. For example, millions of peers across the world may be cooperating on a grand experiment in astronomy, and astronomers may be interested in asking decision support queries that require the aggregation of vast amounts of data covering thousands of peers [24].

Consider a single table T that is horizontally partitioned and distributed over a P2P system. An aggregation query can be defined as follows:

```
Select Agg-Op(col) From T Where selection-condition
```

The *Agg-Op* may be any aggregation operator such as SUM, COUNT, AVG, MAX, and MIN. *Col* may be any numeric column of T, or even an expression involving multiple columns, and the *selection-condition* decides which tuples should be involved in the aggregation. Recently, traditional databases and decision support systems have witnessed the development of new *Approximate Query Processing techniques* AQP (e.g. [26], [152]) for aggregation queries. These techniques are mainly based on sampling, histograms, and wavelets.

Initial works have aimed to support aggregation queries in P2P systems by introducing OLAP techniques which employ materialized views over data ([109], [99]). However, the distribution and management of such views seems to be very difficult in such dynamic and decentralized environments. A recent work has investigated the feasibility of online sampling techniques for AQP in P2P systems [24]. The authors abandon trying to pick uniform random samples, which are nearly impossible to obtain in P2P systems. Instead, they have proposed to work with *skewed samples* while being able to accurately estimate the skew during sampling process.

Aggregation queries provide aggregate values which support the user/application with information about tendencies within data. For example, the data cube [52], which is the most popular data model used for OLAP systems, generalizes the *GROUP BY* operation to N dimensions. Pre-computed aggregate values are stored in the cube cells and then, the OLAP system provides tools to navigate within these cells. This allows, for example, to examine the total number of sales of a given product in the last week of the current year, which have been reported in all cities of France (see Figure 2.17).

Fuzzy summaries. As seen before, fuzziness can be introduced into the user interface to allow more flexibility in query formulation. Fuzzy queries may be interpreted in a quantitative preference framework, provided that: 1) a membership function gives a similarity value of tuples to elementary query requirements (the fuzzy or gradual predicates) and 2) fuzzy aggregation computes an overall score that allows ranking items in the result set. However, we believe that it could not be the users' very first intention when they deal with such fuzzy queries. The simple fact that they need to define membership functions to compute attribute-oriented scores is somehow less natural than explicitly formulating preferences into query [144].

The literature also offers studies of how to express concepts or needs through constructs such as operators or linguistic variables [83]. One of the main challenges of extending query

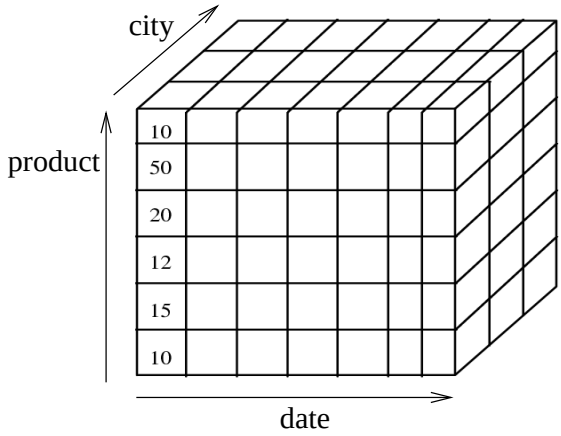

Figure2.17: Data cube

languages is to enrich query formulation without drastically reducing the performance of the query evaluation process. Linguistic summaries, studied by Yager *et al* in [153], serve that concern by expressing the content of a set of data. The new expression is a description of the data using linguistic terms. Many works, some prior to Yager's, fall into the domain of linguistic summaries.

Quantified summaries approaches [114, 123] use fuzzy quantifiers in addition to linguistic terms to describe the data. For instance, in SummarySQL [123], evaluating "summary most from PATIENTS where age is young" provides a degree of validity for the proposition "most PATIENTS are young." Linguistic summaries also comprise fuzzy rules-based summaries. Such summaries are discovered by searching associations and relations between attribute values [31] or by exploiting fuzzy functional dependencies [30, 43]. They produce, in the case of gradual rules of Bosc *et al* [31], propositions such as "the more age is old, the more patient day is high[2]." It is also possible to summarize records by repeatedly generalizing linguistic descriptions. This approach uses techniques from automatic learning and classification. Its output is a tree of descriptions. Lee and Kim's "is-a" hierarchies [84] and the SaintEtiQ model [51] are instances of this approach.

At the end of this chapter, we lighten the importance of querying such fuzzy summaries in centralized as well as in distributed P2P environments. First of all, these database summaries are a means of significantly reducing the volume of input for processes that require access to the database. The response time benefits from the downsizing. However, this response time gain is made clearly at the expense of a loss of precision in the answer (i.e. this what we are calling *approximate answering*). This is of no importance when only a rough answer is required. Besides, imprecision can be sometimes a requirement. This is the case for instance when querying a medical database for anonymous, statistical information. Indeed, precise information can violate medical confidentiality. The loss of precision is also of no importance when a request only aims at determining the absence

[2]Patient day: number of days spent in a hospital.

of information in a database. This is the case when one wants to know if a database is likely to answer the query.

Existing techniques have been proposed for querying fuzzy summaries in centralized environments [106, 146], however, such techniques have not been studied in P2P environments yet. The next chapter of this thesis proposes a solution for managing fuzzy summaries in P2P systems to support DB applications with approximate query facilities.

Chapter 3

Summary Management in P2P Systems

In this chapter, we propose a new solution for managing linguistic summaries of structured data in P2P systems. The benefits of this solution are threefold:

- *Semantic-based query routing:* linguistic summaries are considered as semantic indexes that allow locating relevant nodes based on their data descriptions.
- *Flexible query formulation:* querying linguistic summaries is supported with a user interface that allows to formulate queries using linguistic terms. Linguistic terms are concepts defined over the data attribute domains, and thus represent the user's perception of the shared data.
- *Approximate query answering:* linguistic summaries provide an intelligible representation of the underlying data, and thus allow to return approximate query answers. Remind that approximate answers are synthesized information, provided at a higher abstraction level of the original queried data.

A proposal for summary management in P2P systems should satisfy two requirements:

- The employment of a summarization technique that permits, at each peer, to summarize a voluminous data source in a resource-efficient manner. The produced summaries should satisfy some properties to allow obtaining the desired benefits (listed above), expected from their exploitation.
- The definition of efficient techniques that allow cooperating peers to exchange and maintain summaries over their shared data in a decentralized manner, and without generating large traffic overhead on the P2P network.

Accordingly, we make the following contributions. In Section 3.1, we first propose a summarization process that exhibits many salient features making from it an interesting process to be integrated into P2P environments. In Section 3.2, we study the feasibility of

its integration into an existing Peer Data Management System (PDMS). So, we work in the context of APPA (*Atlas Peer-to-Peer Architecture*) [120], a PDMS whose main objective is to provide high level services for advanced P2P applications. Then, in Section 3.3, we define a summary model in the context of hierarchical P2P systems. Here, we relax the assumptions relying on APPA's services, and which mainly abstract the summary distribution issue.

Having defined the solutions for summary creation/maintenance in both contexts, we discuss in Section 3.4 the gains that might be obtained in query processing from the use of data summaries. In Section 3.5, the performance of our proposals are evaluated through a cost model and a simulation model. Simulation results show that the cost of query routing is significantly reduced without incurring high costs of summary updating. Section 3.6 concludes.

3.1 Summarization Process

In this section, we first briefly discuss the existing approaches of data summarization according to the characteristics required for our purposes. Then, we define the input data and describe the summarization process of the approach we have adopted in our work. Finally, we formally define the structure of distributed summaries in P2P systems.

3.1.1 Data Summarization

The data summarization paradigm has been extensively studied in centralized environments to address the problem of managing voluminous data sets. In the literature, many approaches have been proposed, each satisfying the requirements of specific applications. Here, we discuss the required characteristics that have controlled the choice of our data summarization process.

- **Compression:** the summarization process should provide compact versions of the underlying data. However, we are not referring here to (syntactic) compression techniques. These techniques consider a data source as a large byte string and thus use compression algorithms such as Huffman or Lempel-Ziv coding. They were mainly proposed to deal with throughput and space storage constraints. In fact, we are concerned with compression (or data reduction) techniques that can provide fast and approximate answers. This is very efficient in the context where obtaining an exact answer is a time-consuming process, and an approximate answer may give information that are sufficiently satisfying. Examples of such techniques are discrete wavelet transform and linear regression used in signal processing [115], sampling and histograms [71] used in statistics. Index trees such as B^+Tree [40] and K-d-Tree [28] could also be considered as data reduction techniques in the database field. These indexes do not provide approximate answers, however, they allow to optimize and accelerate the access within a large data set.

- **Intelligibility:** the summarization process should synthesize the information disseminated within large datasets in order to provide the *essential*. In fact, the data summarization techniques are data reduction techniques that are supposed to provide intelligible representations of the underlying data. Three categories of database summarization techniques have been proposed in the literature. The first one focuses on aggregate computation. Examples are OLAP and multidimensional databases which allow an end-user to query, visualize and access part of the database using cubes of aggregate values computed from raw data [22]. The second category, so-called semantic compression (SC), deals with intentional characterization of groups of individuals to provide higher-level models such as decision trees or association rules. Examples are [128], [67] which explicitly refer to semantic compression of structured data.

  The last category is interested in metadata-based semantic compression (MDBSC) approaches. These approaches use metadata to guide the compression process. The produced summaries are highly comprehensible since their descriptions rely on user-defined vocabularies. The main difference between SC and MDBSC is that the latter provide data descriptions which precisely fit the user perception of the domain, whereas the former aims to identify hidden patterns from data. One representative of the MDBSC approaches is the Attribute-Oriented Induction process (AOI). It provides reduced versions of database relations using *is-a* hierarchies, i.e. a concept tree built over each attribute domain [56].

- **Robustness:** the summarization process should handle the vagueness and the imprecision inherent to natural language. The theory of fuzzy sets provides a formal framework associated with a symbolic/numerical interface using linguistic variables [83] and fuzzy partitions [127]. Hence, the linguistic summaries have the advantages of being robust and formulated in a user-friendly language (i.e. linguistic labels).

- **Scalability:** the summarization process should be scalable in terms of the amount of processed data. It should be able to treat voluminous databases with low time complexity, and controlled memory consumption. Besides, an important issue is the ability of the process to be parallelized and distributed among multiple processors or computers.

In our work, the approach used for data summarization is based on SaintEtiQ [126]: an online linguistic approach for summarizing databases. The SaintEtiQ model aims at apprehending the information from a database in a synthetic manner. This is done through generating linguistic, multidimensional summaries which are arranged and incrementally maintained in a hierarchy. The hierarchies of SaintEtiQ summaries differ from those obtained by the *is-a* approach in that they rely on one set of linguistic terms, without level assignment. Indeed, the SaintEtiQ model proceeds first in an abstraction of data by the use of a user-defined vocabulary, and then in performing a classification that produces data descriptions at different levels of granularity (i.e. levels of abstraction). The SaintEtiQ

process exhibits many interesting features and is proposed to be efficiently integrated into P2P environments, as it is revealed in the rest of this section.

3.1.2 Input Data

The summarization process takes as input the original data to be summarized, and the *"Background Knowledge"* BK which guides the process by providing information about the user perception of the domain.

3.1.2.1 Data Model

The data to be summarized come from relational databases. As such, the data are organized into records, with a schema $R(A_1, A_2, \ldots, A_n)$. Each attribute A_i is defined on an attribute domain D_i, which may be numeric or symbolic. Thus, each tuple t consists of n attribute values from domains D_1 to D_n. It is given by:

$$t = \langle t.A_1, t.A_2, \ldots, t.A_n \rangle$$

A constraint on these data is that it should be complete: any value $t.A_i$ is necessarily known, elementary, precise, and certain. In other terms, all records with a null value are dismissed.

3.1.2.2 Background Knowledge

A unique feature of the summarization system is its extensive use of a *Background Knowledge (BK)*, which relies on linguistic variables and fuzzy partitions.

Consider the following relational database of a given hospital, which is reduced to a single *Patient* relation (Table 3.1)[1]. Figure 3.1 shows a linguistic variable defined on the attribute AGE where descriptor YOUNG ADULT is defined as being plainly satisfactory to describe values between 19 and 37 and less satisfactory as the age is out of this range. Similarly, Figure 3.2 provides the linguistic variable defined on attribute BMI[2]. Thus, linguistic variables come with linguistic terms (i.e. descriptors) used to characterize domain values and, by extension, database tuples. For a continuous domain D_i, the linguistic variable is a fuzzy partition of the attribute domain. For a discrete domain D_i (DISEASE and SEX in our example), the BK element is a fuzzy set of nominal values. In short, the BK supports the summarization process with means to match attribute domain values with the summary expression vocabulary.

Note that the BK, given by users or experts of the data domain, concerns the attributes which are considered as pertinent to the summarization process. In our example, we have excluded the *patient day* attribute. However, when the descriptions of that attribute values might be useful for the hospital application (i.e. requiring information about the length of stay of patients), a corresponding fuzzy partition should be also provided.

[1]Patient day: number of days spent in the hospital.

[2]Recall that BMI is the patient's body weight divided by the square of its height.

Id	Age	Sex	BMI	Patient days	Disease
t_1	16	*female*	16	350	*Anorexia*
t_2	60	*male*	32	4	*High blood pressure*
t_3	18	*female*	17	280	*Anorexia*
t_4	17	*female*	18	230	*Anorexia*
t_5	54	*female*	26	14	*Osteoporosis*

Table3.1: Raw data

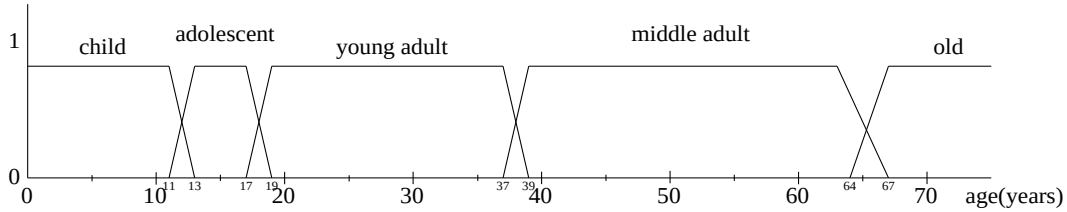

Figure3.1: Fuzzy Linguistic Partition on age

3.1.3 Process Architecture

A service oriented architecture has been designed for the summarization process in order to incrementally build data summaries. By "incremental", we mean that the database tuples are processed one by one, and the final hierarchy of summaries is obtained by repeating this summarization process for each tuple in the table at hand. The architecture is organized into two separate services: online mapping and summarization.

3.1.3.1 Mapping Service

The mapping service takes as input the original relational records and performs an abstraction of the data using the BK. A representation of the data under the form of fuzzy sets is obtained. Basically, the mapping operation replaces the original attribute values of every record in the table by a set of linguistic descriptors defined in the BK. For instance, with the linguistic variable defined on the attribute AGE (Figure 3.1), a value $t.AGE = 18$ years is mapped to $\{0.5/adolescent, 0.5/youngadult\}$ where 0.5 is a membership grade that tells how well the label *young adult* describes the value 18. Extending this mapping

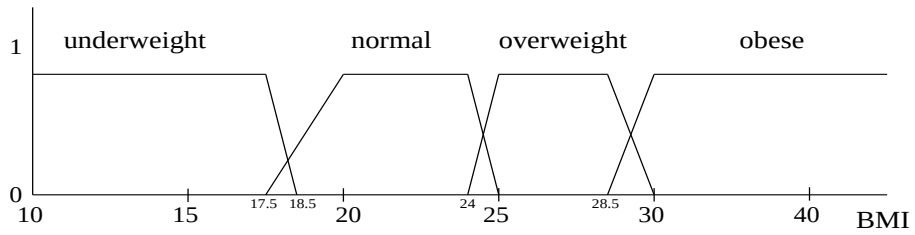

Figure3.2: Fuzzy Linguistic Partition on BMI

to all the attributes of a relation could be seen as locating the overlapping cells in a grid-based multidimensional space which maps records of the original table. The fuzzy grid is provided by the BK and corresponds to the user's perception of the domain.

In our example, tuples of Table 3.1 are mapped into five distinct grid-cells denoted by c_1, c_2, c_3, c_4, and c_5 in Table 3.2. The fuzziness in the vocabulary definition of *BK* permits to express any single value with more than one fuzzy descriptor and thus avoid threshold effect thanks to the smooth transition between different categories. For instance, the tuple t_3 of Table 3.1 is mapped to the two grid cells c_1 and c_2 since the value of the attribute *age* is mapped to the two linguistic terms: *adolescent* and *young adult*. Similarly, the tuple t_4 is mapped to c_1 and c_3 since the value of the attribute *BMI* is mapped to $\{0.7/underweight, 0.3/normal\}$. The tuple count column gives the proportion of records that belongs to the cell, and $0.5/young\ adult$ says that *young adult* fits the data only with a degree of 0.5. The degree of a label is the maximum of membership grades to that label, computed over all the tuples in the corresponding grid cell. For instance, the degree of *adolescent* in c_1 is equal to one, which is the maximum grade value of the two tuples t_1 and t_3.

Id	Age	Sex	BMI	Disease	tuple count
c_1	*adolescent*	*female*	*underweight*	*Anorexia*	2.2
c_2	$0.5/young\ adult$	*female*	*underweight*	*Anorexia*	0.5
c_3	*adolescent*	*female*	$0.3/normal$	*Anorexia*	0.3
c_4	*middle adult*	*male*	*obese*	*High blood pressure*	1
c_5	*middle adult*	*female*	*overweight*	*Osteoporosis*	1

Table3.2: Grid-cells mapping

Therefore, the BK leads to the point where tuples become indistinguishable and then are grouped into grid-cells such that there are finally many more records than cells. Every new (coarser) tuple stores a record count and attribute-dependent measures (min, max, mean, standard deviation, etc.). It is then called a *summary*.

3.1.3.2 Summarization Service

The summarization service is the last and the most sophisticated step of the SaintEtiQ system. It takes *grid-cells* as input and outputs a collection of summaries hierarchically arranged from the most generalized one (the root) to the most specialized ones (the leaves) [126]. Summaries are clusters of grid-cells, defining hyperrectangles in the multidimensional space. In the basic process, leaves are grid-cells themselves and the clustering task is performed on K cells rather than N tuples ($K << N$).

From the mapping step, cells are introduced continuously in the hierarchy with a top-down approach inspired of D.H. Fisher's Cobweb [80], a conceptual clustering algorithm. Then, they are incorporated into best fitting nodes descending the tree. Three more operators could be apply, depending on partition's score, that are *create*, *merge* and *split*

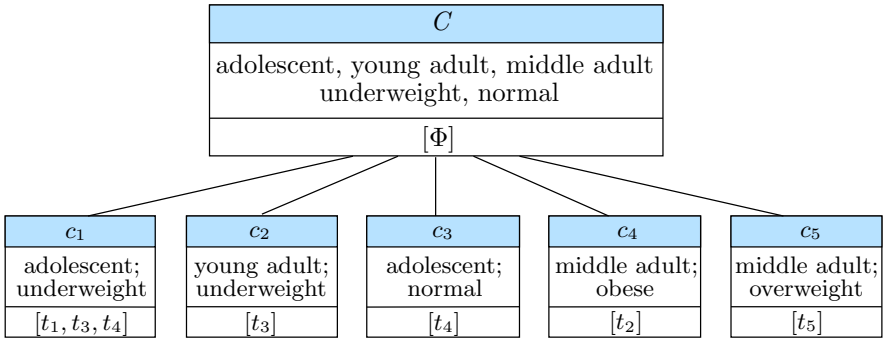

Figure3.3: Example of SaintEtiQ hierarchy

nodes. They allow developing the tree and updating its current state. Figure 3.3 represents the summary hierarchy built from the cells c_1, c_2, c_3, c_4, and c_5.

The summarization process described above has been successfully integrated to an existing DataBase Management System (DBMS) thanks to the design of a service-oriented architecture where service calls are made through the SOAP protocol. On the other hand, the *eXtensible Markup Language* XML is the format adopted for exchanging data with the DBMS, as well as for summary representation all along the summarization process (i.e. raw data, BK, grid cells and the final summaries are represented and exchanged under the form of XML documents).

3.1.3.3 Scalability Issues

Memory consumption and time complexity are the two main factors that need to be taken care of in order to guaranty the capacity of the summary system to handle massive datasets. First, the time complexity of the SaintEtiQ process is in $O(n)$, where n is the number of candidate tuples to incorporate into a hierarchy of summaries. However, the number of candidate tuples that are produced by the mapping service is dependent only on the fuzziness of the BK definition. A crisp BK will produce exactly as many candidate tuples as there are original tuples. Besides, an important feature is that in the summarization process, raw data have to be parsed only once, and this is performed with a low time cost. Second, the system requires low memory consumption for performing the summary construction algorithm as well as for storing the produced summaries. Moreover, a cache manager is in charge of summary caching in memory and it can be bounded to a given memory requirement. Usually, less than a hundred of summaries are needed in the cache since this number covers the two or three top levels of even a wide hierarchy. Least recently used summaries are discarded when a required summary is not found in the cache.

On the other hand, the parallelization of the summary system is a key feature to ensure smooth scalability. The implementation of the system is based on the Message-Oriented Programming paradigm. Each sub-system is autonomous and collaborates with the others through disconnected asynchronous method invocations. It is among the

least demanding approaches in terms of availability and centralization. The autonomy of summary components allows for a distributed computing of the summary process. Once a component completes the treatment and evaluates the best operator for the hierarchy modification, if needed, a similar method is successively called on children nodes. The cache manager is able to handle several lists of summaries residing on different computers [126].

To summarize, our summary system combines advantages such as linear time complexity, controlled memory consumption, and a parallelized computing of the summarization process. Thanks to these advantages, we believe that this summary system is scalable in a distributed environment, and promises a successful integration in P2P systems.

3.1.4 Distributed Summary Representation

In this section, we introduce basic definitions related to the summarization process.

Definition 1 ***Summary*** *Let $E = \langle A_1, \ldots, A_n \rangle$ be a n-dimensional space equipped with a grid that defines basic n-dimensional areas called cells in E. Let R be a relation defined on the cartesian product of domains D_{A_i} of dimensions A_i in E. Summary z of relation R is the bounding box of the cluster of cells populated by records of R.*

The above definition is constructive since it proposes to build generalized summaries (hyper-rectangles) from cells that are specialized ones. In fact, it is equivalent to performing an *addition* on cells:

$$z = c_1 + c_2 + \ldots + c_p$$

where $c_i \in L_z$, the set of p cells (summaries) covered by z.

A summary z is then an *intentional description* associated with a set of tuples R_z as its *extent* and a set of cells L_z that are populated by records of R_z.

Thus, summaries are areas of E with hyper-rectangle shapes provided by BK. They are nodes of the summary tree built by the SaintEtiQ system.

Definition 2 ***Summary Tree*** *A summary tree is a collection S of summaries connected by $\preccurlyeq$, the following partial order:*

$$\forall z, z' \in \mathcal{Z}, \quad z \preccurlyeq z' \Longleftrightarrow R_z \subseteq R_{z'}$$

The above link between two summaries provides a generalization/specialization relationship. And assuming that summaries are hyper-rectangles in a multidimensional space, the partial ordering defines *nested summaries* from the larger one to the single cells. General trends in the data could be identified in the very first levels of the tree whereas precise information has to be looked at near the leaves.

For our purpose, we also consider a summary tree as an indexing structure over distributed data in a P2P system. Thus, we add a new dimension to the definition of a

summary node z: a *peer-extent* P_z, which provides the set of peers having data described by z.

Definition 3 ***Peer-extent*** *Let z be a summary in a given hierarchy of summaries S, and P the set of all peers who participated to the construction of S. The peer-extent P_z of the summary z is the subset of peers owning, at least, one record of its extent R_z: $P_z = \{p \in P \mid R_z \cap R_p \neq \emptyset\}$, where R_p is the view over the database of node p, used to build summaries.*

Due to the above definition, we extend the notion of *data-oriented* summary in a given database, to a *source-oriented* summary in a given P2P network. In other words, our summary can be used as a database index (e.g. referring to relevant tuples), as well as a semantic index in a distributed system (e.g. referring to relevant nodes).

The summary hierarchy S will be characterized by its *Coverage* in the P2P system; that is, the fraction of data sources described by S. Relative to the hierarchy S, we call *Partner Peer* a peer whose data is described by at least a summary of S.

Definition 4 ***Partner peers*** *The set of Partner peers P_S of a summary hierarchy S is the union of peer-extents of all summaries in S: $P_S = \{\cup_{z \in S} P_z\}$.*

For simplicity, in the following we designate by "summary" a hierarchy of summaries maintained in a P2P system, unless otherwise specified.

3.2 Summary Model for APPA

APPA is a P2P data management system developed by our team to provide high level services for advanced P2P applications, which must deal with semantically rich data (e.g. XML documents, relational tables, etc.) [120]. In this section, we present a summary model that aims to successfully integrate a new summary service, PEERSUM, into the APPA architecture. We first present an overview of APPA. Second, we state the problem of managing summaries in P2P systems, and thus propose the architecture of the summary model designed for APPA. Then, we describe the PEERSUM's algorithms for summary management.

3.2.1 APPA

APPA has a layered service-based architecture. Besides the traditional advantages of using services (encapsulation, reuse, portability, etc.), this enables APPA to be network-independent so it can be implemented over different structured (e.g. DHT) and super-peer P2P networks. The main reason for this choice is to be able to exploit rapid and continuing progress in P2P networks. Another reason is that it is unlikely that a single P2P network design will be able to address the specific requirements of many different applications. Obviously, different implementations will yield different trade-offs between performance, fault-tolerance, scalability, quality of service, etc. For instance, fault-tolerance can be

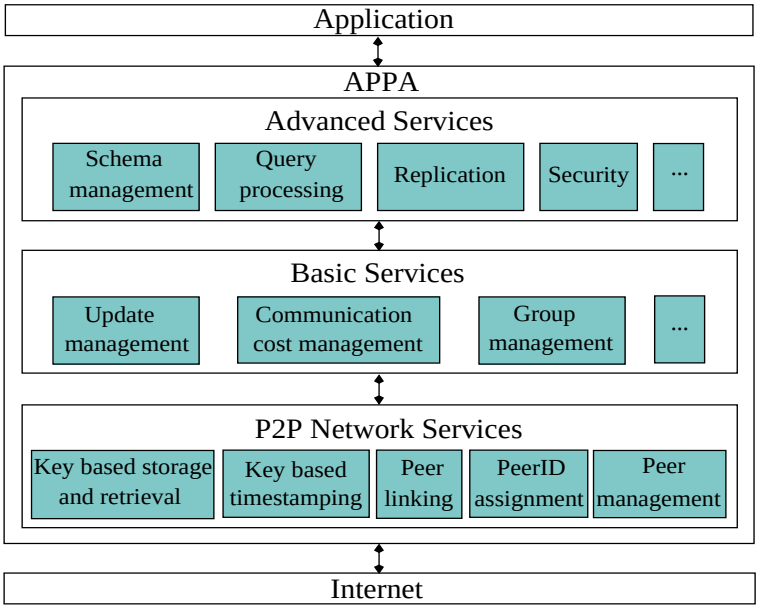

Figure3.4: APPA Architecture

higher in DHTs because no peer is a single point of failure. On the other hand, through index servers, super-peer systems enable more efficient query processing. Furthermore, different P2P networks could be combined in order to exploit their relative advantages, e.g. DHT for key-based search and super-peer for more complex searching.

There are three layers of services in APPA (see Figure 3.4): P2P network, basic services and advanced services.

P2P network services. This layer provides network independence with services that are common to different P2P networks:

- *Peer id assignment:* assigns a unique id to a peer using a specific method, e.g. a combination of super-peer id and counter in a super-peer network.
- *Peer linking:* links a peer to some other peers, e.g. by locating a zone in CAN.
- *Key-based storage and retrieval (KSR):* stores and retrieves a (key, object) pair in the P2P network, e.g. through hashing over all peers in DHT networks or using super-peers in super-peer networks. An important aspect of KSR is that it allows managing data using object semantics (i.e. with KSR it is possible to get and set specific data attributes).
- *Key-based time stamping (KTS):* generates monotonically increasing timestamps which are used for ordering the events occurred in the P2P system.
- *Peer communication:* enables peers to exchange messages (i.e. service calls).

Basic services. This layer provides elementary services for the advanced services using the P2P network layer:

- *Update management service (UMS):* provides high availability for the (key, object) pairs by replicating them over several peers using multiple hash functions [16]. It also deals with updating the stored replicas and also efficient retrieval of current replicas.
- *Communication cost management:* estimates the communication costs for accessing a set of objects that are stored in the P2P network. These costs are computed based on latencies and transfer rates, and they are refreshed according to the dynamic connections and disconnections of nodes.
- *Group management:* allows peers to join an abstract group, become members of the group and send and receive membership notifications. This is similar to group communication systems [38].

Advanced services. This layer provides advanced services for semantically rich data sharing including query processing [14, 15, 17], replication [44, 93], schema management, security, etc. using the basic services.

In our work, we aim to integrate PEERSUM, a new service for managing data summaries over the underlying data. PEERSUM supports query processing with semantic indexes, and allows for approximate query answering using the intentional descriptions of the available summaries.

3.2.2 Model architecture

Given a P2P network, we consider the two following assumptions.

- Each peer p owns some tuples (R_p) in a global, horizontally partitioned relation R.
- Users that are willing to cooperate agree on a Background Knowledge BK, which represents their common perception of the domain.

Thus, here we do not address the problem of semantic heterogeneity among peers, since it is a separate P2P issue on its own. Besides, our work mainly targets collaborative database applications where the participants are supposed to work on "related" data. In such a context, the number of participants is also supposed to be limited, and thus the assumption of a common BK seems not to be a strength constraint. An example of such BK in a medical collaboration is the Systematized Nomenclature of Medicine Clinical Terms (SNOMED CT) [7], which provides a common language that enables a consistent way of capturing, sharing and aggregating health data across specialties and sites of care. On the other hand, our summaries are data structures that respect the original data schemas [126]. Hence, we can assume that the techniques that have been proposed to deal with information integration in P2P systems (e.g. [73], [120]) can be used here to overcome the heterogeneity of both data and summary representations, in the context of heterogeneous data.

Let $G = (V, E)$ be the graph corresponding to a P2P network of size N, where V is the set of nodes (i.e. $|V| = N$), and E is the set of links between nodes. Our ultimate

goal is to maintain a global summary that completely describes the global relation R. However, as stated before, the relation R is horizontally partitioned and distributed among autonomous peers. Hence, the problem can be defined as follows. Given that each peer p_i locally maintains a Local Summary LS_i, we aim to construct the global summary GS_c such that:

$$GS_c = \cup_{i=1}^{N}(LS_i)$$

The local summaries are obtained by integrating the summarization process previously defined into each peer's DBMS. The operator $\cup$ designates the summary merging operation which will be discussed later. Note that GS_c is an approximation of the summary which might be obtained if the global relation R were totally available and summarized under a central coordination.

Once again, the autonomous and dynamic nature of P2P networks imposes additional constraints and makes the convergence to GS_c quite challenging. It is difficult to build and to keep this summary consistent relative to the current data instances it describes. So, the problem can be redefined as follows.

Given the set of materialized local summaries $\{LS_i, 1 \leq i \leq N\}$, we require to build/materialize the set of global summaries $\{GS_j, 1 \leq j \leq N_G\}$ such that:

- $GS_j = \cup_{i=1}^{l}(S_i)$, where S_i is a local or global summary. Here, the latter is defined as being the merging result of, at least, two summaries (i.e. $l \geq 2$).

- The set of materialized local/global summaries (each having its set of partner peers P_{GS_j}), and the set of links ($E_s \subset E$) between nodes belonging to different sets of partner peers (i.e. links connecting different summaries), provide together an approximation of the *virtual* summary GS_c.

$$GS_c \approx (\{LS_i, 1 \leq i \leq N_L\}, \{GS_j, 1 \leq j \leq N_G\}, E_s) \tag{3.1}$$

  N_L is the number of local summaries, which is the number of peers that have not participated to any existing global summary GS_j. While N_G is the number of global summaries built in the network (i.e. $|\cup_{j=1}^{N_G}(P_{GS_j})| + N_L = N$).

- A "good" trade-off should be achieved between the cost of updating the set of materialized summaries and the benefits obtained from exploiting these summaries in query processing.

In APPA, we adopt an incremental mechanism for summary construction. The "*coverage*" of a summary S in the network is defined as being the fraction of peers that own data described by S. This coverage quantifies the convergence of S to the complete summary GS_c, which is obviously characterized by a coverage = 1.

The architecture of our summary model is presented in Figure 3.5. The incremental aspect of the summary construction approach is described as follows. Peers that cooperate are exchanging and merging summaries, in order to build a Global Summary GS_j over their shared data. This summary is characterized by a continuous evolution in term of coverage. In fact, the cooperation between two sets of peers, each having constructed

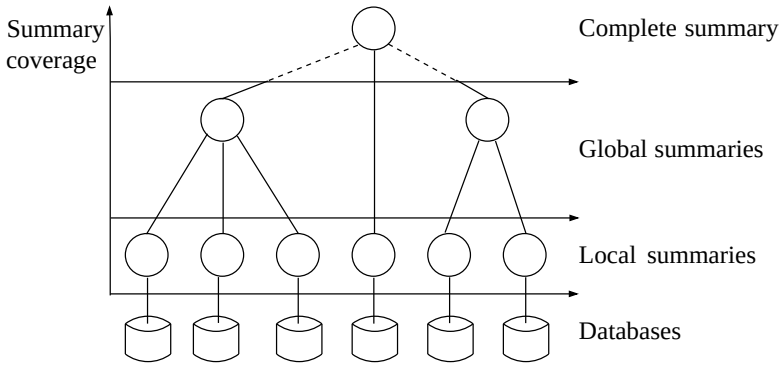

Figure3.5: Model Architecture for APPA

a global summary, will result in a higher-coverage one. Obviously, the "good" trade-off mentioned above in problem statement constraints the level that our incremental mechanism of summary construction is allowed to reach (Figure 3.5). In other terms, it constraints the number N_G of global summaries. A cost analysis model is presented later in Section 3.5.1.

3.2.3 Summary Management in PeerSum

According to our summary model, PEERSUM must address the following requirements:

- Peers cooperate for exchanging and merging summaries into a global summary,
- Peers share a common storage in which the global summary is maintained.

The peer linking and peer communication services of the *APPA*'s P2P network layer allow peers to communicate and exchange messages (through service calls), while cooperating for a global summary construction. Besides, the update management service (UMS) of the basic layer and the Key-based Storage and Retrieval (KSR) service of the P2P network layer, work together to provide a common storage in which a global summary is maintained. This common storage increases the availability of "*P2P data*" (e.g. metadata, indexes, summaries) produced and used by advanced services. The UMS and KSR services manage data based on keys. A key is a data identifier which determines which peer should store the data in the system, e.g. through hashing over all peers in DHT networks or using super-peers for storage and retrieval in super-peer networks. All data operations on the common storage are key-based, i.e. they require a key as parameter.

In the following, we will describe our algorithms for summary construction and maintenance. First, we work in a static context where all the participants remain connected. Then, we address the dynamicity of peers and propose appropriate solutions.

3.2.3.1 Summary Construction

Starting up with a local summary level (see Figure 3.5), we present the algorithm for peer cooperation that allows constructing a global summary GS. We assume that each global

summary is associated with a *Cooperation List* (CL) that provides information about its partner peers. An entry in the cooperation list is composed of two fields. A partner peer identifier *PeerID*, and a *2*-bit freshness value v that provides information about the freshness of the descriptions as well as the availability of the corresponding database.

- value 0 (initial value): the descriptions are fresh relative to the original data,
- value 1: the descriptions are expired and have to be refreshed,
- value 2: the original data are not available. This value will be used while addressing peer volatility in Section 3.2.3.3.

Both the global summary and its cooperation list are considered as "summary data" and are maintained in the common storage, using the UMS and KSR services. The algorithm of summary construction is divided into the three following phases.

Cooperation Request The algorithm starts at an *initiator* peer p_{init}, i.e. the peer who aims to benefit from a new (or existing) global summary. To this end, p_{init} sends a cooperation request message *Coop_Request* to its neighbors. This message contains p_{init}'s address and a given value of TTL. One may think that a large value of TTL allows to obtain directly a high-coverage summary. However, due to the autonomous nature of P2P systems, p_{init} may keep waiting for a very long time without having constructed that global summary. Hence, we choose to limit the value of TTL (e.g. $TTL = 3$), although it results initially in a low summary coverage. The incremental aspect of the construction mechanism will permit to increase the coverage of the newly constructed summary, as long as new contacts with new cooperating peers are encountered.

Each peer p that receives the message, performs the following steps.

1. Checks if the *Coop_Request* message has been already received. In that case, p discards the message. Otherwise, it saves the address of the sender as its *parent*.
2. Decrements TTL by one. If TTL remains positive (i.e. $TTL > 0$), p sends the message to its neighbors, except the *parent*, with the new TTL value.

Thus, the *Coop_Request* message is propagated through a tree, rooted at peer p_{init} and with a maximum depth of TTL (see Figure 3.6).

After forwarding the *Coop_Request* message, a peer p must wait for its neighbor's responses. To avoid waiting some nodes which might never respond to the request, we compute the p's wait time using a cost function based on TTL, network dependent parameters and p's local processing parameters.

Cooperation Response The cooperation response *Coop_Resp* of a peer p has the following structure: $Coop\_Resp = \langle S, PeerIDs, GSKeys \rangle$. S is the summary obtained by merging the summaries received from p's children. $PeerIDs$ is the list of identifiers of peers that have participated to the construction of the global summary (i.e. p's descendants and

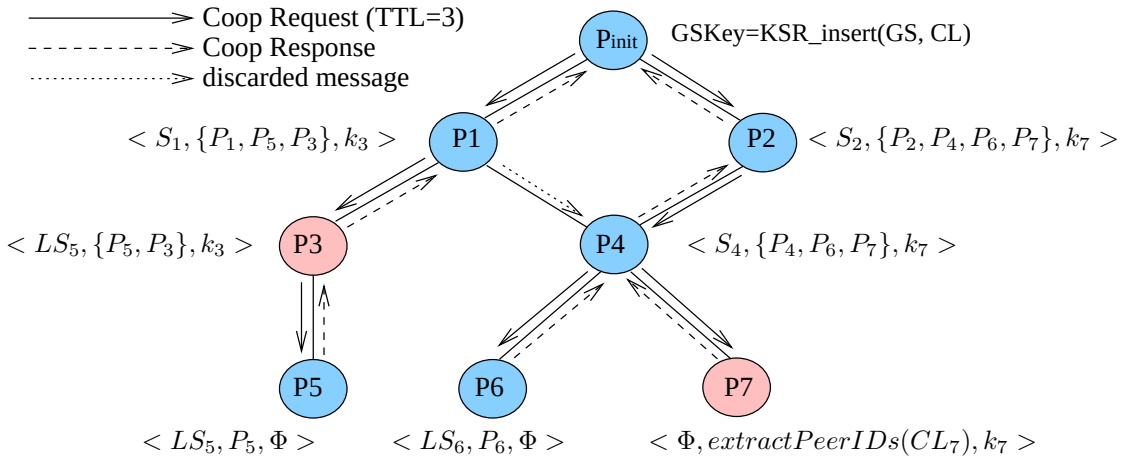

Figure3.6: Global Summary Construction

their partners). $GSKeys$ is the list of keys of existing global summaries that have been discovered at those peers.

We describe now how the p's response is initialized. If p is a partner peer (the pink peers in Figure 3.6), its response will be the following: $Coop\_Resp = \langle \emptyset$, extractPeerIDs(CL), {GSKey}$\rangle$. $GSKey$ is the key of the global summary to which it has already participated. The $extractPeerIDs(CL)$ function returns the identifiers of p's partners which are available in the corresponding cooperation list CL. In that case, we say that p locates at the boundary of two knowledge scopes of two different summaries. Hence, it allows merging them into a higher-coverage one. Otherwise (the blue peers in Figure 3.6), the response of peer p will include its local summary and its identifier, i.e. $Coop_Resp = \langle$p.LS, {p.ID}, $\emptyset\rangle$.

In the case where peer p has received the $Coop\_Request$ message with a TTL value of 1 (e.g. the leaf peers 5, 6 and 7 in Figure 3.6), it sends directly a message containing its cooperation response to its parent. To avoid bottleneck, we can assume that a leaf peer waits a little (random) time before sending its response.

However, if peer p has participated to the propagation of the $Coop\_Request$ message (i.e. $TTL > 1$), it must wait for its children's responses. When a child's $Coop\_Resp$ arrives, peer p merges it with its own response by making the union of $PeerIDs$ and $GSKeys$ lists, and the merging of summaries Ss. Once the wait time expires, peer p sends the final response to its parent.

Summary Data Storage The merging of cooperation responses at intermediate peers continues until reaching the initiator peer p_{init}. When its wait time expires, p_{init} proceeds to store the new summary data, i.e. the new global summary GS and its cooperation list CL, using the KSR service: $GSKey := KSR\_insert(S, CL)$. CL contains each peer identifier obtained in the final $PeerIDs$ list, associated with a freshness value v equal to zero. At the end, p_{init} sends the new key ($GSKey$) to all the participant peers whose identifiers are available in $PeerIDs$. These peers become the GS's partner peers, and

may thus access the new summary data.

In the above algorithm, all the participants are supposed to remain connected and accessible during the summary construction. However, even under static conditions, peers may take more time than expected to respond to the cooperation request. Besides, each peer computes locally its wait time using a cost function, which is based on local parameters of other peers. Thus, the autonomy of peers and the inaccuracy in wait time computation may result in late cooperation responses. This is the case where a peer p receives a response from one of its children, after it has sent its final response to its own parent q. To address this problem, we propose the following solutions.

- Peer p sends directly the late response to the initiator peer p_{init}, which will merge it with the other received responses. Remember that the p_{init}'s address is communicated to all peers along with the *Coop_Request* message. If p_{init} has already stored the new summary data, it may either ignore it or decide to merge it with the summary data in order to increase its coverage. The summary data are accessed using $GSKey$.

- To avoid overloading the initiator peer p_{init} with many late response arrivals, we can suppose that peer p sends directly the late response to its parent q. If the latter is still in its waiting phase, it ordinarily treats it and thus incorporates it to its final response. Otherwise, peer q will send it in its turn to its parent, and so on until reaching a peer that is still waiting for cooperation responses or, in the worst case, reaching peer p_{init}.

- Let us assume that the *Coop_Request* contains an additional *ancestor* field, whose initial value is the p_{init}'s address. Then, each peer will save that field value as its "ancestor" and replace it by the address of its parent, before forwarding the message to its neighbors. Under this assumption, we can suppose that peer p sends directly the late response to its ancestor, let say peer r, instead of sending it to its parent q. This will increase the probability of finding a peer in its waiting phase, as well as to reduce the number of messages that may be incurred by the previous solution.

Note that, only the initiator peer p_{init} can insert the new summary data it has computed. This avoids the conflicts that may arise due to multiple updates. Indeed, all the responses (including the late ones) have to be finally treated by p_{init}. However, it is not about a centralized role. Once the final global summary is built and stored, the mechanism of its updating is fully distributed, as it will be discussed in the next section.

3.2.3.2 Summary Maintenance

An important issue for any indexing technique is to efficiently maintain the indexes against data changes.

For a local summary, it has been demonstrated that the summarization process guarantees an incremental maintenance (using a *push* mode for exchanging data with the

DBMS), while performing with a low complexity. This allows for an online data processing without the need for an overall summary computation.

In this section, we propose a strategy for maintaining the summary data: a global summary GS which has been obtained by merging the local summaries of the set of peers P_{GS}, and its associated cooperation list CL. The objective is to keep GS consistent with the current instances of the local summaries. The latters are supposed to be consistent with the current instances of the original data sources. Our maintenance strategy uses both *push* and *pull* techniques, in order to minimize the number of messages exchanged in the system.

Push: Cooperation List Update Each peer p in P_{GS} is responsible for refreshing its own element in the cooperation list CL. The partner p monitors the modification rate issued on its local summary LS. When LS is considered as enough modified, the peer p sets its freshness value v to 1, through a push message. The value 1 indicates that the local summary version being merged while constructing GS does not correspond any more to the current instance of the database.

An important feature is that the frequency of push messages depends on modifications issued on local summaries, rather than on the underlying databases. It has been demonstrated in [126] that, after a given process time, a summary becomes very stable. As more tuples are processed, the need to adapt the hierarchy decreases and hopefully, once all existing attribute combinations have been processed, incorporating new tuple consists only in sorting it in a tree. A summary modification may be detected by observing the appearance/disappearance of descriptors in the summary intention.

Pull: Summary Update The summary service monitors the fraction of old descriptions in GS, i.e. the number of ones in CL. We may consider that the peer that has been delegated to store CL (by the KSR and UMS services) is in charge of performing this task. Upon each push message sent to update a freshness value, the fraction of ones in CL is checked. If it exceeds a threshold value α, the summary update mechanism will be then triggered. Note that the threshold α is our parameter by which the freshness degree of GS will be controlled. In order to update GS, all the partner peers will be pulled to merge their current local summaries into a GS's version. The algorithm is described as follows.

A summary update message Sum_Update is propagated from a partner to another. This message contains a new summary $NewGS$ (initially empty), and a list of peer identifiers $PeerIDs$ which initially contains the identifiers of all GS's partners (provided by CL). When a partner p receives the Sum_Update message, it first merges $NewGS$ with its local summary and removes its identifier from $PeerIDs$. Then, it sends the message to another partner chosen from $PeerIDs$. If p is the last visited peer (i.e. $PeerIDs$ is empty), it updates the summary data: GS is replaced by the new version $NewGS$, and all the freshness values in CL are reset to zero. This strategy avoids conflicts and guarantees a high availability of the summary data, since only one update

operation is performed by the last visited partner.

Now suppose that a global summary GS_1 has been discovered at peer p while constructing a new global summary GS, as described in Section 3.2.3.1. We have considered that GS_1 is merged to GS, and the identifiers of all its partners are added to the new cooperation list CL. However, we have not taken into account the freshness of GS_1's descriptions. The fraction of ones in CL_1 has been ignored since all the freshness values in CL are initialized to zero. Obviously, this may delay the first execution of the GS's update mechanism. To address this problem, we consider that the *Coop_Resp* contains the cooperation list CL (initially empty), instead of the list of peer identifiers $PeerIDs$. Thus, peer p makes the union of CL and CL_1 rather than only extracting the partner identifiers. For a common partner entry, the freshness value is the binary OR of its freshness values in CL_1 and CL. While for non-partner peers (i.e. peers that participate for the first time to a global summary), new entries containing their identifiers and freshness values of 0 are added.

3.2.3.3 Peer Dynamicity

In P2P systems, another crucial issue is to maintain the data indexes against network changes. Besides the freshness of summary descriptions, the availability of the original data sources should be also taken into account, given the dynamic behavior of peers.

Peer Arrival When a new peer p joins the system, it generally contacts some existing peers to determine the set of its neighbors. If one of these neighbors, peer q, is a partner relative to an existing global summary GS, p becomes a new partner. It gets the GS's key from peer q and adds a new element to the corresponding cooperation list CL. This new element is composed of p's identifier and a freshness value v equal to one. Recall that the value 1 indicates the need of pulling peer p to get new data descriptions. In the case where p connects between two partners that belong to two different summaries, it allows merging them in a higher-coverage one, as discussed in Section 3.2.3.1.

Peer Departure When a partner peer p decides to leave the system, it first sets its freshness value v to 2 in the cooperation list. This value reminds the participation of the disconnected peer p to the corresponding global summary GS, but also indicates the unavailability of the original data. There are two alternatives to deal with such a freshness value. The first one consists in keeping the p's data descriptions if they were fresh before p leaves the system (i.e. $v = 1$). Thus, these descriptions may be exploited while approximately answering a query. The second alternative consists in considering the p's data descriptions as expired, since the original data are not available. Thus, a partner departure will accelerate the summary update initiating. In the rest of this paper, we adopt the second alternative and consider only a 1-bit freshness value v: a value 0 to indicate the freshness of data descriptions, and a value 1 to indicate either their expiration or their unavailability.

If peer p failed, however, it could not notify its partners by its departure. In that case, its data descriptions will remain in the global summary until a new summary update is executed. The update algorithm does not require the participation of a disconnected peer. The global summary GS is reconstructed, and descriptions of unavailable data will be then omitted.

3.2.4 Discussion

The first phase of our work has mainly studied the feasibility of integrating a data summarization process into an existing P2P data management system. The summarization process that has been adopted exhibits salient features such as robustness, intelligibility and scalability. In the context of APPA, we have proposed solutions for summary creation/maintenance, based on APPA's services. Each peer maintains a local summary of its database, and cooperates with other peers to maintain a global summary, which covers an increasing number of available data sources. The summary update mechanism is efficient since it optimizes the number of exchanged messages through combining both push and pull techniques. Besides, the trade-off between the generated traffic overhead and the freshness of summary descriptions can be tuned using a system parameter. The performance evaluation of our proposal will be discussed later in this chapter.

However, here we indicate the following limitation of our summary model. The summary construction mechanism we have considered is incremental: a global summary is in continuous evolution in term of *coverage*. This might be impractical in large-scale P2P systems and under bandwidth consumption constraints. The freshness of summary descriptions may be sacrificed at the expense of reducing the number of exchanged messages (i.e. allow a large fraction of ones in the cooperation list). Besides, this incremental mechanism supposes that any contact between any two peers will result in a cooperation between them. Thus, their respective global summaries are merged into a higher-coverage one. However, such an assumption is not relevant when the number of participants is high. Even when peer are working on related data in a given application, they generally tend to work on groups (i.e. group locality [13]).

3.3 Summary Model for Hierarchical P2P Networks

In the previous summary model proposed for APPA, we have worked under the following two assumptions:

- The Global Summary GS in Figure 3.5 is characterized by a continuous evolution in terms of coverage.
- The APPA's services provide a common storage in which the global summaries are maintained.

In the second phase of our work, we relax these two assumptions for a closer study to the problem of distributing summaries within the network. So, in this section, we discuss

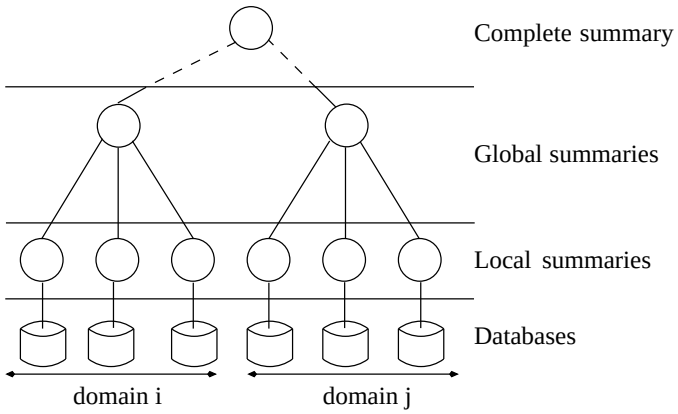

Figure3.7: Model Architecture for Hierarchical P2P Networks

a new summary model and present appropriate summary management techniques.

3.3.1 Model Architecture

Data indexes are maintained in P2P systems using one of the following approaches. A *centralized* approach maintains a global index over all the data shared in the network, and thus provides a centralized-search facility. A *hybrid decentralized* approach distributes indexes among some specialized nodes (e.g. supernodes), while a *pure decentralized* approach distributes indexes among all the participants in the network (e.g. structured DHTs, Routing Indices). Each of these approaches provides a different trade-off between the cost of maintaining the indexes and the benefits obtained for queries. In our work, we have adopted the second approach since it is the only one that exploits peer heterogeneity, which is a central key to allow P2P systems scaling up without compromising their decentralized nature [133]. Besides, some argue that the super-peer networks [35] are the most suited for content-based search [68, 149] since their inherent hierarchical structure allows to build a similar hierarchy over the shared data and metadata.

Let us examine the architecture of our summary model which is presented in Figure 3.7. It is clear that the hierarchy among global summaries follows the hierarchical structure of the underlying network (two summary levels). The network is organized into *domains*, where a domain is defined as being the set of a supernode and its associated leaf nodes. In a given domain, peers cooperate to maintain a global summary over their shared data. The set of global materialized summaries and links between the corresponding domains, provide an approximation of the summary GS_c (Equation 3.1).

Obviously, an important issue here is how to organize the network into domains, i.e. how the superpeers are selected and other peers are grouped around them in a fully decentralized manner. The number and the size of domains have a direct impact on the cost of summary maintenance. Thus, in the following, we first discuss the issue of network self-organization. Then, we discuss summary management.

3.3.2 Network Self-Organization

Intuitively, the term "self-organization" describes the ability of a P2P network to organize its participants into a cooperative framework, without the need of external intervention or control. For our purposes, we will understand *self-organization* as the capability of partitioning the network into domains in order to optimally distribute data summaries, without using global information or restricting peer autonomy. In this section, we first give the rationale behind our solution, and then provide the algorithm for network self-organization.

3.3.2.1 Rationale

In [108], the authors have addressed the problem of maintaining distributed indexes in a P2P network. They have proposed a two-tier structure. Nodes are partitioned into independent domains, and nodes within a domain build a global index over their shared data. However, the authors have assumed that there exists a fixed number of domains k, and a node is assigned to one of these domains at random. As a first issue, they have studied the optimal number of domains in order to minimize the total cost of queries and index maintenance. It has been shown that this number is a function of the the total number of nodes in the network. Then, they have addressed an important issue which is enabling a dynamic, self-tuning index that approaches the optimal index configuration even as the number of nodes varies in the systems.

In our work, we do not suppose that nodes are assigned to domains at random. Instead, we are interested in how domains are created and nodes are assigned to them. Several works have proposed diverse solutions for network organization (also known by network clustering). This will be discussed in chapter 3. Here, for the concreteness of our distributed summaries proposal, we present a primitive function that enables to partition the network around high-connectivity nodes.

A number of recent studies [133], [101] have shown that the existing complex networks have common characteristics, including power law degree distributions, small diameter, etc. In these networks, called *power-law networks*, most nodes have few links and a tiny number of hubs have a large number of links. More specifically, the fraction of nodes with k links is proportional to $k^{-\beta}$, where β is called the exponent of the distribution.

Our solution for network organization is completely decentralized, and mainly exploits the power-law distribution of node degrees. It does not rely on any global information, however, it uses local information by considering that a node has only to know about the entities and the connectedness of its neighbors. The key idea is that random walks in power-law networks naturally gravitate toward the high-degree nodes. A *random walk* is a technique proposed by [136] to replace flooding in unstructured P2P systems. At each step, a query message is forwarded to a randomly chosen neighbor until sufficient responses to the query are found. Although it makes better utilization of the P2P network than flooding, a random walk is essentially a blind search in that it does not take into account any indication of how likely it is the chosen node will have responses for the query. Adamic *et* al. [81] addressed this problem and showed that a better scaling is achieved

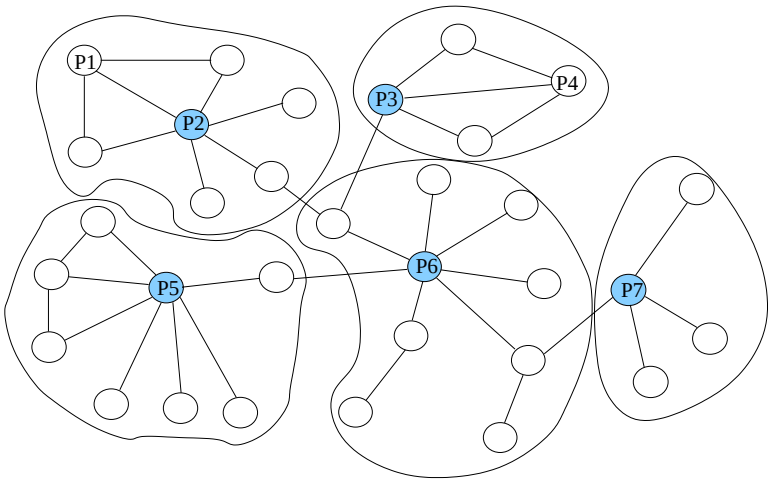

Figure3.8: Summary Peer Identification

by intentionally choosing high degree nodes. We will refer to this routing technique as "*selective walk*".

In our work, we aim to identify the superpeers, which will be referred later as *summary peers*, in a power law network with an exponent β and maximum degree k_{max}. Summary peers are defined as being high degree peers, which will serve as *centers of summary-attraction*. Using a selective walk which naturally and rapidly gravitates toward high degree nodes, a peer p finds the nearest summary peer SP to which it sends a duplicate of its local summary LS. The set of peers that discover the same summary peer SP are grouped around it and form a domain. These peers become *partners* relative to a global summary GS obtained by merging their local summaries. In [104], any node with degree k is considered as a high-degree node if $k \geq k_{max}/2$. However, k_{max} scales like $O(N^{1/\beta})$ [145] and is a global information. In the next section, we propose an IS_SUMPEER function that is executed locally at each peer to decide whether it is a summary peer or not, using minimum local information.

3.3.2.2 Algorithm

For our purpose, we have extracted a general model of a high-degree node in a power law network. To illustrate, consider the network of figure 3.8. The IS_SUMPEER function consists in matching this model with each node of the network. Algorithm 1 shows the steps involved in making this matching. First, we check if the current peer p is among the highest-degree peers in its neighborhood. In other words, we check if the degree k of peer p is greater than the median value of the set of its neighbor's degrees (i.e. $|subset\_inf| > |subset\_sup|$). This first condition proposes as candidates peers p_1, p_2, p_3, p_4, p_5, p_6, p_7. Then, we verify if the local maximum degree k_{pmax} in p's neighborhood does not exceed $2 \cdot k$. This condition allows to exclude peer p_1 whose degree is slightly greater than the median value, but significantly less (e.g. factor of less than 0.5) than the maximum value of its neighbor's degrees. Finally, we examine if k is larger than the

mean value of neighbor's degrees by a constant ct. This condition makes a difference in the matching result when the neighbor's degrees follow an asymmetric distribution with a positive skew, i.e. there are a small number of very large degrees. In that case, the mean value is greater than the median. The constant ct permits to tune the selectivity of the matching function. Larger is ct, less is the total number of summary peers. For instance, a value 1 of ct keeps peer p_4 as candidate, while a value 2 excludes it. Indeed, peer p is considered as a summary peer if the above three conditions are satisfied simultaneously. In our example, the summary peers finally obtained (for $ct = 2$) are p_2, p_3, p_5, p_6, p_7.

Algorithm 1 $Is\_SumPeer$

1: **function** $Is\_SumPeer(k, NL)$
2: k is the degree of the current peer p, and NL is the Neighboring List that contains the identifiers of p's neighbors and their degrees.
3: $subset\_inf := p_i \; \forall \; 1 \leq i \leq |NL|$ such that $k_i < k$
4: $subset\_sup := p_i \; \forall \; 1 \leq i \leq |NL|$ such that $k_i > k$
5: $k_{pmax} :=$ **max** $(k_i), \forall \; 1 \leq i \leq |NL|$
6: $k_{mean} :=$ **mean** $(k_i), \forall \; 1 \leq i \leq |NL|$
7: **if** $(|subset\_inf| > |subset\_sup|)$ **and** $(k > k_{pmax}/2)$ **and** $(k > ct \cdot k_{mean})$ **then**
8: $Is\_SumPeer :=$ true
9: **else** $Is\_SumPeer :=$ false
10: **end if**
11: **end function**

Using the Brite topology generator [2], we simulate N-node P2P networks whose node degrees follow a power law distribution with a mean value of 4. Figure 3.9 shows the number of summary peers in function of the total number of peers N. We see that this number is proportional to $\sqrt{n}$ for network sizes smaller than 1024, and is proportional to n for larger networks. Since our domains are formed around the summary peers, thus figure 3.9 gives directly the number d of domains obtained in the network. The shown results are similar to those found in [108]. It has been proved, theoretically and by simulation, that the optimal number of domains is in $O(\sqrt{n})$ for total-lookup queries, and in $O(n)$ for partial-lookup queries. Recall that a *total-lookup* query requires all results that are available in the system, whereas a *partial-lookup* query requires any m results, for some constant m. Total-lookup query are very difficult and costly in large P2P networks, and thus all queries, in practice, are processed as being partial-lookup queries.

We conclude that using our primitive degree-based function, the network can be partitioned into an optimal number of domains: for total-lookup queries in small-sized networks, and for partial-lookup queries in larger-sized networks. Certainly, this work is not complete and requires further examination and discussion. But, it may be considered as the initial phase of other works like [108]. In other terms, it constitutes the beforehand organization of a P2P network, which should be then associated with efficient mechanisms for maintaining such organization against dynamicity.

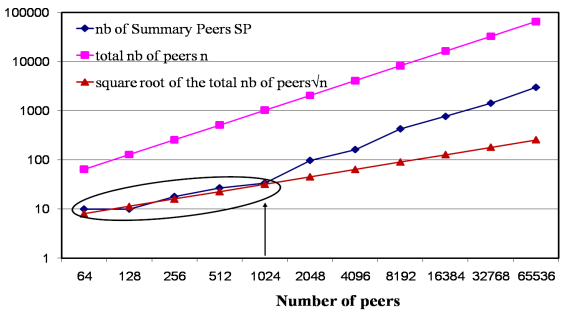

Figure3.9: Number of summary peers vs. number of peers

3.3.3 Summary Management

As in our summary model for APPA, we assume that each global summary GS is associated with a *Cooperation List* (CL) (see Section 3.2.3.1). Algorithm 2 shows the messages exchanged between peers in order to build a global summary GS. A summary peer SP broadcasts a SUMPEER message that contains its identifier, to indicate its ability to host summaries. Since SP is supposed to have high connectivity, a small value of TTL (Time-To-Live) is sufficient to cover a large number of peers (e.g. $TTL = 2$). The message contains also a hop value h, initialized to 0, which is used to compute the distances between SP and the visited peers. A peer p who received a first SUMPEER message, maintains information about the corresponding summary peer SP (i.e. Line 16). Then, p sends to SP a LOCALSUM message that contains its local summary LS, and thus becomes a partner peer in the SP's domain. Upon receiving this last message, SP merges LS to its current global summary GS, and adds a new element in the cooperation list.

However, a peer p who is already a partner may receive a new SUMPEER message. In such a case, only if the new summary peer is nearer than the old one (based on latency), it chooses to drop its old partnership through a DROP message (i.e. Line 14), and it proceeds to participate to a new domain. We now suppose that a peer p does not belong to any domain (i.e. p is not a partner peer), and wants to participate to a global summary construction. Using a selective walk, it can rapidly find a summary peer SP (i.e. FIND message). The information about SP, which is maintained at each of its partners, makes the selective walk even shorter. Once a partner or a summary peer is reached, the FIND message is stopped (i.e. Line 32).

The summary maintenance strategy is similar to that adopted in APPA (Section 3.2.3.2). The main difference is that the summary peers will be now in charge of storing the global summaries, and monitoring the freshness of their descriptions in order to initiate the update mechanism whenever it is required.

3.3.4 Peer Dynamicity

According to our summary model, a peer may be either a summary peer or a partner peer. The departure/arrival of a partner peer is treated as described in Section 3.2.3.3.

Algorithm 2 *Global Summary Construction*

```
1: // Definition of different types of messages
2: SUMPEER=⟨sender⟩ ⟨id, h, TTL⟩; FIND=⟨sender⟩ ⟨h, TTL⟩
3: LOCALSUM=⟨sender⟩ ⟨LS⟩; DROP=⟨sender⟩ ⟨⟩
4: // Treatment of messages
5: Switch msg.type
6: // Receiving information about a summary peer
7: Case (SumPeer):
8:     msg.h++; msg.TTL−−
9:     if (this.SumPeer=null) or (this.SumPeer.h¿msg.h) then
10:        if (this.IsPartner) then
11:            Send DROP message to this.SumPeer.id
12:        end if
13:        this.SumPeer:= ⟨msg.id, msg.h⟩
14:        LOCALSUM:= new msg (this.LS); Send LOCALSUM to msg.sender
15:        IsPartner:= True
16:    end if
17:    if msg.TTL > 0 then
18:        Send msg to all neighbors
19:    end if
20: end Case
21: // Searching for a summary peer
22: Case (Find):
23:    msg.TTL−−; msg.h++
24:    if (this.Is_SumPeer) then
25:        PEERSUM:= new msg (this.id, msg.h, 1); Send PEERSUM to msg.sender
26:    else
27:        if (this.SumPeer ≠ null) then
28:            PEERSUM:= new msg (this.SumPeer.id, (msg.h + this.SumPeer.h), 1)
29:            Send PEERSUM to msg.sender
30:        else
31:            if (msg.TTL > 0) then
32:                p′:= highest degree peer in N(p); Send msg to p′
33:            end if
34:        end if
35:    end if
36: end Case
37: // arrival of a new partner
38: Case (LocalSum):
39:    CoopList.add (msg.sender, 0); GlobalSum:= merge (GlobalSum, msg.LS)
40: end Case
41: // departure of a partner
42: Case (Drop):
43:    CoopList.remove (msg.sender)
44: end Case
```

Capacity level	percentage of nodes
$1x$	20%
$10x$	45%
$100x$	30%
$1000x$	4.9%
$10000x$	0.1%

Table3.3: Gnutella-like node capacity distributions

However, here we discuss the effects of the dynamic behavior of summary peers.

In section 3.3.2, we have presented our IS_SUMPEER function that is executed at each peer to decide whether it is a summary peer or not. This function is based on node connectivity, and thus variations of node degrees may incur modifications in the function results. Hence, we can suppose that a peer periodically executes the IS_SUMPEER function.

However, we believe that the results of the IS_SUMPEER function do not change frequently thanks to two characteristics of power law P2P networks. First, connections tend to be formed preferentially since peers tend to discover high-degree nodes in the network overlay [133]. Second, although the nodes join and leave the network with a high rate, we suppose that each node leave is very probably accompanied by a new node join such the the total number of nodes remains the same. Thus, the cases in which a summary peer becomes an ordinary peer, or an ordinary peer is submitted to a significant degree variation rarely occur. In the latter case, when a new highly-available peer attracts many peer connections and becomes a summary peer, it simply diffuses this information as described in section 3.3.3, and a new domain starts to appear around it.

Now, when a summary peer SP decides to leave the system, it sends a release message to all its partners using the cooperation list. Upon receiving such a message, a partner p makes a selective walk to find a new summary peer. However, if SP failed, it could not notify its partners. A partner p who has tried to send push or query messages to SP will detect its departure and thus search for a new one.

3.3.5 Capacity-based Summary Distribution

Thus far, we have considered a centralized approach for storing a global summary in a given domain. Each summary peer SP is in charge of storing the global summary GS of its domain. However, this implies that each query posed by a partner p will be sent to SP, which may be then overloaded. Here we give a simple solution to distribute GS, and thus the query load, based on node capacity. In [155], capacities are assigned to nodes based on a distribution that is derived from the measured bandwidth distributions for Gnutella, as reported by Saroiu *et al* [133]. The capacity distribution has five levels of capacity, each separated by an order of magnitude (Table 3.3). A node i is modeled as possessing a capacity C_i, which represents the number of queries that it can process per unit time. In addition to its capacity, each node i is assigned a query generation rate q_i, which is the

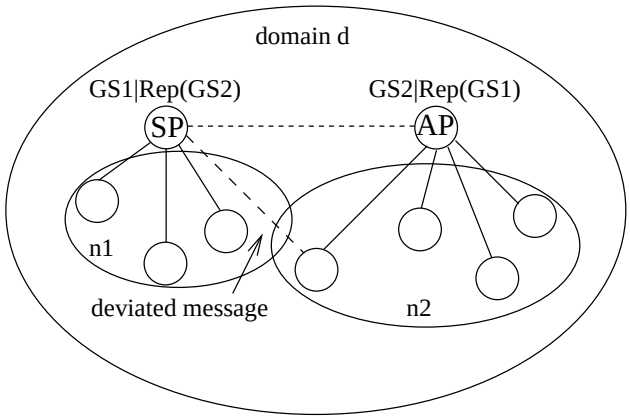

Figure3.10: capacity-based summary distribution

number of queries that it generates per unit time. Therefore, a node i is represented by: $i = (C_i, q_i)$.

Now, we suppose that each summary peer SP delegates an *Assistant Peer* AP, which is the highest-capacity neighbor that belongs to the same domain. The summary peer SP accepts n_1 peers as partners, among the n_d peers of its domain d such that: $\sum_{i=1}^{n_1} q_i < C_{SP}$. If n_1 is greater than n_d, SP is considered as a high-capacity peer, and thus will host the global summary of the entire domain. Otherwise, we adopt a *deviation* method to distribute the charge between the summary peer and its assistant. When SP receives a LOCALSUM message (see algorithm 2) and it cannot support the query generation rate of the sender, it deviates the message to the assistant AP. In this case, the summary peer SP serves as a "beacon". Using a selective walk, a peer p rapidly finds SP, which will send him to the assistant AP in order to handle a duplicate of its local summary. However, the global summary GS of the domain d is divided into two summaries: GS_1 and GS_2, maintained by SP and AP respectively. Thus, to allow each partner peer exploiting both summaries, we replicate GS_2 at SP and GS_1 at AP, as shown by Figure 3.10.

We note that a modification of the network topology can improve our network organization scheme. In the above solution, the capacity of a summary peer SP may be less than the capacity of its assistant AP. In this case, a neighboring link to SP, on which a LOCALSUM message has been deviated, will be dropped and replaced by a new link to AP. Therefore, the network topology will be adapted to ensure a congruence between summary peers and high-capacity peers (i.e. after a given time, AP becomes a high-degree peer in its locality). This is not a new idea: other works have addressed this problem. For instance, in [155], the authors have proposed a topology adaptation algorithm which ensures that well-connected peers have good capacities. Obviously, such a topology adaptation allows reducing the number of deviated messages in a given domain.

3.3.6 Discussion

This second phase of our work has proposed a model for managing summaries in hierarchical P2P networks. The model provides two levels of materialized summaries. Nodes are organized into domains (clusters). Each node maintains a local summary of its database, and nodes within a domain cooperate to maintain a global summary over their shared data. This model can be generalized to n-summary levels. However, this is different from the model proposed for APPA in Section 3.2.2. In the latter, the construction of a given global summary implies that all global summaries which have been merged to it, are no longer materialized in the network. Hence, each peer only knows about one materialized global summary whose network coverage is incrementally increased.

Here, by n-summary levels, we mean that a materialized hierarchy between global summaries (of different coverages) is maintained in the network. For example, we may consider that summary peers exchange their global summaries and maintain a third summary level, which does not result in removing the underlying level from the model. The first summary level allows to efficiently retrieve relevant tuples within peers' databases. The second summary level guides the query propagation within domains, while the third one may assist the query while traveling from a domain to another.

3.4 Query Processing

We describe now how a query Q, posed at a peer p, is processed. Our approach consists in querying at first the global summary GS that is available to peer p. In the APPA summary model, the key of the global summary allows peer p to access GS. While in the hierarchical model, peer p first sends Q to the summary peer SP of its domain, which then proceeds to query the available global summary GS. the type of the query, precise or flexible (when using linguistic terms), and the nature of the returned results allow to distinguish four cases of summary querying. These cases are illustrated when presenting the two phases of the summary querying mechanism: 1) query reformulation and 2) query evaluation.

3.4.1 Query Reformulation

First, a precise query Q must be rewritten into a flexible query Q^* in order to be handled by the summary querying process. For instance, consider the following query Q on the Patient relation in Table 3.1:

```
select age from Patient where sex = ‘‘female’’
and BMI < 19 and disease = ‘‘anorexia’’
```

This phase replaces the original value of each selection predicate by the corresponding descriptors defined in the Background Knowledge (BK). Therefore, the above query is transformed to Q^*:

```
select age from Patient where sex = ‘‘female’’ and
BMI in {underweight,normal} and disease = ‘‘anorexia’’
```

The mechanism that assists this query rewriting phase is already defined and used in the mapping service of the summarization process.

Let QS (resp.QS^*) be the *Query Scope* of query Q (resp.Q^*) in the domain, that is, the set of peers that should be visited to answer the query. Obviously, the query reformulation phase may induce false positives in query results. To illustrate, a patient having a BMI value of 20 could be returned as an answer to the query Q^*, while the selection predicate on the attribute `BMI` of the original query Q is not satisfied. However, false negatives cannot occur, which is expressed by the following inclusion: $QS \subseteq QS^*$.

In the rest of this paper, we suppose that a user query is directly formulated using descriptors defined in the BK (i.e. $Q = Q^*$). As we discussed in the introduction of this work, a doctor that participates to a given medical collaboration, may ask queries like "the *age* of *female* patients diagnosed with *anorexia* and having an *underweight* or *normal BMI*". Thus, we eliminate potential false positives that may result from query rewriting. Note that an interface has been defined to allow users to formulate their queries, using linguistic terms, without having knowledge of the pseudo-code language used in the querying mechanism. To illustrate the formulation of queries, recall the following notations:

- R: the schema of the summarized relation
- A: the set of attributes of relation R ($|A| = n$)
- A_i: The i^{th} attribute of relation R
- t: a tuple of the relation R
- z: a summary node of the summary hierarchy S built over the relation R

In the query Q, we distinguish the set X of attributes whose values x are specified by the query predicates, from the set Y of attributes on which the query is projected. In other terms, the general form of query Q is given by: *Select* Y *from* R *where* $X = x$.

For each attribute $A_i \in X$, a "*required characteristic*" is defined as being a linguistic term that appears in the query for attribute A_i. The set C_i of these required characteristics is given by: $C_i = \{d_i^1, d_i^2, \ldots, d_i^m\}$. By extension, the characterization of query Q is the set of k characterizations defined on the attributes of set X: $C = \{C_1, \ldots, C_k\}$, where $k = |X|$. In our example: $X = \{sex, BMI, disease\}$, $Y = \{age\}$, $C_{sex} = \{female\}$, $C_{BMI} = \{underweight, normal\}$, $C_{disease} = \{anorexia\}$, and $C = \{C_{sex}, C_{BMI}, C_{disease}\}$.

Finally, a logical proposition P is associated to a query characterization such that:

$$P = \bigwedge_{i=1}^{k} (\bigvee_{j=1}^{m_i} d_i^j)$$

Intuitively, the presence of multiple required characteristics on a given attribute denotes an alternative, and thus the intra-attribute logical connector is disjunctive (i.e. $C_i = \bigvee_{j=1}^{m_i} d_i^j$). However, the presence of simultaneous characterizations of different attributes implies that the inter-attribute connector is conjunctive (i.e. $P = \bigwedge_{i=1}^{k}(C_i)$). In our example, the query Q is transformed to the proposition P= *(female) AND (underweight OR normal) AND (anorexia)*. In fact, the proposition P represents the canonical form of the query, which will be evaluated by the summary querying process.

3.4.2 Query Evaluation

First of all, to eliminate any ambiguity, it is worthy to recall that all along our work, we designate by (local/global) summary a hierarchy of summary nodes whose structure has been described in Section 3.1.4.

This phase deals with matching the set of summary nodes organized in the hierarchy S, against the query Q. A valuation function has been defined to valuate the corresponding proposition P in the context of a summary node z, and thus to determine if z is considered as a result. Then, a selection algorithm performs a fast exploration of the hierarchy and returns the set Z_Q of most abstract summary nodes that satisfy the query. For more details see [146].

Once Z_Q determined, the evaluation process is able to achieve two distinct tasks: 1) Summary answering, and 2) Peer localization.

3.4.2.1 Approximate Answering

A distinctive feature of our approach is that a query can be processed entirely in the summary domain. An approximate answer can be provided from summary descriptions, without having to access original database records.

In the result set Z_Q, distinct summary nodes may answer the query in different manners. For instance, the set Z_Q contains summary nodes in which the attribute BMI is described either by *underweight*, or *normal*, or even by the both descriptors. A classification task has been defined in order to regroup the summary nodes in Z_Q according to their characteristics vis-a-vis the query: summary nodes that have the same required characteristics on all predicates (e.g. *sex*, *BMI* and *disease*) form a class. The aggregation in a given class is a union of descriptors: for each attribute in the selection set Y (i.e. *age*), the querying process supplies a set of descriptors which characterize the summary nodes that answer the query through the same interpretation [146]. For example, according to Table 3.2, the output set obtained for the different classes found in z_Q is:

- $\{female, underweight, anorexia\} \Rightarrow age = \{adolescent, young\ adult\}$
- $\{female, normal, anorexia\} \Rightarrow age = \{adolescent\}$

In other words, *all* female patients diagnosed with *anorexia* and having an *underweight* or *normal* BMI are *adolescent* or *young* girls. Moreover, the information provided by the

tuple count columns may further indicate that almost all of these girls are *adolescent* (i.e. a value of 2.5 for *adolescent*, and a value of 0.5 for *young adult*).

3.4.2.2 Peer Localization

In centralized environments, returning exact answers to the flexible query Q is simply an extension of the mechanism of returning approximate answers. The extent R_z of a summary node z (see Definition 1) provides the original records that are described by that summary intent. Thus, such a functionality only requires to connect to the underlying database in order to retrieve data.

However, in P2P systems, a summary node may describe tuples that are highly distributed in the network. Therefore, the query Q should be first forwarded to the relevant peers whose databases contain the result tuples. In Definition 3, we have added the *Peer-extent* dimension to a summary node structure, in order to provide the set of peers P_z having data described by its intent.

Here, we can assume that in a given local summary, the extents of its summary nodes are maintained to be able to retrieve raw data from the underlying database. While in a global summary this information is omitted and only the *Peer-extents* of its summary nodes are maintained to be able to reach the relevant peers in the distributed network.

Based on the *Peer-extents* of summary nodes in the set Z_Q, we can define the set P_Q of relevant peers as follows: $P_Q = \{\cup_{z \in Z_Q} P_z\}$. The query Q is directly propagated to these relevant peers. However, the efficiency of this query routing depends on the completeness and the freshness of summaries, since stale answers may occur in query results. We define a *False Positive* as the case in which a peer p belongs to P_Q and there is actually no data in the p source that satisfies Q (i.e. $p \notin QS$). A *False Negative* is the reverse case in which a p does not belong to P_Q, whereas there exists at least one tuple in the p data source that satisfies Q (i.e. $p \in QS$).

In the case where exact answers are required, suppose now that processing a query Q in a given domain d_i returns C_i results, while the user requires C_t results. We note that, if C_t is less than the total number of results available in the network, Q is said to be a *partial-lookup* query. Otherwise, it is a *total-lookup* query. Obviously, when C_i is less than C_t, the query should be propagated to other domains. To this end, we adopt the following variation of the flooding mechanism.

Let P_i the subset of peers that have answered the query Q in the domain d_i: $|P_i| = (1 - FP) \cdot |P_Q|$, where FP is the fraction of false positives in query results. The query hit in the domain is given by: $(|P_i| \,/\, |d_i|)$. As shown by many studies, the existing P2P networks have small-world features [13]. In such a context, users tend to work in groups. A group of users, although not always located in geographical proximity, tends to use the same set of resources (i.e. *group locality* property). Thus, we assume that the probability of finding answers to query Q in the neighborhood of a relevant peer in P_i, is very high since results are supposed to be *nearby*. This probability is also high in the neighborhood of the originator peer p since some of its neighbors may be interested in the same data, and thus have cached answers to similar queries. Such assumptions are even more relevant

in the context of interest-based clustered networks. Therefore, the summary peer SP_i of domain d_i sends a flooding request to each peer in P_i as well as to peer p. Upon receiving this request, each of those peers sends the query to its neighbors that do not belong to its domain, with a limited value of TTL. Once a new domain is reached or TTL becomes zero, the query is stopped. Besides, the summary peer SP sends the request to the set of summary peers it knows in the system. This will accelerate covering a large number of domains. In each visited domain, the query is processed as described above. When the number of query results becomes sufficient (i.e. larger than C_t), or the network is entirely covered, the query routing is terminated.

3.5 Performance evaluation

In this section, we devise a simple model for the summary management cost. Then, we evaluate our model by simulation using the BRITE topology generator and SimJava.

3.5.1 Cost Model

A critical issue in summary management is to trade off the summary updating cost against the benefits obtained for queries.

3.5.1.1 Summary Update Cost

Here, our first undertaking is to optimize the update cost while taking into account *query accuracy*. In the next section, we discuss query accuracy which is measured in terms of the percentage of false positives and false negatives in query results. The cost of updating summaries is divided into: usage of peer resources, i.e. time cost and storage cost, and the traffic overhead generated in the network.

Time Cost A unique feature of SaintEtiQ is that the changes in the database are reflected through an incremental maintenance of the summary hierarchy. The time complexity of the summarization process is in $O(K)$ where K is the number of cells to be incorporated in that hierarchy [126]. For a global summary update, we are concerned with the complexity of merging summaries. The Merging method that has been proposed is based on the SaintEtiQ engine. This method consists in incorporating the leaves L_z of a given summary hierarchy S_1 into an another S_2, using the same algorithm described by the SaintEtiQ summarization service (referenced in Section 3.1.3.2). It has been proved that the complexity C_{M12} of the Merging(S_1, S_2) process is constant w.r.t the number of tuples [94]. More precisely, C_{M12} depends on the maximum number of leaves of S_1 to incorporate into S_2. However, the number of leaves in a summary hierarchy is not an issue because it can be adjusted by the user according to the desired precision. A detailed Background Knowledge (BK) will lead to a greater precision in summary description, with the natural consequence of a larger summary. Moreover, the hierarchy is constructed in a

top-down approach and it is possible to set the summarization process so that the leaves have any desired precision.

Storage Cost We denote by k the average size of a summary node z. In the average-case assumption, there are $\sum_{i=0}^{d} B^i = (B^{d+1} - 1)/(B - 1)$ nodes in a B-arity tree with d, the average depth of the hierarchy. Thus the average space requirement is given by: $C_m = k.(B^{d+1} - 1)/(B - 1)$. Based on real tests, $k = 512$ bytes gives a rough estimation of the space required for each summary. An important issue is that the size of the hierarchy is quite related to its stabilization (i.e. B and d). As more cells are processed, the need to adapt the hierarchy decreases and incorporating a new cell may consist only in sorting a tree. Hence, the structure of the hierarchy remains stable and no additional space is required. On the other hand, when we merge two hierarchies S_1 and S_2 having sizes of C_{m1} and C_{m2} respectively, the size of the resultant hierarchy is always in the order of the $max\,(C_{m1}, C_{m2})$. However, the size of a summary hierarchy is limited to a maximum value which corresponds to a maximum number of leaves that cover all the possible combinations of the BK descriptors. Thus, storing the global summary at the summary peer is not a strength constraint.

According to the above discussion, the usage of peer resources is optimized by the summarization process itself, and the distribution of summary merging while updating a global summary. Thus, we restrict now our focus to the traffic overhead generated in the P2P network.

Network Traffic Recall that there are two types of exchanged messages: *push* and *update* messages. Let local summaries have an average lifetime of L seconds in a given global summary. Once L expired, the node sends a (push) message to update its freshness value v in the cooperation list CL. The summary update algorithm is then initiated whenever the following condition is satisfied:

$$\sum_{v \in CL} v/|CL| \geq \alpha$$

where α is a threshold that represents the ratio of old descriptions tolerated in the global summary. While updating, only one message is propagated among all partner peers until the new global summary version is stored at the summary peer SP. Let F_{rec} be the reconciliation frequency. The update cost is:

$$C_{up} = 1/L + F_{rec} \text{messages per node per second} \tag{3.2}$$

In this expression, $1/L$ represents the number of push messages which depends either on the modification rate issued on local summaries or the connection/disconnection rate of peers in the system. Higher is the rate, lower is the lifetime L, and thus a large number of push messages are entailed in the system. F_{rec} represents the number of update messages which depends on the value of α. This threshold is our system parameter that provides a trade-off between the cost of summary updating and query accuracy. If α is large, the

update cost is low since a low frequency of update is required, but query results may be less accurate due both to false positives stemming from the descriptions of non existent data, and to false negatives due to the loss of relevant data descriptions whereas they are available in the system. If α is small, the update cost is high but there are few query results that refer to data no longer in the system, and nearly all available results are returned by the query.

3.5.1.2 Query Cost

When a query Q is posed at a peer p, it is first matched against the global summary available at the summary peer SP of its domain, to determine the set of relevant peers P_Q. Then, Q is directly propagated to those peers. The query cost in a domain d is given by:

$$C_d = (1 + |P_Q| + (1 - FP) \cdot |P_Q|) \;\; messages,$$

where $(1 - FP) \cdot |P_Q|$ represents the query responses messages (i.e. query hit in the domain).

Here we note that, the cooperation list CL associated with a global summary provides information about the relevance of each database description. Thus, it gives more flexibility in tuning the *recall/precision* trade-off of the query answers in domain d. The set of all partner peers P_H in CL can be divided into two subsets: $P_{old} = \{p \in P_H \mid p.v = 1\}$, the set of peers whose descriptions are considered old, and $P_{fresh} = \{p \in P_H \mid p.v = 0\}$ the set of peers whose descriptions are considered fresh according to their current data instances. Thus, if a query Q is propagated only to the set $V = P_Q \cap P_{fresh}$, then precision is maximum since all visited peers are certainly matching peers (no false positives), but recall depends on the fraction of false negatives in query results that could be returned by the set of excluded peers $P_Q \backslash P_{fresh}$. On the contrary, if the query Q is propagated to the extended set $V = P_Q \cup P_{old}$, the recall value is maximum since all matching peers are visited (no false negatives), but precision depends on the fraction of false positives in query results that are returned by the set of peers P_{old}.

Now we consider that the selectivity of query Q is very high, such that each relevant peer has only one result tuple. Thus, when a user requires C_t tuples, we have to visit C_t relevant peers. The cost of inter-domain query flooding is given by:

$$C_f = ((1 - FP) \cdot |P_Q| + 2) \cdot \sum_{i=1}^{TTL} k^i \;\; messages,$$

where k is the average degree value (e.g. average degree of 3.5, similar to Gnutella-type graphs). Remember that, the set of relevant peers who have answered the query (i.e. $(1 - FP) \cdot |P_Q|$), the originator and the summary peers participate to query flooding. In this expression, we consider that a summary peer has on average k long-range links to k summary peers. As a consequence, the total cost of a query is:

$$C_Q = C_d \cdot \frac{C_t}{(1-FP) \cdot |P_Q|} + C_f \cdot (1 - \frac{C_t}{(1-FP) \cdot |P_Q|}) \tag{3.3}$$

Parameter	value
Network configuration	
local summary lifetime L	skewed distribution, Mean=3h, Median=1h
number of peers n	16–5000
Workload configuration	
number of queries q	200
matching nodes/query $hits$	10%
System parameter	
freshness threshold α	0.1–0.8

Table3.4: Simulation Parameters

In this expression, the term $C_t/((1-FP)\cdot|P_Q|)$ represents the number of domains that should be visited. For example, when $C_t = ((1-FP)\cdot|P_Q|)$, one domain is sufficient and no query flooding is required.

3.5.2 Simulation

We evaluated the performance of our solutions through simulation, based on the above cost model. First, we describe the simulation setup. Then we present simulation results to evaluate various performance dimensions and parameters: scale up, query accuracy, effect of the freshness threshold α.

3.5.2.1 Simulation Setup

We used the SimJava package [50] and the BRITE universal topology generator [2] to simulate a power law P2P network, with an average degree of 4. The simulation parameters are shown in Table 3.4. In large-scale P2P file-sharing systems, a user connects mainly to download some data and may then leave the system without any constraint. As reported in [133], these systems are highly dynamic and node lifetimes are measured in hours. As such, data indexes should be mainly maintained against network changes. In collaborative database applications, however, the P2P system is supposed to be more stable. Here, the data indexes should be mainly maintained against data changes, since the shared data may be submitted to a significant modification rate.

As stated before, our work targets collaborative applications sharing semantically rich data. In our tests, we have used synthetic data since it was difficult to obtain/use real, highly distributed P2P databases. Future works aim to employ our proposal in the context of astronomical applications, which seem to be attractive because of the huge amount of information stored into the databases. Thus, to provide meaningful results, we have evaluated the performance of our solutions in worst contexts where the data are highly updated. We have considered that local summary lifetimes follow a skewed distribution with a mean lifetime of 3 hours, and a median lifetime of 60 minutes. Note

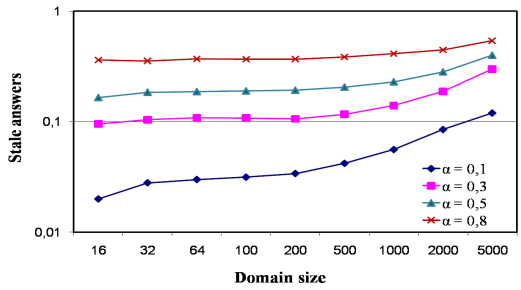

Figure3.11: Stale answers vs. domain size

that summary lifetimes of hours means that the underlying data is submitted to a very high modification rate, since the summaries are supposed to be more stable than original data (as discussed in Section 3.2.3.2). Besides, such lifetime values allow to predict the performance of PeerSum in large-scale data sharing P2P systems where the rate of node departure/arrival dominates the global summary update initiating. In our tests, we work with moderated-size P2P networks, i.e. the number of peers varies between 16 and 5000. In our query workload, the query rate is 0.00083 queries per node per second (one query per node per 20 minutes) as suggested in [35]. Each query is matched by 10% of the total number of peers. Finally, our system parameter α varies between 0.1 and 0.8.

3.5.2.2 Update Cost

In this set of experiments, we quantify the trade-off between query accuracy and the cost of updating a global summary in a given domain. Figure 3.11 depicts the fraction of stale answers in query results for different values of the threshold α. Here, we illustrate the worst case. For each partner peer p having a freshness value equal to 1, if it is selected in the set P_Q then it is considered as false positive. Otherwise, it is considered as false negative. However, this is not the real case. Though it has a freshness value equal to 1, the peer p does not incur stale answers unless its database is changed relative to the posed query Q. Thus, Figure 3.11 shows the worst, but very reasonable values. For instance, the fraction of stale answers is limited to 11% for a network of 500 peers when the threshold α is set to 0.3 (30% of the peers are tolerated to have old/non existent descriptions).

As mentioned in Section 3.5.1.2, if we choose to propagate the query only to the set $V = P_Q \cap P_{fresh}$ we eliminate the possible false positives in query results. However, this may lead to additional false negatives. Figure 3.12 shows the fraction of false negatives in function of the domain size. Here we take into account the probability of the database modification relative to the query, for a peer having a freshness value equal to 1. We see that the fraction of false negatives is limited to 3% for a domain size less than 2000 (i.e. network size less than 8000). The real estimation of stale answers shows a reduction by a factor of 4.5 with respect to the preceded values.

Figure 3.13 depicts the update cost in function of the domain size, and this for two threshold values. The total number of messages increases with the domain size, but not

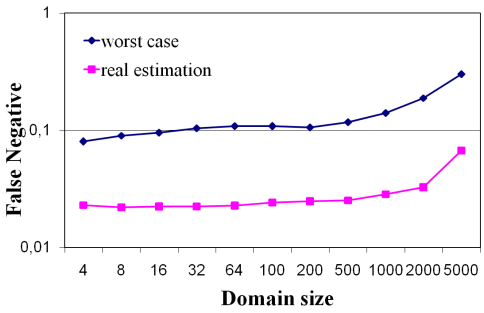

Figure3.12: False negative vs. domain size

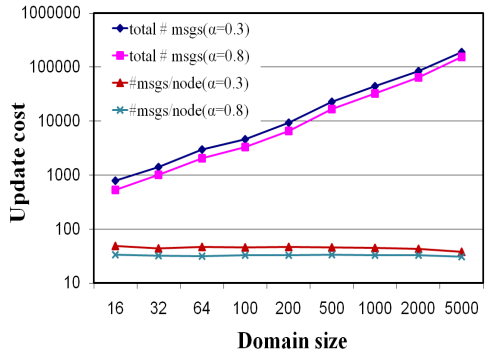

Figure3.13: number of messages vs. domain size

surprisingly, the number of messages per node remains almost the same. In the update cost equation 3.5.1.1, the number of push messages for a given peer is independent of domain size. More interestingly, when the threshold value decreases (from 0.8 to 0.3) we notice a little cost increasing of 1.2 on average. For a domain of 1000 peers, the update cost increases from 0.01056 to 0.01296 messages per node per minute (not shown in figure). However, a small value of the threshold α allows to reduce significantly the fraction of stale answers in query results, as seen in Figure 3.11. We conclude therefore that tuning our system parameter, i.e. the threshold α, do not incur additional traffic overhead, while improving query accuracy.

3.5.2.3 Query Cost

In this set of experiments, we compare our algorithm for query processing against centralized-index and pure non-index/flooding algorithms. A centralized-index approach is very efficient since a single message allows locating relevant data. However, a central index is vulnerable to attack and it is difficult to keep it up-to-date. Flooding algorithms are very used in real life, due to their simplicity and the lack of complex state information at each peer. A pure flooding algorithm consists in broadcasting the query in the network till a stop condition is satisfied, which may lead to a very high query execution cost. Here, we limit the flooding by a value 3 of TTL (Time-To-Live).

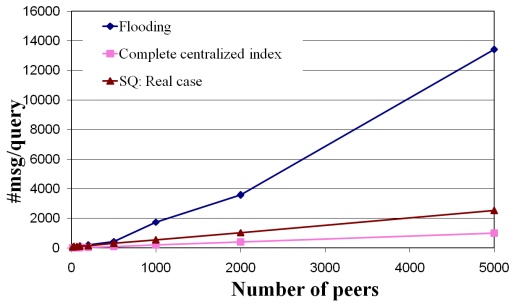

Figure3.14: Query cost vs. number of peers

According to Table 3.4, the query hit is 10% of the total number of peers. For our query processing approach, which is mainly based on summary querying (SQ), we consider that each visited domain provides 10% of the number of relevant peers (i.e. 1% of the network size). In other words, we should visit 10 domains for each query Q. From equation 3.5.1.2, we obtain: $C_Q = (10 \cdot C_d + 9 \cdot C_f)$ messages. Figure 3.14 depicts the number of exchanged messages to process a query Q, in function of the total number of peers. The centralized-index algorithm shows the best results that can be expected from any query processing algorithm, when the index is complete and consistent, i.e. the index covers the totality of data available in the system, and there are no stale answers in query results. In that case, the query cost is: $C_Q = 1 + 2 \cdot ((0.1) \cdot n)$ messages, which includes the query message sent to the index, the query messages sent to the relevant peers and the query response messages returned to the originator peer p.

In Figure 3.14, we observe that our algorithm SQ shows good results by significantly reducing the number of exchanged messages, in comparison with a pure query flooding algorithm. For instance, the query cost is reduced by a factor of 3.5 for a network of 2000 peers, and this reduction becomes more important with a larger-sized network. We note that in our tests, we have considered the worst case of our algorithm, in which the fraction of stale answers of Figure 3.11 occurs in query results (for $\alpha = 0.3$).

3.6 Conclusion

This chapter proposed a solution for managing data summaries in P2P systems. Our approach for data summarization is based on SaintEtiQ [126]: an online linguistic approach for summarizing databases. The innovation of our proposal relies on the double exploitation of the employed summaries. First, as semantic indexes, they support locating relevant nodes based on their data descriptions. Second, due to their intelligibility, they can be directly queried to approximately answer user queries, without the need for exploring original data.

The first part of this work studied the integration of our summarization technique into the APPA system. Thus, we defined a PEERSUM service which manages data summaries relying on services provided by the APPA's architecture. Then, the second part proposed

a solution for managing data summaries in hierarchical P2P networks. In both contexts, we first defined an appropriate summary model which describes the different levels of summaries materialized in the network. Second, we proposed efficient solutions for building summaries and updating their intentional descriptions against both data and network changes. Then, we defined a query processing mechanism which relies on querying the available summaries. This mechanism is able to perform two distinct tasks, according to user/application requirements. It may determine the set of relevant nodes to which the query should be sent in order to get precise answers (i.e. peer localization). It may also directly return approximate answers from the intentional descriptions of the queried summary (i.e. approximate answering). We evaluated the performance of our proposals through a cost model and a simulation model. We showed that the use of summaries in P2P systems allows to reduce the query cost, without incurring high costs of summary update.

Chapter 4

CBST for Unstructured P2P Systems

As discussed before, the techniques that have been proposed to improve the performance of P2P systems fall into four main categories. *Data indexing* and *network clustering* techniques mainly aim to reduce the response time and the bandwidth consumption while locating relevant data. *Caching* and *replication* techniques mainly address the consistency and the availability of the shared data. *Mediation* techniques tend to remedy the problem of integrating heterogeneous data.

In the previous chapter, we have proposed a data indexing technique that allows for efficient data discovery in advanced P2P applications. However, this technique also belongs to the newly arising data summarization category, which allows for an approximate query answering through providing intelligible data descriptions.

In this chapter, we address the problem of reducing network traffic in unstructured P2P systems, based on network clustering schemes. To this end, we propose $CBST$, a Cluster-Based Search Technique for P2P systems, which is implemented over a connectivity-based clustering protocol. The objective of $CBST$ is to reduce the total number of messages generated per query. Ideally, each generated message has to be useful to the query propagation mechanism, i.e. to enlarge the coverage of query propagation through reaching a new peer. In the following, we discuss the motivation behind this work, describe our contribution, and present the chapter organization.

4.1 Motivation

While structured systems are well-studied in the P2P research community, unstructured systems are still the most deployed systems in today's internet. Unstructured systems are attractive because of their simplicity and their high robustness. However, the high bandwidth consumption in these networks remains a pertinent issue for both users and internet-service providers. For the end users, the most faced problem is that the participation into the network swamps all the available bandwidth and thus renders the link ineffective for any other use. For internet-service providers and network operators,

the P2P traffic tends to be inefficient and costly while transiting on the physical network layer.

The amount of traffic generated on a P2P network is dependent on two factors: the nature of application and the way in which the messages are forwarded. The application operating on the network directly impacts the amount of traffic. For instance, bittorent [1] would generate significant traffic since it is especially designed for the use of downloading large files. This first factor is application-dependent and thus uncontrollable by P2P network designers. The research effort has been mainly put on providing efficient message routing protocols. This second factor is in fact reflective of the properties of different types of P2P network topologies (i.e. their degrees of structuring and decentralization).

In unstructured networks, there is no control on either the overlay topology or the data placement, and thus queries are non-deterministically routed. Basically, the fundamental routing mechanism is flooding, in which a peer sends a message to its neighbors, which in turn forward the message to all their neighbors except the message sender. A query message is associated with a Time-To-Live (TTL) value, which is decreased by one when it travels across one hop. At a given peer, the message comes to its end if it becomes redundant (i.e. no further neighbors) or the TTL value becomes zero. This mechanism is simple, highly reliable and allows to reach a large number of peers in the system (i.e. high coverage). Despite of these advantages, there are two major concerns with flooding.

1. **Blindness:** a peer forwards a query message, whether self-initiated or received, to its neighbors without any information on how these neighbors may contribute to query results. To overcome this problem, many techniques have exploited data semantics at the application layer, to assist query routing with information about the location or the direction toward relevant data.

2. **Message Redundancy:** a peer may receive the same query message multiple times. This is due to the ad hoc nature of P2P connections, i.e. the neighbor selection process is random and non-discriminant.

In our work, we address the second issue, based on connectivity-based clustering schemes. Such clustering schemes are considered as a way to discover inherent structural patterns from the overlay topology. The network is partitioned into clusters based on node connectivity, such that nodes within clusters are highly connected, while nodes between clusters are loosely connected. In the literature, two main protocols have been proposed, i.e. the Connectivity-based Distributed node Clustering (CDC) [121], and the SCM-based Distributed Clustering (SDC) [86]. These works have been introduced by arguing the benefits of such a clustering from the search performance point of view. However, none of them has proposed an appropriate routing protocol, or provided analytical models or experimental results on how their clustering schemes contribute to reduce the bandwidth consumption. The main focus was on the clustering scheme accuracy, i.e. using the Scale Coverage Measure (SCM), and its maintenance against node dynamicity.

This chapter proposes $CBST$, a cluster-based search technique ($CBST$) for unstructured P2P systems. This proposal leverages the idea of exploiting the clustering

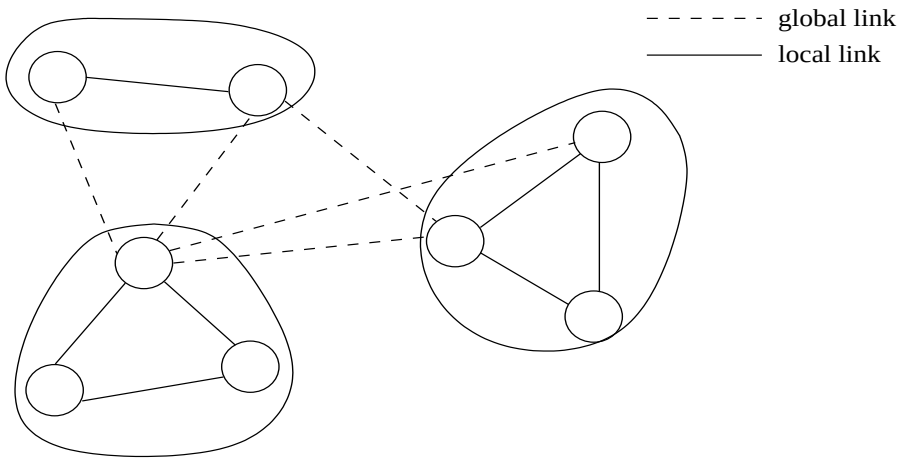

Figure4.1: Local and global links in a clustered network

features of the overlay topology, in order to reduce the network traffic by eliminating redundant query messages.

4.2 Contribution

We consider an unstructured P2P network that is clustered based on node connectivity, as presented in Figure 4.1. Each node belongs to, at most, one cluster and maintains two types of connections: local links that connects to nodes within its cluster (intra-cluster), and global links that connects to nodes in other clusters. Consequently, nodes can perform two types of broadcasts, local and global, along the two types of links. Note that an *orphan* node, which has not joined any other cluster in the network, is considered as a separate cluster on its own. All along this work, we assume that the network is *index-free*, i.e. nodes do not maintain any global distributed indexes.

Now suppose that a query Q is posed at node n_o, with a given value of TTL. We define the query scope QS as being the subgraph that contains n_o, and all other nodes locating at a maximum distance of TTL from n_o. Remember that, in the previous chapter, the query scope has been defined as the set of nodes that should be visited to answer the query. However, because of the flooding blindness, each node that could be reachable based on the TTL value, should be visited since it may contribute to the query results. All nodes may *equi-probably* answer the query.

The performance of query flooding is generally quantified by the pair (M, P). M is the number of generated query messages, and P is the number of visited peers. As said before, the flooding mechanism provides a highest coverage (i.e. $P = |QS|$). But, the main issue is the number of redundant messages, which is obtained by $M - P$. Therefore, our objective is to define a search technique that improves the flooding performance by eliminating the redundancy in query messages (i.e. reducing the M value), while keeping a good coverage (i.e. P slightly less than QS). To illustrate the above problem, let us

think of the graph corresponding to the P2P network as the road map of a given city. The roads are the edges of the graph, and the intersection of two or more roads are the nodes of the graph. Suppose that a group of salespersons is in charge of selling the same merchandise in that city, and each one is paid for the number of articles he could sold. The salespersons do not have any information about the probability of finding a client interested in their merchandise (i.e. absence of data indexes). Thus, their mission is to visit the largest number of clients with minimum traveling. In other words, each road should be traversed only once.

In our work, we propose $CBST$, a search technique that relies on structural information provided by the underlying connectivity-based clustering protocol. The $CBST$ technique works as follows. Based on a local knowledge about the network clustering in its neighborhood, each node maintains information about the local links within its cluster (intra-cluster information), as well as about global links connecting its cluster to other reachable clusters (inter-cluster information). The intra-cluster routing information could be represented by a spanning tree, rooted at that node and covering all its partners (i.e. the nodes that belong to its cluster). These information are efficiently gathered and maintained in a cluster-based routing table.

The benefits in query routing are two folds. First, the query Q is efficiently disseminated in a given cluster, without any redundant messages, using the spanning tree of the first node contacted in that cluster. Second, the query messages between clusters are restricted to those traversing the global links specified by the routing table of the querying node n_o. Extensive simulations have demonstrated the efficiency of the $CBST$ technique compared to the pure flooding and random walk routing techniques.

Therefore, this chapter makes the following contributions:

1. Definition of cluster-based routing tables on top of connectivity-based clustering scheme: application to SDC.

2. Specification of a cluster-based query propagation protocol.

3. Presentation of the missing proof of the cluster-based search techniques efficiency (by extensive simulation and experimental results).

The rest of this chapter is organized as follows. Section 2 discusses related works. Section 3 gives an overview over the connectivity-based clustering schemes. In section 4, we describe our cluster-based search technique $CBST$. Section 5 discusses the trade-off between search accuracy and bandwidth consumption that can be achieved by $CBST$. Section 6 presents the performance evaluation through simulation. Section 7 concludes.

4.3 Related Work

In this section, we first briefly reconsider some works that have been discussed in the first chapter, and which are related to the issue addressed here. Then, we present a general taxonomy of the network clustering schemes.

In the literature, much work has been done to improve the performance of unstructured P2P systems, through refining the network topologies and their message routing protocols.

Early works on the Gnutella system have adopted to build a hierarchical overlay topology. The idea is that node heterogeneity could be exploited by introducing asymmetry in node functionalities (i.e. *heterogeneity-aware* topologies). More capable nodes are dynamically designated as supernodes. A supernode groups a set of leaf nodes, and broadcasts messages on their behalf. This allows to reduce the network traffic and to optimize the usage of network resources. However, the asymmetry in node roles should not incur load balancing or fault tolerance problems.

Later, structured P2P systems have been proposed to remedy the problem of randomness and non-deterministic query routing, while keeping a complete symmetry in node roles. The principle behind is that the overlay topology and data placement are tightly controlled in order to achieve an efficient, deterministic lookup. However, this solution compromises node autonomy, and may restrict the query expressiveness since processing complex queries in structured systems remains an open issue.

Parallel works have focused on improving the performance of unstructured systems through proposing refined variations of the original flooding algorithm (e.g. [136], [143]). They mainly attempt to reduce the total number of messages generated per query. A representative example is the random walk algorithm [136]. The querying node first sends the query message to a given number k of its neighbors. Then, at each forwarding step, an intermediate node sends the query to a randomly chosen neighbor. Obviously, this approach significantly reduces the number of query messages. However, it has a much lower network coverage compared to the original flooding algorithm, i.e. the number of visited nodes may be also significantly reduced. Indeed, the main issue with these works is that they tend to alleviate the shortcoming of flooding, without exercising much care to retain its merit (i.e. good network coverage).

To address the blindness problem of flooding-based approaches, many works have employed data indexing techniques (e.g. [9], [34]) which have been discussed in the first chapter of this thesis. P2P data indexes should be small in size, efficiently updated against both data and network changes. Besides, to support sharing semantically rich data, data indexes should also refer to data content (i.e. semantic indexes). Unlike DHTs, such semantic indexes allow to efficiently guide the query propagation mechanism, without restricting search expressiveness.

Another research axis has focused on network clustering which consists in organizing the nodes into clusters, based on a given criterion. A clustering criterion could be a physical network metric (e.g. bandwidth, latency), some peer property/behavior (e.g. node connectivity/stability), or even defined at the application layer (e.g. similar interests).

4.3.1 Clustering at the physical network layer

Works such as [90], [130], [148] perform network clustering based on physical proximity, which can be defined by measuring latencies between nodes. These clustering schemes address the problem of mismatching between the logical P2P network and the underlying

physical network: two neighbors in the physical overlay are not necessarily neighbors in the P2P overlay. This network mismatching increases the network traffic since a logical hop may amount to an unnecessarily large number of physical hops. While it might at best be considered inefficient in stationary networks, this problem could prove critical in the context of mobile adhoc networks (MANETs) [157]. In a recent trend, more research effort is directed toward the deployment of P2P systems in the context of wireless adhoc networks, which are bound by physical constraints (e.g. coverage area of transmission signals) [32], [76]. Besides, there are many contexts where it is desirable to have physical proximity between network entities, including multicasting. Narada [39], an end system multicast, first constructs a rich connected graph on which to further construct shortest path spanning trees. Each tree rooted at the corresponding source using well-known routing algorithms. This approach introduces large overhead of forming the graph and trees in a large scope, and does not consider the dynamic joining and leaving characteristics of peers. The overhead of Narada is proportional to the multicast group size. In P2P systems, the main approach used for clustering nodes based on physical proximity consists in measuring latencies between each peer and multiple stable internet servers called "landmarks". This approach allows to estimate the latency between nodes. However, it is usually performed in the global network, which might impact the measurements accuracy.

4.3.2 Clustering at the application layer

Works such as [10], [132] perform network clustering at the application layer, based on semantic proximity or interest-based proximity. Semantic proximity between peers is defined as the similarities between their cache contents or download patterns [57]. These clustering schemes tend to overcome the flooding blindness by connecting semantically related peers in the P2P network. In [10], the authors assume global knowledge of the semantic grouping of the shared documents, according to a predefined classification. Accordingly, they build multiple overlays, each of which corresponding to a separate semantic relationship. The main issue with this approach is that it is difficult to obtain a precise classification, and to maintain the resultant overlays against changes in nodes preferences. In [132], the concept of shortcut is proposed, allowing peers to form direct connections to peers of similar interests. The similarity of interests are captured implicitly based on recent downloads and accordingly, *interest-based shortcuts* are dynamically created in the network. In the original network, these shortcuts are discovered progressively through flooding. Besides, the search is first done on interest-based shortcuts. However, when shortcuts fail, peers resort to using the underlying Gnutella overlay and its original flooding mechanism. So a low-cost flooding is also essential in this clustering scheme.

4.3.3 Clustering at the logical network layer

Works such as [86], [121] perform network clustering at the P2P network layer, based on node connectivity. Connectivity-based clustering schemes have wide ranging applications

such in mobile adhoc and P2P sensor networks. These clustering schemes partition the network into non-overlapping sub-networks (clusters) with bounded *diameter*. A cluster diameter is defined as the maximum length of the shortest paths among all pairs of nodes in that cluster. One approach consists in designating cluster heads which form a dominating set of the network. These cluster heads will coordinate their one-hop neighbors [100], [151] (e.g. control channel access, power measurements, routing responsibilities, etc.). Certainly, this approach requires efficient load balancing strategies [18]. Another approach adopts a more symmetric topology by proposing fully decentralized clustering without the designation of cluster heads [79], [49]. This sort of clustering infrastructure serves to create a dynamic backbone of a network for the purpose of improving the quality of service of the application operating on the network. For instance, efficient routing protocols could be defined by taking advantage of the clustering features of the underlying network [98], [49]. Another example is that the characterization of the inherent clustering features can be very critical in network modeling [33].

In this chapter, we are dealing with connectivity-based clustering schemes when proposing a cluster-based search technique for P2P systems. We are not working at the application layer where clustering is based on semantic relations and addresses the issue of blind searches, from the data placement point of view. We are not dealing either with topologically-aware clustering which is a separate issue on its own. In fact, our work simply consists in exploiting the characteristics of a P2P network topology, to draw a kind of network map at each node. Such a map allows message to travel in the network while distinguishing between short and long routes, and being aware of impasses.

4.4 Connectivity-based clustering schemes

A connectivity-based clustering protocol should fulfill the following requirements in order to be considered as appropriate for P2P systems.

- A natural requirement is that it should organize the network such that nodes are highly connected in the same clusters and less connected between clusters.
- It should well control the cluster size (or cluster diameter). Due to the lack of knowledge about network structure, it is expensive to maintain expanded clusters in large P2P networks.
- It should be fully distributed. Nodes should form clusters automatically without the knowledge of the complete network topology.
- It should recover from node dynamics, with small overhead in term of the number of messages exchanged between nodes.

Centralized clustering algorithms, like MCL (Markov Cluster) [129], can achieve high clustering accuracy, i.e. the first requirement is well satisfied. However, such algorithms assume that the complete network topology is available at a central point, and thus cannot

be used in P2P networks. The CDC scheme [91] is a distributed approach that discovers connectivity-based clusters in P2P networks. A set of nodes are selected as "originators" and clusters are formed around them using flow simulation. The quality of the clustering scheme depends on how well the "originators" are distributed in the network. Furthermore, the main issue with the CDC algorithm is that it cannot handle node dynamicity in a decent way. The whole network has to be re-clustered at each node join or leave, which incurs a large traffic overhead.

The SCM-based Distributed Clustering (SDC) protocol [86] proposes to satisfy all the design criteria discussed above. The protocol performs in a fully distributed way, which is briefly described in Section 4.4.2. The clustering accuracy is dynamically adjusted using the Scaled Coverage Measure (SCM), a practical clustering accuracy measure, which is presented in Section 4.4.1. Besides, the cluster size is well controlled, and the node arrival/departure is locally handled with a small number of messages, while keeping a good quality of clustering.

4.4.1 SCM: Accuracy Measure for Graph Clustering

The *Scaled Coverage Measure* (SCM) has been proposed by S.Van Dangon [129] to evaluate the accuracy of a clustering scheme. The key idea behind this performance measure is that an optimal clustering of a given graph should minimize both the number of inter-cluster edges and the number of non-neighbor vertices in each cluster. Let $G = (V, E)$ be a graph, where V is the set of nodes corresponding to the set of peers in a P2P system, and E is the set of links, which are the logical connections between peers. We assume that $C = \{C_1, C_2, \ldots, C_l\}$ is a given clustering on graph G. Each cluster C_i is a non-empty subset of V, and $\cup_{i=1}^{l} C_i = V$. Given a node $n_i \in V$, we have the following notations:

- **Nbr**(n_i): the set of neighbors of node n_i;
- **Clust**(n_i): the set of nodes in the same cluster as node n_i (excluding n_i);
- **FalsePos**(n_i): the set of nodes in the same cluster as n_i but not neighbors of n_i;
- **FalseNeg**(n_i): the set of neighbors of n_i but not in the same cluster as n_i.

Then the Scaled Coverage Measure of node n_i with respect to the clustering C, $SCM(n_i)$, is defined as:

$$SCM(n_i) = 1 - \frac{|FalsePos(n_i)| + |FalseNeg(n_i)|}{|Nbr(n_i) \cup Clust(n_i)|} \tag{4.1}$$

The SCM value of the graph G, $SCM(G)$, is defined as the average of the SCM values of all of the nodes: $SCM(G) = \sum_{n_i} SCM(n_i)/N$, where N is the network size (i.e. $|V| = N$). It is easy to see that this value, which lies in $[0, 1]$, reflects the accuracy of clustering. The higher the SCM value, the smaller the connectivity between clusters and the higher the connectivity within clusters. Note that the SCM value of an orphan node is 0. Thus,

an additional requirement for a good clustering C is that it should result in a minimum number of orphan nodes, since they reduce its accuracy.

For any graph, there exists a highest SCM value that depends on the inherent network structure. To illustrate, this value is equal to 1 for a graph containing only isolated, fully connected components. The SCM measure allows to quantify, and thus to compare the accuracy of different clustering schemes. However, in the context of P2P systems, the comparison of two clustering schemes should not be restricted to the comparison of the optimal SCM values they can achieve on the network. As we discussed earlier in this section, a "good" clustering scheme should be also fully distributed, and should handle peer dynamicity in a very efficient way.

4.4.2 The SDC Protocol

Given a network, each node n_o is initialized as an orphan node with its own *clust_id* (any unique id is sufficient) and *clust_size* (1 in this case). For SCM computation, node n_o maintains two variables a_o and b_o such that:

$$a_o = |Nbr(n_o) \cup Clust(n_o)|\,, \quad b_o = |FalsePos(n_o, C)| + |FalseNeg(n_o, C)|$$

Initially, $a_o = b_o = |Nbr(n_o)|$, and according to Equation 4.1:

$$SCM = 1 - \frac{b_o}{a_o} \tag{4.2}$$

When running the SDC protocol, all nodes start to exchange messages with their neighbors, conduct some simple computation, and form clusters in a greedy manner. After a number of rounds of communication, the clustering procedure becomes stable without further message exchange and the network is finally clustered.

In order to be clustered, a node n_o executes the clustering procedure, which may involve the following clustering messages.

- **Clust_Probe.** Node n_o sends this message to its neighbors to discover the "neighbor clusters", i.e. clusters that exist in its neighborhood. Upon receiving this message, each neighbor will send its *clust_id* and *clust_size* back to n_o.

- **Clust_Request.** Node n_o issues a *Clust_Request* message, which will be flooded in each neighbor cluster C_i (i.e. request for joining C_i) and n_o's current cluster $Clust(n_o)$ (i.e. request for leaving $Clust(n_o)$). To control the number of exchanged messages, as well as to control the cluster diameter, a Time-To-Live (TTL) value is associated to this message. In the rest of this chapter, this TTL value will be referred as *diameter* (D), to distinguish it from the TTL value associated to a user query.

- **Clust_Reply.** Upon receiving the *Clust_Request* message, a node n_j in C_i computes the gain $\Delta SCM(n_j)$ assuming that node n_o joins C_i. Note that this computation

only requires the information of whether n_o is a n_j's neighbor or not. According to equation 4.2, if $n_o \in Nbr(n_j)$,

$$\Delta SCM(n_j) = 1/a_{n_j}.$$

Otherwise, the gain in SCM is given by,

$$\Delta SCM(n_j) = b_{n_j}/a_{n_j} - (b_{n_j}+1)/(a_{n_j}+1).$$

Similarly, each node in $Clust(n_o)$ has to compute its gain as if n_o leaves its current cluster. After gain computation, node n_j sends back a *Clust_Reply* message carrying $\Delta SCM(n_j)$ and n_j's *clust_id* back to node n_o.

- **Clust_Reject.** Based on the *diameter* value D in *Clust_Request*, a node n_j in C_i can determine whether or not the cluster diameter will be excedeed due to the joining of node n_o. In this case, n_j stops forwarding *Clust_Request* and a *Clust_Reject* message is sent back to node n_o. Once receiving such a message, node n_o will remove C_i from the list of neighbor clusters which it attempts to join.

- **Clust_Update.** Upon receiving *Clust_Reply* messages from all the nodes in its cluster and the neighbor cluster C_i (in the case where no *Clust_Reject* is received from the latter), node n_o computes the overall gain $\Delta SCM(G)$ assuming it leaves its original cluster and joins C_i. If $\Delta SCM(G) > 0$, n_o should join C_i. There might be multiple clusters of which ΔSCM are positive, n_o should join the one with the maximum SCM gain. Once n_o determines which cluster to join, a *Clust_Update* message containing its *clust_id* is flooded in its original cluster and the new cluster it will join. Then, n_o and each node receiving this message will update their clustering information.

- **Clust_Wait.** This message is used in the case where multiple nodes try to join the same cluster simultaneously. Only one node can be served, and other nodes are "locked" until this node achieves its clustering procedure. A node that receives a *Clust_Wait* message has to wait a predefined period of time before its next clustering attempt.

After node n_o joins the new cluster, its neighbors in the original cluster are affected and should check whether they should join other clusters, in the same way as node n_o has done. The whole procedure will end if no node can join any cluster based on $\Delta SCM(G)$ and the cluster diameter control.

4.4.2.1 Performance in static systems

The SDC protocol provides good clustering accuracy. Moreover, it performs much better than the existing decentralized connectivity-based clustering protocol (i.e. the CDC protocol), yielding much smaller message overhead, especially for power-law topologies.

Another observation is that for the same network size, power-law topologies have smaller overhead than pure random topologies, in which nodes have the same probability of being connected with each others. This is because, the message overhead in power-law topologies is mainly incurred by the clustering of a small number of high degree nodes.

4.4.2.2 Performance in dynamic systems

Compared to the CDC protocol, it has been demonstrated that the SDC protocol can maintain a higher clustering accuracy after each node join or leave, while only much smaller overhead is introduced. This is due to the very efficient mechanism adopted by SDC to handle node dynamicity. In fact, with node arrival and departure, the network structure is changed and the existing clusters are affected. Re-doing the whole clustering procedure, as suggested by CDC, may keep good clustering accuracy. However, this is very inefficient and the procedure may never stabilize if the network is highly dynamic. The key idea behind the solution adopted by SDC, is that node entry and exit are localized events and only few nodes are affected and thus need to be re-clustered.

4.4.2.3 Cluster Size Control

In [86], simulation results have shown that the average cluster size obtained from both SDC and CDC protocols are very stable, for topologies with different scales. Intuitively, as long as the overall topology structure is not changed, the cluster size should follow the same pattern even if the topology scale is changed significantly. In addition, the SDC protocol can maintain stricter bounds for the cluster size i.e. the cluster size distribution does not spread as in CDC. Another observation is that SDC can effectively eliminate orphan nodes.

4.5 CBST for unstructured P2P systems

In our work, we aim at defining a cluster-based search technique on the top of a connectivity-based clustering protocol (i.e. the SDC protocol). This technique takes advantage of the clustering information in order to improve the performance of query routing in unstructured P2P networks.

In this section, we first define a mechanism for building *cluster-based* routing tables, and updating their entries against node dynamicity. Second, we propose a routing mechanism that allows a query to travel across clusters such that a maximum number of nodes is visited (for a given value of TTL), and redundant query messages are quasi-eliminated.

4.5.1 Cluster-based Routing Table

Let $G = (V, E)$ be the graph corresponding to an unstructured P2P network, and C the clustering obtained from running the SDC protocol on that graph. Suppose that node

Neighbor	Cost(nb of hops)	Destination
Partner entries		
n_1	h_{11}	p_1
n_2	h_{22}	p_2
...	...	...
n_n	h_{nk}	p_k
Cluster entries		
n_1	h_{11}	C_1
n_2	h_{22}	C_2
...	...	...
n_m	h_{ml}	C_l

Table4.1: Routing table of node n_o

$n_o \in V$ belongs to the cluster $C_o \in C$. According to the notations in 4.4.1, $Nbr(n_o)$ is the set of n_o's neighbors, and $Clust(n_o)$ is the set of n_o's partners, i.e. the set of nodes that belong to the same cluster C_o. Note that $C_o = Clust(n_o) \cup \{n_o\}$.

4.5.1.1 Routing Table Description

Table 4.1 describes the routing table maintained at node n_o. We distinguish between intra-cluster routing information, which is represented by the first set of *partner* entries, and inter-cluster routing information, which is represented by the second set of *cluster* entries. The former provides information about paths to each partner $p \in Clust(n_o)$, while the latter provides information about paths to a subset of reachable clusters.

A partner entry e_p (respectively cluster entry e_c) in Table 4.1 is read as follows. Going through neighbor n_i, node n_o can reach a partner p_j (respectively a new cluster C_j), with a minimal number of hops h_{ij}. Thus, the set of partner entries is equivalent to a spanning tree of the subgraph C_o, rooted at node n_o. The number of these entries is limited to the cluster size, which is well controlled by the clustering protocol. On the other hand, to control the number of cluster entries, node n_o chooses to keep information only about clusters that contain, at least, one node locating at a maximum distance of D. Before presenting how the routing table RT at node n_o is maintained, we list the set of properties that should be satisfied in order to keep RT staleness.

Property 1 *$Pa = Clust(n_o)$, where Pa is the set of destinations (peers) of partner entries in the no's RT ($Pa := \{n_i \in V / \exists e_p \in RT, e_p.destination = n_i\}$).*

Property 2 *$\forall e_p \in RT$, $e_p.cost = min(distance(n_o, n_i))$ over all the intra-cluster paths (n_o, n_i), where $n_i = e_p.destination$.*

These two first properties guarantee that the set of partner entries in n_o's RT represents the spanning tree of the subgraph C_o, rooted at node n_o itself.

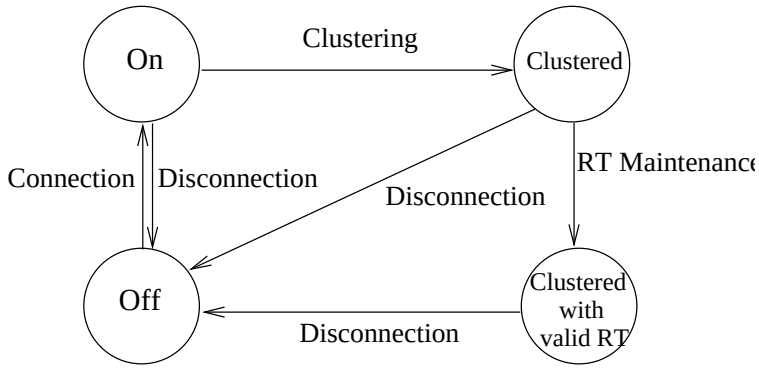

Figure4.2: States of node n_o

Property 3 *$\forall e_c \in RT$, $\exists n_i \in D\text{-}Nbr(n_o)$ such that $n_i.clust\_id = e_c.destination$, where $D\text{-}Nbr(n_o)$ is the D-neighborhood of node n_o ($D\text{-}Nbr(n_o) = \{n_i \in V / distance(n_o, n_i) \leq D\}$).*

This property guarantees that each cluster referred by a cluster entry in n_o's RT is still existing in the network, and contains at least one node belonging to the D-neighborhood of node n_o.

Property 4 *$\forall e \in RT$, $e.dist \leq D$.*

This final property guarantees that the cost values in n_o's RT are all bounded by the predefined threshold D, which is used by the SDC protocol to control the cluster diameter.

4.5.1.2 Table Maintenance vs. Clustering Scheme Changes

For our purpose, we distinguish four node states in a given P2P network (Figure 4.2).

- Once connected to the network, a node n_o becomes "ON", and thus able to participate to the clustering scheme.
- Through running the clustering protocol, node n_o joins the cluster which provides him an optimal SCM value. Thus, node n_o becomes in a "CLUSTERED" state.
- Using the mechanism described in this section, n_o is supported with a cluster-based routing table RT. Here, node n_o is said to be "CLUSTERED WITH VALID RT". By valid we mean that the RT satisfies all the properties listed in the previous section.
- Obviously, node n_o could leave the system at any moment, and thus returns to an "OFF" state.

In this section, we assume that all nodes are ON, and we describe how their routing tables are created/updated while the clustering scheme is changing. The problem of updating these routing tables against node dynamicity will be treated in Section 4.5.1.3.

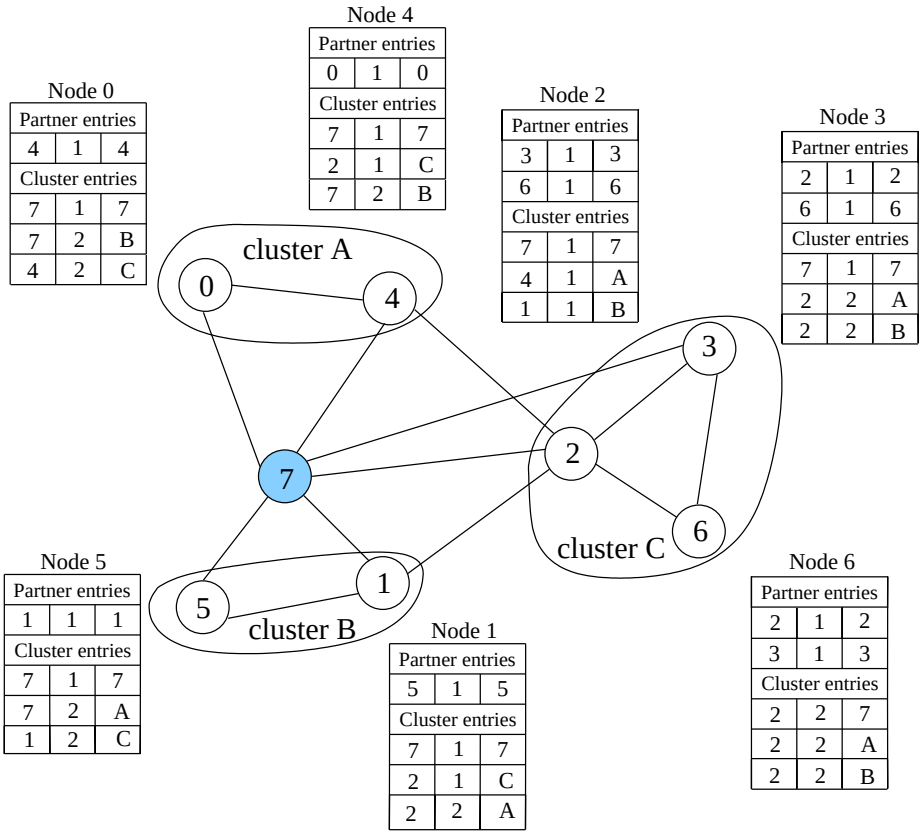

Figure4.3: A network example: node 7 attempts to join a new cluster

Our objective is to allow a node n_o to make the two transitions {CLUSTERING, RT MAINTENANCE} simultaneously. In other words, we aim to exploit the clustering messages exchanged between nodes in order to create/maintain our routing tables. Hence, node n_o, which is moving from cluster C_{old} to cluster C_{new}, should have a valid RT at the end of the clustering procedure, without the need for exchanging additional messages. Certainly, the routing tables of nodes that are affected by this clustering change should also be appropriately maintained. For illustration, consider the network of Figure 4.3, which contains 8 nodes and 4 different clusters (the orphan node 7 is considered as a separate cluster). All nodes maintain their initial routing tables, which are in a valid state, when node 7 attempts to join a new cluster. we assume that node 7 will execute the clustering procedure for the first time, so its initial routing table is empty.

a) Table of node n_o:

According to SDC, node n_o first probes its neighborhood to find candidate clusters to which it could join. Each neighbor sends its cluster *clust_id* back to n_o. Thus, upon receiving the return messages, node n_o updates its routing table RT by adding a cluster entry for each candidate cluster C_i, if such an entry does not already exist. Note that, if

node n_o is not executing the clustering procedure for the first time, its initial table RT is not empty and may already contain an entry for a given candidate cluster. Thus, only for new discovered clusters, entries in the following form are added: $e_c = \langle n_i, 1, C_i \rangle$. Node n_i is a neighbor that belongs to cluster C_i. In our example, Table 4.2 shows the initial table of node 7 who has discovered the three clusters A, B and C.

Neighbor	Cost(hops)	Destination
Partner entries		
Cluster entries		
node 0	1	*Cluster A*
node 1	1	*Cluster B*
node 2	1	*Cluster C*

Table4.2: Initial routing table of node 7

Neighbor	Cost(hops)	Destination
Partner entries		
node 1	1	*node* 1
node 5	1	*node* 5
Cluster entries		
node 0	1	*Cluster A*
node 2	1	*Cluster C*

Table4.3: Final routing table of node 7

After message exchange and SCM gain computation (as described in Section 4.4.2), node n_o may either stay in its old cluster C_{old} or join a new cluster cluster C_{new}. In the first case, the initial RT of node n_o becomes its final table. In the second case, the initial routing table is updated as follows.

- The cluster entry that corresponds to the newly joined cluster C_{new} is removed. In our example, node 7 joins cluster B and thus, the second entry of Table 4.2 is removed from its final table Table 4.3.
- According to the size of the old cluster (C_{old}_Size), we distinguish two cases. If $C_{old}\_Size > 2$, then node n_o has not been orphan and has had entries for its old partners in C_{old}. In this case, n_o chooses (randomly) a node n_p in $|Pa \cap Nbr(n_o)|$ and adds the following cluster entry: $e_c = \langle n_p, 1, C_{old} \rangle$[1]. Note that clustering based on the SCM measure guarantees that the set P_a contains at least one neighbor (i.e. $|Pa(n_o) \cap Nbr(n_o)| \neq \Phi$). For instance, if node 2 left cluster C, it should remove the partner entries of nodes 3 and 6. But, it should add the following entry: $\langle node\ 3, 1, Cluster\ C \rangle$ to remind the existence of cluster C in its neighborhood. If $C_{old}\_Size \leq 2$, the old cluster will disappear with n_o's departure, and thus a corresponding cluster entry should not be added to RT.
- Entries that describe paths to new partners are added, e.g. two partner entries for nodes 1 and 5 are added to Table 4.3. A new partner is identified thanks to the $Clust\_Reply$ message it has sent to node n_o.

b) Table of an affected node n_j:

The clustering change made by node n_o will certainly incur some modifications

[1]Recall that P_a is the set of old partners provided by the destination column in RT.

on other nodes' routing tables. In the following, we describe how these modifications are made in order to keep all routing tables in a valid state.

According to SDC, node n_o sends a $Clust\_Update$ message to be flooded in both old (if still exist) and new clusters (e.g. cluster B). This message allows old and new partners to update their clustering information, i.e. the cluster size and SCM values. Such a message can be also used to update the routing tables of visited nodes. To this end, we assume that it is flooded in all candidate clusters, and is associated with the routing table of its sender. Hence, the $Clust\_Update$ message forwarded by node n_i has the following structure.

$$Clust\_Update = \langle n_i.clust\_id, RT_i, TTL_i \rangle ,$$

where $n_i.clust\_id$ is the identifier of the n_i's current cluster, and RT_i is its current routing table. The TTL value is initialized by node n_o to D, and is decremented by one after each message hop.

Suppose now that node n_j receives the $Clust\_Update$ message from node n_i. Algorithm 4.5.1.2 shows the steps involved in updating its routing table RT_j. Routing table modifications are illustrated through comparing the initial routing tables in Figure 4.3 and the final ones which are presented in the final network in Figure 4.4.

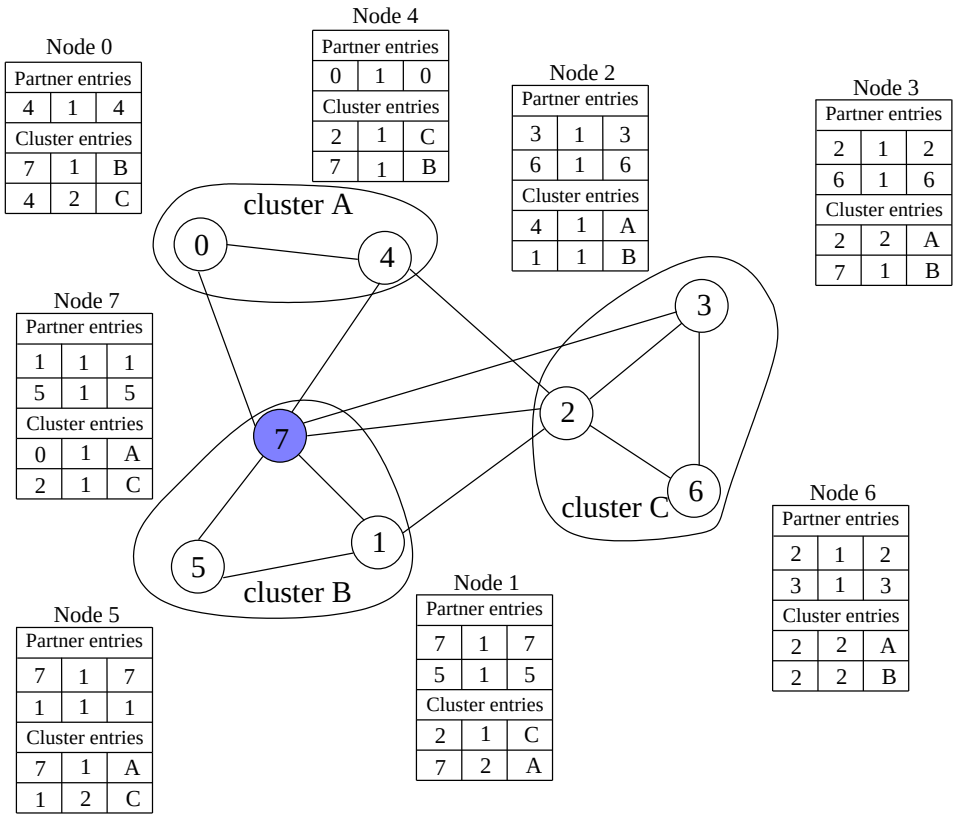

Figure4.4: Final routing tables

First, each cluster entry that supposes that node n_o belongs to the old cluster C_{old} should be removed. This entry deletion starts at n_o's neighbors (Lines $5-7$), and is then

Algorithm 3 *Routing Table Update*

1: *RT_i (respect RT_j): Routing table of the sender node n_i (respect current node n_j)*
2: *C_i (respect C_j): Cluster of the sender node n_i (respect current node n_j)*
3: *// 1.Remove entries referring to clusters which are not reached anymore through n_i*
4: **For each** (e_c in RT_j such that $e_c.neighbor = n_i$)
5: **if** ($e_c.cost = 1$) **and** ($C_i \neq e_c.destination$) **then**
6: RT_j.**remove**(e_c)
7: **end if**
8: **if** ($e_c.cost > 1$) **and** ($\nexists e'_c$ in RT_i such that $e_c.destination = e'_c.destination$) **then**
9: RT_j.**remove**(e_c)
10: **end if**
11: **end For each**
12: *// 2.Remove partner entries referring to nodes which do not belong anymore to C_j*
13: **For each** (e_p in RT_j such that $e_p.neighbor = n_i$)
14: **if** ($C_i \neq C_j$) **then**
15: RT_j.**remove**(e_p)
16: **else if** ($\nexists e'_p$ in RT_i such that $e_p.destination = e'_p.destination$) **then**
17: RT_j.**remove**(e_p)
18: **end if**
19: **if** the set of partner entries is **empty then**
20: $C_j := n_j\_id$
21: **end if**
22: **end For each**
23: *// 3.Add new entries, and update common entries based on cost values*
24: **For each** e in RT_i
25: **if** (e is a **cluster entry**) **or** ((e is a **partner entry**) **and** ($C_i = C_j$)) **then**
26: **if** ($\nexists e'$ in RT_j such that $e'.destination = e.destination$) **then**
27: $entry$ = **new** Entry (n_i, ($e.cost + 1$), $e.destination$); RT_j.**add**($entry$)
28: **else if** $e'.cost > (e.cost + 1)$ **then**
29: $e'.cost := (e.cost + 1)$; $e'.neighbor := n_i$
30: **else if** (n_i **is** partner) **and** ($e'.neighbor$ **is not** partner) **then**
31: $e'.neighbor := n_i$
32: **end if**
33: **end if**
34: **end For each**
35: *// 4.Add an entry referring to C_i, if such an entry does not exist*
36: **if** $C_i \neq C_j$ **then**
37: **if** ($\nexists e_c$ in RT_j such that $e_c.destination = C_i$) **then**
38: $entry$ = **new** Entry (n_i, 1, C_i); RT_j.**add**($entry$)
39: **end if**
40: *// 5.Add an entry referring to n_i, if such an entry does not exist at a partner node*
41: **else if** ($\nexists e_p$ in RT_j such that $e_p.destination = n_i$) **then**
42: $entry$ = **new** Entry (n_i, 1, n_i); RT_j.**add**($entry$)
43: **end if**

propagated from a node to another (Lines $8-10$). For instance, the third entry in the initial table of node 6 which refers to the orphan node 7 is removed from its final table.

Second, each partner entry that wrongly supposes that the next hop node (i.e. neighbor field) belongs to the same cluster as the current node should be removed. This allows to delete the partner entry referring to node n_o at its old partners. Similarly, this entry deletion starts at neighbor partners (i.e. Lines $14-15$) and then, is propagated from a node to another (Lines $16-17$).

Third, the clustering change by node n_o may:

- introduce a new cluster into the D-neighborhood of a given node. Thus, a new corresponding entry should be added (Lines $26-27$).
- provide a shorter path to a cluster that already overlaps the D-neighborhood of a given node. Thus, the corresponding entry should be appropriately updated (Lines $28-29$). For instance, the last entry in the initial table of node 6, which describes the path to cluster B, is replaced by the fourth entry in its final table, since it provides a smaller cost value. In the case where an alternative path to a given cluster is provided with a same cost, then the current node chooses the path in which the next hop node is a partner (Lines $30-31$). For instance, in the routing table of node 1, the last cluster entry describes the path to cluster A via node 2, with a cost of 2 hops. This entry in the final table by an entry that describes another path, which goes through the new partner 7 with a same cost.The benefit of such a choice is discussed while presenting our query propagation mechanism.

Fourth, if node n_o has joined another orphan node, then a new cluster C_{new} has been created. In this case, a corresponding entry does not exist at any node in the network. However, such an entry can not be added as described for other cluster entries (Lines $26-27$), since node n_o does not maintain an entry referring to its own cluster (i.e. this new entry is completely absent from the first propagated table RT_o). To this end, checking the condition defined at Lines $36-37$ allows to add an entry for C_{new} at neighbor nodes (Line 38). Then, this entry is propagated to other nodes (Lines $26-29$).

Similarly, node n_o does not maintain a partner entry referring to itself. Thus, checking the condition defined at Lines 41 allows to add an entry for n_o at neighbor partners (Line 42), e.g. at nodes 1 and 5. Then, this entry is propagated to other partners (Lines $26-29$).

Finally, we note that node n_o could have only one partner in its old cluster C_{old}. Upon receiving the *Clust_Update* message, this old partner removes the partner entry and becomes again orphan (i.e. Lines $19-20$). Remind that the old neighbor partners of node n_o are (negatively) affected by its clustering change, and thus re-execute at their turn the clustering procedure in order to determine whether they should also move or not.

4.5.1.3 Table Maintenance vs. Network Changes

So far, we have described how the routing tables are maintained against the modifications issued on the clustering scheme, mainly when nodes move from a cluster to another.

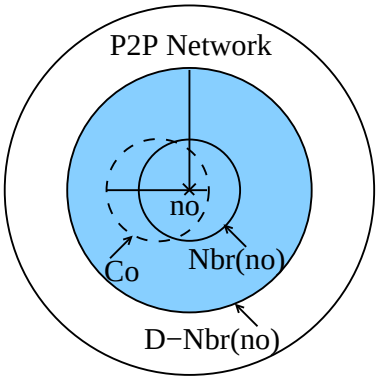

Figure4.5: Network around node n_o

However, once the clustering scheme is established (i.e. all nodes have executed the clustering procedure and found their final clusters), and the routing tables are in a valid state, the only events that can disturb this network stability is node connection/disconnection.

Suppose now that node n_o enters/leaves the P2P system. Figure 4.5 shows the decomposition of the network into subgraphs according to the relative position to node n_o. Here, C_o is the cluster of node n_o before its leaves, or the new cluster it joins upon its arrival into the network. The diameter d of cluster C_o is bounded by the predefined threshold D.

First, consider a node n_i which is out of the D-neighborhood of node n_o (i.e. $n_i \notin D$-Nbr(n_o)). Its routing table RT_i is not affected and remains in its valid state. In fact, according to Property 4, each entry e in RT_i describes a path to a given destination such that $e.cost \leq D$. Hence, each node n_j locating on this path should satisfy the following condition: $distance(n_i, n_j) \leq D$. Since $distance(n_i, n_o) > D$, node n_o does not affect any path information maintained in RT_i. **Therefore, a first conclusion is that the node arrival/departure are localized events and their impacts on routing tables are limited to the D-neighborhood of the arriving/leaving node (i.e. the gray zone in Figure 4.5).**

When node n_o leaves or enters the P2P network, the topology structure in its neighborhood is changed, and thus existing clusters may be affected. In [86], the SDC protocol handles node dynamics in a very efficient way. It ensures a good clustering accuracy, with a low cost in term of the number of exchanged messages. Once again, we aim here at exploiting the clustering messages in order to update our cluster-based routing tables in D-$Nbr(n_o)$.

According to SDC, before leaving the network, node n_o sends a LEAVE message that contains its identifier to all nodes in $Nbr(n_o)$, as well as in C_o through flooding. As such, each affected node in $Nbr(n_o) \cup C_o$ can update its clustering information, i.e. cluster size and SCM value. Concerning the routing tables, all *stale* entries that describe paths traversing node n_o should be removed. To this end, we assume in our work that the LEAVE

message sent by a node n_i has the following structure:

$$leave = \langle n_o, SE_i, TTL_i \rangle$$

where SE_i is the set of stale entries in its routing table RT_i, relative to node n_o's exit. While the TTL_i value controls the message flooding, and is dynamically adjusted at node n_i. In the following, we describe how SE_i and TTL_i are determined.

a) Set of Stale Entries SE:

At a neighbor node n_i in $Nbr(n_o)$, the set SE_i of stale entries is given by:

$$SE_i = \{e \in RT_i \mid e.neighbor = n_o\}$$

For instance, if we consider that node 0 is leaving from the network of figure 4.4, the sets of stale entries at nodes 4 and 7 are: $SE_4 = \{ep_1\}$, and $SE_7 = \{ec_1\}$.

At a non-neighbor node n_j which has received the leave message from a neighbor n_i, the set SE_j of stale entries is given by:

$$SE_j = \{e \in RT_j \mid \exists e' \in SE_i;\ (e.neighbor = n_i)\ and\ (e.destination = e'.destination)\}$$

In our example, the set of stale entries at node 5 which receives the message from node 7 is: $SE_5 = \{ec_1\}$.

b) Dynamic TTL value:

In order to reach all the stale entries in D-$Nbr(n_o)$, a solution could be to flood the LEAVE message with a TTL value, initialized by n_o to D, and decremented by one after each message hop. However, to reduce the number of exchanged messages, we adopt an alternative in which each node n_i locally adjusts the TTL value.

First, if the set SE_i of stale entries is empty at node n_i, the TTL value is set to 0 and the LEAVE message is stopped. Otherwise, the TTL value is set as follows. Let e be a stale entry found at node n_i. This entry describes the path to a given destination *dest*. Suppose now that there is a node n_j who is reaching the same destination *dest* through node n_i. Hence, there is a stale entry e' that should be also removed from the routing table of node n_j. However, according to Property 4:

$$e'.cost \leq D$$

$$\Rightarrow distance(n_j, n_i) + e.cost \leq D$$

$$\Rightarrow distance(n_j, n_i) \leq D - e.cost \tag{4.3}$$

The value of TTL_i should be greater than $distance(n_j, n_i)$ in order to reach n_j and remove its stale entry e'. According to 4.3, TTL_i could be set to $D - e.cost$. By applying this condition to all stale entries at node n_i, we conclude that the TTL value is given by:

$$\boldsymbol{TTL_i = (D - min_{e \in SE_i}(e.cost))}$$

This alternative allows to remove all the stale entries at affected nodes in $D\text{-}Nbr(n_o)$, with a minimal number of LEAVE messages.

Node arrival is treated in a similar way as node departure. We do not give the details of the proposed solution. As stated before, the key idea is to limit the strategy of routing table updates to the affected regions, with a dynamically controlled flooding.

4.5.2 Query Propagation

We describe now how a query Q is propagated in the network, using cluster-based routing tables. The query message is in the following form.

$$Q\_Msg = \langle Q\_id, cluster\_id, TTL, dest\_List \rangle$$

- Q_id: the query identifier. It mainly allows to avoid processing the same query multiple times.
- $cluster\_id$: the cluster identifier of the node sending the query. It mainly allows to avoid serving the same query within the same cluster multiple times.
- TTL: the TTL value which is the maximum number of hops a query Q can make in the network.
- $dest\_List$: the destination list which contains routing information provided by the routing tables of visited nodes. It is the key field that allows to avoid redundant query messages within clusters, as well as between clusters.

Recall that our routing tables do not provide information about the requested objects. The destination list $dest\_List$ serves as a memory that keeps trace of the query all along its path. Thus, at each forwarding step, the query is able to remember all the routes it has already traversed. In other terms, the query propagation mechanism is still *blind* from a data location point of view, but is supported with memory which allows avoiding unnecessary messages.

An $dest\_List$'s element is represented by: $\langle neighbor, dest \rangle$, where $neighbor$ identifies which neighbor of the current node should be chosen for the next query hop, in order to reach destination $dest$ with a minimal cost. A neighbor field value of -1 indicates that the corresponding destination is either reached or targeted by another path that doest not include the current node. To illustrate the query propagation mechanism, suppose that node 0 in our network example issues a query Q_0 (Figure 4.6).

Upon receiving the query message, a node n first decrements the TTL value. If $TTL > 0$, it proceeds to *update* the query message and *forward* it as described by Algorithm 4. Otherwise, the query forwarding is stopped and no further nodes are visited via node n.

To update $dest\_List$, node n calls three different methods.

Algorithm 4 *Query Message Treatment at node n*

1: $Q\_Msg = \langle Q\_id, cluster\_id, TTL, dest\_List \rangle$
2: RT= the routing table of node n
3: **Treat**(Q_Msg)
4: *// 0.Checking the TTL value*
5: $--Q\_Msg.TTL$
6: **if** (Q_Msg.TTL > 0) **then**
7: *// 1.Updating the query destination list*
8: **addNewPartnerElts** to $Q\_Msg.dest\_List$
9: **setElts** of $Q\_Msg.dest\_List$
10: **addNewClusterElts** to $Q\_Msg.dest\_List$
11: *// 2.Forwarding the query message*
12: $Q\_Msg =$ **new** query message ($\langle Q\_id, this.cluster\_id, TTL, dest\_List \rangle$)
13: **For each** elt in $Q\_Msg.dest\_List$
14: **if** (elt.neighbor$\neq -1$) **then**
15: **send** Q_Msg to elt.neighbor
16: **end if**
17: **end For each**
18: **end if**

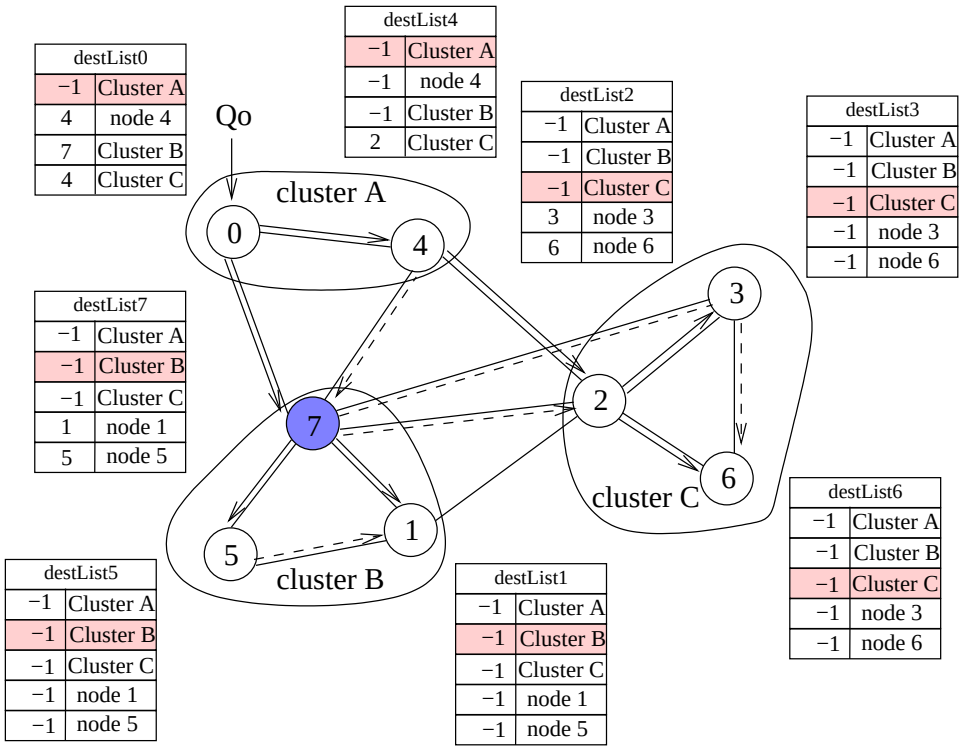

Figure4.6: Cluster-based query propagation

a) Adding new partner entries:
The first method, i.e. *addNewPartnerElts* at Line 10, verifies if n is the first visited node in its cluster. In that case, n adds new partner elements, each of which provides information about the path to a given partner. In other terms, the query will be disseminated in a given cluster according to the spanning tree rooted at the first visited node. For instance, node 7 and node 2, which are the first visited nodes in their clusters B and C (Figure 4.6), are in charge of adding new elements corresponding to their partners.

b) Setting entries:
A second method, i.e. *setAllElts* at Line 11, consists in updating each element e in *dest_List* as follows. If $n = e.neighbor$ which means that node n locates on the path described by element e, then n has to precise the next query hop based on its routing table. Otherwise, the neighbor field value is set to -1 to indicate that all paths that traverse node n are not supposed to reach the corresponding destination.

c) Adding new cluster entries:
A third method, i.e. *addNewClusterElts* at Line 12, checks if the routing table of node n contains information about additional clusters that do not figure in *dest_List*. If such *cluster* entries exist then corresponding elements, which are extracted from n's routing table, are added to *dest_List*.

The algorithms of these three methods are detailed in the Appendix. However, to illustrate the destination list update, consider $dest\_List_4$ of query Q_0 in Figure 4.6. Node 4 belongs to the same cluster as the query sender, which is node 0. The last element in $dest\_List_4$ indicates that through node 2 the query can reach cluster C. The two first entries indicate that node 4, as well as its own cluster A, have been already visited. The third entry informs node 4 that it does not locate on the path leading to cluster B. This allows avoiding redundant messages since node 4 could also send a message to cluster B via its neighbor node 7.

Once *dest_List* updated, node n forwards the query message to each neighbor in *dest_List* (Line 17). For example, node 4 sends the query only to node 2. Note that if node n does not belong to the same cluster as the query message sender, it sets the cluster identifier *cluster_id* of the query to its cluster identifier before forwarding it (Line 14).

Now let us examine the cost of propagating query Q_0 in Figure 4.6. Our cluster-based propagation mechanism requires only 7 messages to cover the entire network (i.e. the continuous arrows). In comparison, a flooding mechanism would result in 4 additional, unnecessary messages (i.e. dashed arrows). Obviously, this propagation efficiency may be compromised by a smaller value of query accuracy, which is quantified by the total number of visited nodes for a given value of TTL. For example, node 6 could be reached with a TTL value of 2 using the flooding technique, while a TTL value of 3 is required for our cluster-based technique. In fact, this trade-off between query accuracy and query propagation cost, which has been highlighted in the introduction of this chapter, will be discussed later.

4.6 Discussion

Search techniques in P2P networks can be evaluated according to the following metrics.

- Efficiency in object discovery. A search is *successful* if it discovers at least one replica of the requested object. The search efficiency could be quantified by the success rate (or *accuracy*), which is the ratio of successful to total searches made.
- Bandwidth consumption. Minimizing message production always represents a high-priority goal for all distributed systems.
- Adaptation to changing topologies. A search technique should adapt to the dynamic nature of P2P systems. In such systems, nodes can enter and leave without any constraint.

4.6.1 Search Accuracy vs. Bandwidth Consumption

As introduced in this chapter, the performance of flooded-based search techniques is quantified by the pair (M, P), where M is the total number of exchanged messages, and P is the total number of visited nodes.

The search accuracy of blind techniques depends on the number of visited nodes P. To increase the probability of finding a replica of the requested object, these techniques tend to propagate the query to a large number of nodes, since they do not dispose of any information related to object locations. However, the main issue is to achieve a good trade-off between search accuracy and bandwidth consumption. In other terms, increasing the number of visited nodes P should not result in increasing the number of redundant messages $M - P$, which unnecessarily overload the network.

a) Intra-cluster messages:

In our approach, a query is disseminated inside a given cluster based on the spanning tree rooted at the first visited node in that cluster. Thus, **the intra-cluster query propagation guarantees that all partner nodes falling into the query scope are visited only once, through the shortest intra-cluster paths. In the case where the cluster diameter $d \leq TTL$, the cluster will be entirely covered**.

b) Inter-cluster messages:

Now suppose that node n maintains information about l clusters in its routing table. If $TTL \leq D$, then the set of clusters visited by the query is included among those l clusters. Otherwise, i.e. if the TTL value permits to go beyond the D-$Nbr(n)$, additional clusters could be visited by the query. Here, two different nodes locating on two different query routes might have information about a same new cluster. Thus, the latter will be reached twice, which may incur redundant messages.

Another issue is that, a cluster entry maintained in n's routing table describes the shortest path to a first contacted node in that cluster. However, it does not guarantee that all the nodes visited in the new cluster are reached with a minimal number of hops.

4.6.2 Clustering Scheme Accuracy vs. Bandwidth Consumption

Besides studying the benefits obtained for query processing, the cost of maintaining an accurate clustering scheme should be taken into account, especially in the context of P2P systems. In [86], simulation results show that the cost of recovering from a given node arrival/departure is very limited, and quasi-independent of the network size. This interesting result is due to the very efficient solution adopted by *SDC* to handle node dynamicity. This cost is limited to 50 messages (the cost of node joining is slightly less than the cost of node leaving), and this for a network size ranging from 1000 to 5000.

In our work, we have mainly used the clustering messages in order to update the cluster-based routing tables. However, we require some additional messages to make all the necessary modifications to all affected tables (Sections 4.5.1.2 and 4.5.1.3).

According to SDC, the *Clust_Update* message has only to be flooded in the old and new clusters of the moving node n_o. While the *Leave* message should be flooded in $Nbr(n_o)$ and the old cluster C_o of the leaving node n_o.

For our CBST purposes, however, the *Clust_Update* and the *Leave* messages should be flooded in all neighboring clusters of node n_o. However, this incurs a very limited number of messages, and this because of the two following reasons:

- The number of neighbor clusters of node n_o is supposed to be very small since the clustering metric is node connectivity. Nodes that are neighbors are preferred to be partners in the same cluster, which is taken into consideration by the *SCM* gain computation.

- The *SDC* protocol has a stable performance in term of controlling the cluster size. Thus the flooding within a given cluster is well controlled. Besides, the solution we have proposed for dynamically adjusting the TTL value of the leave message reduces at maximum the number of induced messages.

4.7 Experimental Results

We validated *CBST* through event-driven simulations, which are commonly used to evaluate the performance of large scale P2P systems. Simulations allow controlling system parameters, and thus studying their impact on overall system performance. Furthermore, our performance evaluation consists mainly in quantifying the trade-off between search accuracy and bandwidth consumption. The first is measured in terms of the number of visited nodes, while the second is measured in terms of the number of exchanged messages in the system. Network parameters such as latency and bandwidth do not interfere with these measurements. Thus, simulation results are supposed to give a rough estimation of real values that might be obtained from real implementations.

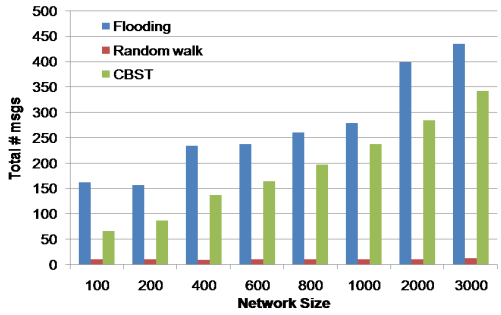

Figure4.7: Query cost vs. network size

4.7.1 Simulation Setup

We used the SimJava package [50] and the BRITE universal topology generator [2] to generate power law P2P networks, with an average node degree of 4. Note that the SDC protocol has been tested over both random and power-law topologies. However, the latter yield better performance regarding the overhead traffic, which is required for maintaining good clustering accuracy, and the cluster size control. On the other hand, recent studies (e.g. [133], [101]) have shown that many real-life networks (e.g. social networks, P2P networks) have common characteristics, including power law degree distributions.

In our experiments, we first generate network topologies with sizes varying between 100 and 3000. Then, we run the SDC protocol to establish a connectivity-based clustering scheme over the generated networks. The value of D, which is used to control the cluster size is set to 3, as considered in [86]. Finally, we implement our search technique over these clustered networks in both static and dynamic settings.

4.7.2 Perfomance in Static Systems

In this set of experiments, we assume that all nodes remain connected to the system, the clustering scheme is stable, and all routing tables are in a valid state. Under these assumptions, we quantify the trade-off between search accuracy and bandwidth consumption. We compare $CBST$ against flooding and random walk [136] techniques. The comparison to flooding techniques is very relevant for evaluating search accuracy, since they provide the highest accuracy values (for a given TTL). However, they produce a huge traffic overhead in the network. Thus, we choose to compare $CBST$ to the random walk technique, which is a good representant of the blind techniques that have been proposed to overcome the problem of bandwidth consumption. In these experiments, a query Q is associated with a TTL value of 3.

4.7.2.1 Search accuracy vs. Bandwidth Consumption

Figure 4.7 depicts the total number of messages exchanged for propagating a query Q, while Figure 4.8 depicts the number of visited nodes, in function of the network size.

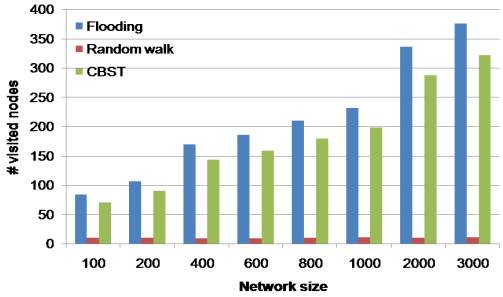

Figure4.8: Number of visited nodes vs. network size

We can observe that the random walk approach reduces very significantly the number of exchanged messages. In fact, a requesting node n sends out k query messages to an equal number of randomly chosen neighbors. Each of these messages is forwarded to a randomly chosen neighbor at each step. In our experiments, we use a high value of k such that a query originator sends the query message to all its neighbors. However, as we can see in Figure 4.8, the search accuracy is also very reduced. A very limited number of nodes are visited by the query Q.

For our $CBST$ technique, Figure 4.7 shows that the number of exchanged messages per query is significantly reduced. For instance, this query cost is reduced by a factor of 2 for a network of 200 nodes. However, in return, $CBST$ incurs a small decrease in search accuracy compared to flooding (Figure 4.8). This result is due to the fact that our approach focuses on eliminating redundant messages, which optimizes the usage of network resources without affecting search accuracy.

4.7.2.2 Influence of TTL

In this set of experiments, we fix the network size to 100 and we vary the value of TTL between 2 and 5. Figure 4.9 gives the number of redundant messages for both flooding and $CBST$ techniques. We can see that, as long as TTL is less than the maximum value of cluster diameter D, the $CBST$ produces a very limited number of redundant message. Moreover, we note that in our approach, a redundant message may not be discarded, and thus may contribute to improve search accuracy. To illustrate, a given node may receive a first query message as being a partner in a cluster, and another message (for the same query) as being a node locating on the path leading to a new cluster. Once TTL exceeds the cluster diameter, the number of $CBST$ redundant messages increases slightly. However, this number is much smaller than the one produced by flooding. For instance, the number of redundant messages is limited to $3,3$ for a TTL value of 5.

4.7.3 Performance in Dynamic Systems

The previous results have shown that $CBST$ achieves a very good trade-off between search accuracy and bandwidth consumption. However, it is very important to study the

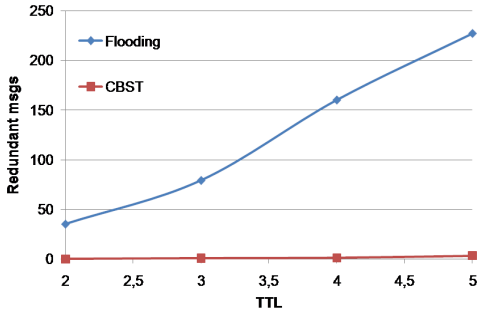

Figure4.9: Influence of TTL

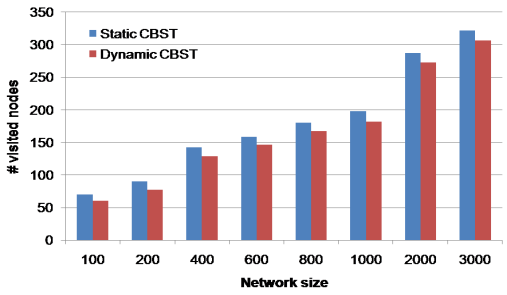

Figure4.10: Number of visited nodes vs. network size

performance of $CBST$ in dynamic P2P systems. In order to propagate a query Q, $CBST$ uses the cluster-based routing tables which may be in an invalid state while recovering from a node arrival/departure.

We assume that 10% of the network size are initially disconnected (i.e. OFF state). Then, nodes start to leave the system at a rate τ. We consider that each node departure is followed by the arrival of another node, such that the total number of nodes remains constant in the system. Simultaneously, users issue queries with a uniform query rate Q_r.

In a first set of experiments, we set the rate τ of node connection/disconnection and the query rate Q_r to 0.1, i.e. 1 event (connection/disconnection or query) per 10 minute. Current file-sharing P2P systems are well characterized by such rate values, as well as by the ratio τ/Q_r value of 1.

Figure 4.10 depicts the number of visited nodes in function of the network size, in both static and dynamic settings. We can observe that node dynamicity results in small degradation of the search accuracy of $CBST$. This degradation is due to the stale entries in routing tables used while propagating a query Q. However, as discussed in Section 4.5.1.2, the node arrival/departure are localized events, and thus only queries that are issued in the same locality are affected. Because of that, we can see also that the number of visited nodes is reduced by a constant factor, which is independent of the network size.

Since the rate ratio Q_r/τ is quited related to the nature of P2P applications, we choose to study the search accuracy of $CBST$ while varying the value of that ratio. Here, the network size is fixed to 100 nodes. Figure 4.11 shows that, for a given query rate, the

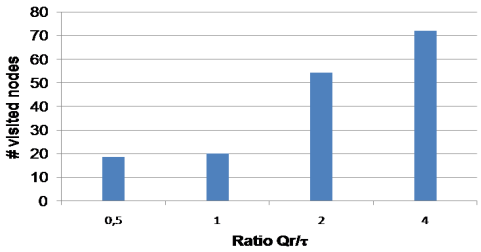

Figure4.11: Number of visited nodes vs. rate ratio

more stable is the network, the higher is the search accuracy. For instance, $CBST$ can improve the search efficiency of database P2P applications, which are characterized by a higher stability.

4.8 Conclusion

In this chapter, we studied how the search efficiency can be improved by clustering a P2P network based on node connectivity. Our solution does not impose a specific network structure or require indexes over the shared data. Instead, it simply relies on inherent clustering patterns that can be extracted from the underlying topologies.

We proposed a Cluster-Based Search Technique (CBST), which is implemented over a connectivity-based clustering protocol. A connectivity-based clustering protocol aims to discover the natural organization of nodes, based on their connectivity. Thus, it delimits the boundaries of sub-graphs (i.e. clusters) which are loosely connected and in which nodes are highly connected.

The CBST technique works as follows. Based on a local knowledge about the network clustering in its neighborhood, each node maintains information about the *local links* to nodes in its cluster (intra-cluster information), as well as about *global links* connecting its cluster to other reachable clusters (inter-cluster information). The intra-cluster routing information is equivalent to a spanning tree, rooted at that node and covering its partners (i.e. the nodes that belong to its cluster). These information are efficiently gathered and maintained in a cluster-based routing table. The benefits in query routing are two folds. First, a query Q is efficiently disseminated in a given cluster, using the spanning tree of the first node contacted in that cluster. Second, the query messages between clusters are restricted to those traversing the global links specified by the querying node.

Extensive simulations demonstrated the efficiency of the $CBST$ technique compared to pure flooding and random walk routing techniques. In fact, the performance of such blind search techniques is generally quantified by the pair (M, P), where M is the total number of exchanged messages, and P is the total number of visited nodes. The search accuracy of these techniques depends on the number of visited nodes P. To increase the probability of finding a replica of the requested object, these techniques tend to propagate the query to a large number of nodes, since they do not dispose of any information related

to object locations. However, the main issue is to achieve a good trade-off between search accuracy and bandwidth consumption. Interesting simulation results showed that the $CBST$ technique allows to eliminate the number of redundant messages $M - P$, which unnecessarily overload the system. In return, only a small decrease in the number of visited nodes per query is incurred. The performance of $CBST$ has been studied in both static and dynamic settings. The effect of the nature of P2P applications, which is characterized by the ratio of node connection/disconnection rate to query generation rate, is also discussed.

Chapter 5

Conclusion

In this chapter, we summarize our main contributions. We also discuss some future directions of research for data sharing in P2P systems.

5.1 Main Contributions

In this thesis, we have addressed the issue of supporting advanced P2P applications with both data localization and summarization techniques. Our contributions are the following.

First, we surveyed data sharing P2P systems. We mainly focused on the evolution from simple file-sharing systems, with limited functionalities, to Peer Data Management Systems (PDMS) that support advanced applications with more sophisticated data management techniques. Advanced P2P applications are dealing with semantically rich data (e.g. XML documents, relational tables), using a high-level SQL-like query language. We started our survey with an overview over the existing P2P network architectures. Then, we presented the data indexing and schema management techniques that have been proposed in the P2P literature. We concluded by discussing the techniques proposed for processing complex queries (i.e. range and join queries). Complex query facilities are necessary to support advanced applications with a high level of search expressiveness. This last part pointed out the lack of querying techniques that allow to approximately answer a query based on data descriptions. These techniques, however, are very interesting for collaborative and decision-support applications.

Second, we proposed a solution for integrating a data summarization technique into P2P systems. Our approach for data summarization is based on SaintEtQ: an online linguistic approach for database summarization [51], [126], [94]. SaintEtiQ has been extensively studied in centralized environments, and exhibits many salient features such as robustness and scalability. The produced summaries are synthetic, multidimensional views over relational database. The novelty of our proposal relies on the double exploitation of these summaries in distributed P2P systems. First, as semantic indexes, they support locating relevant nodes based on data descriptions. Second, due to their intelligibility, they can be directly queried and approximately answer a query without the need for exploring original data.

In the first part of this contribution, we worked in the context of APPA (Atlas Peer-to-Peer Architecture), a P2P data management system which has been developed by our team been building over the last 5 years [120], [93]. The main objective of APPA is to provide high level services for advanced applications. To deal with semantically rich data, APPA supports decentralized schema management and uses novel solutions for query processing, persistent data management with updates, and data replication with semantic-based reconciliation.

In our work, we proposed PeerSum, a new service for managing data summaries [58], [63]. First, we defined a summary model that deals with the dynamic and autonomous nature of P2P systems. This model architecture is characterized by an incremental mechanism of summary construction. Each peer maintains a local summary LS of its own database. Then, peers which are willing to cooperate will exchange and merge summaries, in order to build a Global Summary GS over their shared data. The latter is characterized by a continuous evolution in term of coverage, i.e. the cooperation between two sets of peers, each having constructed a global summary, will result in a higher-coverage one. Second, we defined efficient techniques for global summary creation and maintenance, based on APPA's services. In particular, we assumed that a common storage for global summaries is provided by APPA.

The second part has been done in hierarchical P2P networks [60]. Here, we studied how summaries can be efficiently distributed in P2P systems, which has been abstracted when relying on APPA's services. The idea is to exploit the node heterogeneity in hierarchical (superpeer) networks. Hence, the architecture of our summary model is characterized by 2-summary levels. The first level is provided by the set of local summaries maintained at different peers. The second is obtained by materializing a set of global summaries as follows. The network is organized into *domains*, where a domain is defined as being the set of a supernode and its associated leaf nodes. In a given domain, peers cooperate to maintain a global summary over their shared data, which is stored at the corresponding superpeer. The issue of organizing the network into domains, i.e. how the superpeers are selected and other peers are grouped around them in a fully decentralized manner, is discussed [59], [61]. Then, we presented efficient algorithms for managing summaries in a given domain.

Finally, we proposed a query processing mechanism which relies on the interrogation of available summaries. Our performance is evaluated through a cost model and a simulation model. The simulation results have shown that our solutions allow to significantly reduce the query cost, without incurring high costs of summary update.

At the P2P network layer, we exploited the characteristics of the overlay topology, namely its clustering features, in order to reduce the traffic overhead. This allows to improve the performance of P2P systems, irrespective of the employment of techniques relying on data semantics at the application layer. In fact, the efficiency of such techniques depends on the application and the nature of exchanged data. Besides, they generally have direct implications on the underlying network, such in data indexing schemes, or additional requirements for maintaining a new overlay built on top of the P2P network, as in semantic clustering schemes. A semantic clustering may propose the creation of

new overlays in which semantically related peers are connected to each others. In other terms, such a clustering strives to introduce structure, which is different from discovering inherent structure of the P2P overlay, as we are considering in our work. On the other hand, in the context where such techniques are not employed, or their efficiency degrades (depending on the application), the solution is to resort to flooding-based techniques. In reality, the flooding approach is still a fundamental building block of unstructured P2P systems. It represents the natural way for exchanging messages between nodes which are connected in an ad hoc fashion.

In our work, we proposed a search technique, which is implemented over a connectivity-based clustering protocol, in order to reduce the number of query messages generated by flooding-based algorithms. A connectivity-based clustering protocol aims to discover the natural organization of nodes, based on their connectivity. Thus, it delimits the boundaries of sub-graphs (i.e. clusters) which are loosely connected, and in which nodes are highly connected. In the P2P literature, two main protocols have been proposed, i.e. the Connectivity-based Distributed node Clustering (CDC) [121], and the SCM-based Distributed Clustering (SDC) [86]. These works have been introduced by arguing the benefits of such a clustering from the search performance point of view. However, none of them has proposed an appropriate routing protocol, or provided analytical models or experimental results on how their clustering schemes contribute to reduce the bandwidth consumption. The main focus was on the clustering scheme accuracy, i.e. using the Scale Coverage Measure (SCM), and its maintenance against node dynamicity.

Our Cluster-Based Search Technique (CBST) works as follows. Based on a local knowledge about the network clustering in its neighborhood, each node maintains information about *local links* to nodes in its cluster, as well as about *global links* connecting its cluster to other reachable clusters. The intra-cluster routing information is equivalent to a spanning tree, rooted at that node and covering its partners (i.e. the nodes that belong to its cluster). These information are efficiently gathered and maintained in a cluster-based routing table.

The benefits in query routing are two folds. First, a query Q is efficiently disseminated in a given cluster, using the spanning tree of the first node contacted in that cluster. Second, the query messages between clusters are restricted to those traversing the global links specified by the querying node. Extensive simulations have demonstrated the efficiency of the $CBST$ technique compared to pure flooding and random walk routing techniques.

5.2 Future Work

In this thesis, we have proposed new techniques for an efficient data sharing in P2P systems, as well as for sharing synthesized information which is disseminated within the ever increasing amount of available data. First, we have proposed a novel semantic indexing technique that relies on linguistic data summarization. Second, we have provided the missing proof of the efficiency of cluster-based search technique. By simply exploiting

the clustering features of the P2P network layer, the network traffic can be significantly reduced. At the end of this work, we present a list of directions we plan to pursue in the near future.

Using linguistic summaries for anonymized data sharing. As discussed in Chapter 2, the linguistic summaries maintained in a P2P system allow to locate relevant nodes based on their data descriptions. Besides, these summaries are intelligible data representations whose structure respect the original data schema. So, they can be directly queried to return approximate answers which are provided from their intentional descriptions. However, more interestingly, one can notice that these summaries can serve data anonymization purposes. To illustrate, a given hospital which is participating to a medical collaborative application may first perform an obfuscation of its database through the generation of a local summary. Then, the different participants exchange and share such intelligible summaries, represented in a higher abstraction level. This allow to learn the "essential" from a given database without revealing personal information about patients. As future work, we intend to study how these summaries can be also used to make a data source anonymization. In fact, due to the peer-extent information, a peer may reveal the source providing a given summary while executing the summary update algorithm. In some cases, participants may also prefer hiding the characterization of their data. Hence, a given participant can exploit data summaries of other participants without being able to precise which source is providing a specified summary.

Introducing semantic clustering to our indexing scheme. In our summary model for hierarchical P2P networks, we have assumed that peers are grouped around high-connectivity nodes. However, to better exploit the *data locality* property of most of current P2P systems, we intend to study the organization of nodes based on similarity between their summaries. Ongoing works on the SaintEtiQ model aim to define a similarity distance between summaries. However, such a proposal requires a complete study of the obtained clustering scheme, i.e. the number of clusters, the cluster sizes, the stability against node dynamics.

Introducing indexing to our clustering scheme. In our cluster-based search technique for unstructured P2P systems, we have considered that a given node maintains a spanning tree over its own cluster and information about paths to other reachable clusters. We have demonstrated that the redundancy in query messages is completely eliminated within clusters, but a very limited number of redundant messages are still encountered between clusters. While discussing the search accuracy, we have implicitly supposed that data are uniformly distributed in the P2P system. More studies and comparison to related works should be made to evaluate our proposal in the context of other non-uniform data distributions. Besides, as future work, we plan to elaborate more efficient inter-cluster routing techniques. As an immediate proposal, we may think to weight the global (inter-cluster) links according to their contribution to successful searches. Data indexing schemes can also be employed (DHTs or semantic-based indexes) to ensure an efficient

deterministic search. For instance, consider that each cluster is represented by a node in a given DHT. Thus, the DHT lookup protocol ensures that each cluster containing relevant data according to a user query will be visited. Then, the $CBST$ technique allows to efficiently propagate the query within visited clusters, with a minimum number of messages.

Bibliography

[1] http://www.bittorrent.com.

[2] http://www.cs.bu.edu/brite/.

[3] http://www.gnutella2.com.

[4] http://www.gnutella.com.

[5] http://www.kazaa.com.

[6] http://www.napster.com.

[7] http://www.snomed.org/snomedct.

[8] http://www.w3.org/2001/sw/.

[9] A.Crespo and H.G.Molina. Routing indices for peer-to-peer systems. In Proc. of the 28 tn Conference on Distributed Computing Systems, 2002.

[10] A.Crespo and H.G.Molina. Semantic overlay networks for p2p systems. Technical report, Computer Science Department, Stanford University, 2002.

[11] A.Doan, J.Madhavan, R.Dhamankar, P.Domingos, and A.Halevy. Learning to match ontologies on the semantic web. The VLDB Journal, 12(4):303–319, 2003.

[12] A.Gupta, D.Agrawal, and A.El-Abbadi. Approximate range selection queries in peer-to-peer systems. In CIDR, 2003.

[13] A.Iamnitchi, M.Ripeanu, and I.Foster. Locating data in (small-world?) peer-to-peer scientific collaborations. In IPTPS, pages 232–241, 2002.

[14] R. Akbarinia, E. Pacitti, and P. Valduriez. Reducing network traffic in unstructured p2p systems using top- queries. Distributed and Parallel Databases, 19(2-3):67–86, 2006.

[15] R. Akbarinia, E. Pacitti, and P. Valduriez. Best position algorithms for top-k queries. In VLDB, pages 495–506, 2007.

[16] R. Akbarinia, E. Pacitti, and P. Valduriez. Data currency in replicated dhts. In SIGMOD Conference, pages 211–222, 2007.

[17] R. Akbarinia, E. Pacitti, and P. Valduriez. Processing top-k queries in distributed hash tables. In Euro-Par, pages 489–502, 2007.

[18] A. D. Amis and R. Prakash. Load-balancing clusters in wireless ad hoc networks. In ASSET '00: Proceedings of the 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'00), 2000.

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

[20] A.Rowstron and P.Druschel. Pastry: Scalable decentralized object location and routing for large-scale peer-to-peer systems. In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001.

[21] A.Rowstron and P.Druschel. Storage management and caching in PAST, a large–scale, persistent peer-to-peer storage utility. In Proc.SOSP, 2001.

[22] A.Shoshani. OLAP and statistical databases: Similarities and differences. In Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pages 185–196. ACM Press, 1997.

[23] A.Singla and C.Rohrs. Ultrapeers: another step towards gnutella scalability. Technical report, 2002.

[24] B.Arai, G.Das, D.Gunopulos, and V.Kalogeraki. Approximating aggregation queries in peer-to-peer networks. In ICDE, 2006.

[25] M. Bawa, G. S. Manku, and P. Raghavan. Sets: search enhanced by topic segmentation. In SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, pages 306–313, 2003.

[26] B.Babcock, S.Chaudhuri, and G.Das. Dynamic sample selection for approximate query processing. In SIGMOD, 2003.

[27] B.Cooper and H-G.Molina. Ad hoc, self-supervising peer-to-peer search networks. ACM Trans. Inf. Syst., 23(2):169–200, 2005.

[28] J. L. Bentley. Multidimensional binary search trees used for associative searching. Commun. ACM, 18(9):509–517, 1975.

[29] B.Maniymaran, M.Bertier, and A-M.Kermarrec. Build one, get one free: Leveraging the coexistence of multiple p2p overlay networks. In Proc of the 27th International Conference on Distributed Computing Systems ICDCS, page 33, Washington, DC, USA, 2007. IEEE Computer Society.

[30] P. Bosc, D. Dubois, and H. Prade. Fuzzy functional dependencies and redundancy elimination. JASIS, 49(3):217–235, 1998.

[31] P. Bosc, O. Pivert, and L. Ughetto. On data summaries based on gradual rules. In Fuzzy Days, pages 512–521, 1999.

[32] B.Overeinder, E.Posthumus, and F.Barazier. Integrating p2p networking and computing in the agent scape framework. In Proc. second IEEE Conf on P2P computing, 2002.

[33] T. Bu and D. F. Towsley. On distinguishing between internet power law topology generators. In INFOCOM, 2002.

[34] B.Yang and H-G.Molina. Improving search in peer-to-peer networks. In Proc of the 22 nd International Conference on Distributed Computing Systems (ICDCS), 2002.

[35] B.Yang and H.G.Molina. Comparing hybrid peer-to-peer systems. In Proc VLDB, 2001.

[36] B.Zhao, J.Kubiatowicz, and A.Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, Computer Science Division, U. C.Berkeley, 2001.

[37] B.Zhao, L.Huang, J.Stribling, S.Rhea, A.Joseph, and J.Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications, 22:41–53, 2004.

[38] M. Castro, M. B. Jones, A. marie Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In Infocom'03, 2003.

[39] Y.-H. Chu, S. Rao, S. Seshan, and H. Zhang. A case for end system multicast. IEEE Journal on Selected Areas in Communications, 20(8), 2002.

[40] D. Comer. Ubiquitous b-tree. ACM Comput. Surv., 11(2):121–137, 1979.

[41] C.Schmidt and M.Parashar. Enabling flexible queries with guarantees in p2p systems. IEEE Internet Computing, 08(3):19–26, 2004.

[42] C.Tang, Z.Xu, and M.Mahalingam. Peersearch: Efficient information retrieval in peer-to-peer networks. Technical Report HPL-2002-198, HP Labs, 2002.

[43] J. C. Cubero, J. M. Medina, O. Pons, and M. A. V. Miranda. Data summarization in relational databases through fuzzy dependencies. Information Sciences, 121(3-4):233–270, 1999.

[44] M. E. Dick, V. Martins, and E. Pacitti. A topology-aware approach for distributed data reconciliation in p2p networks. In Euro-Par, pages 318–327, 2007.

[45] D.Malkhi, M.Naor, and D.Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In Proc of the twenty-first annual symposium on Principles of distributed computing, pages 183–192, 2002.

[46] D.Milojicic and *et al.* Peer-to-peer computing. Technical report, HP labs, 2002.

[47] D.Papadias, Y.Tao, G.Fu, and B.Seeger. An optimal and progressive algorithm for skyline queries. In ACM SIGMOD, pages 467–478, 2003.

[48] F.Cuenca-Acuna, C.Peery, R.Martin, and T.Nguyen. Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. In HPDC-12, 2003.

[49] Y. Fernandess and D. Malkhi. K-clustering in wireless ad hoc networks. In POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing, pages 31–37, 2002.

[50] F.Howell and R.McNab. Simjava: a discrete event simulation package for java with the applications in computer systems modeling. In Int. Conf on Web-based Modelling and Simulation, San Diego CA, Society for Computer Simulation, 1998.

[51] G.Raschia and N.Mouaddib. A fuzzy set-based approach to database summarization. Fuzzy sets and systems 129(2), pages 137–162, 2002.

[52] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. J. Data Mining and Knowledge Discovery, 1(1):29–53, 1997.

[53] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? In WebDB Workshop on Databases and the Web, 2001.

[54] G.Skobeltsyn, T.Luu, I.-P. Žarko, M.Rajman, and K.Aberer. Query-driven indexing for scalable peer-to-peer text retrieval. Infoscale, page 14, 2007.

[55] G.Wiederhold. Mediators in the architecture of future information systems. IEEE Computer, 25:38–49, 1992.

[56] J. Han, Y. Fu, Y. Huang, Y. Cai, and N. Cercone. Dblearn: A system prototype for knowledge discovery in relational databases. In Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994, page 516. ACM Press, 1994.

[57] S. B. Handurukande, A.-M. Kermarrec, F. L. Fessant, and L. Massoulie.

[58] R. Hayek, G. Raschia, P. Valduriez, and N. Mouaddib. Design of peersum: A summary service for p2p applications. In GPC, pages 13–26, 2007.

[59] R. Hayek, G. Raschia, P. Valduriez, and N. Mouaddib. Peersum: Summary management in p2p systems. In Bases de Donnes Avances (BDA), 2007.

[60] R. Hayek, G. Raschia, P. Valduriez, and N. Mouaddib. Summary management in p2p systems. In Int. Conf. on Extending Database Technology (EDBT), pages 16–25, Nantes, France, 2008.

[61] R. Hayek, G. Raschia, P. Valduriez, and N. Mouaddib. Gestion de rsums de donnes dans les systmes pair--pair. Ingnierie des Systmes dInformation (ISI), Numro spcial Networking and Information Systems, to appear.

[62] R. Hayek, G. Raschia, P. Valduriez, and N. Mouaddib. Handbook of Peer-to-Peer Networking, chapter Data Localization and Description Through Summaries in P2P Collaborative Applications. Springer, to appear.

[63] R. Hayek, G. Raschia, P. Valduriez, and N. Mouaddib. PeerSum: Summary management in P2P systems. Int. Journal of Pervasive Computing and Communications (IJPCC), Special Issue on Towards Merging Grid and Pervasive Computing, to appear.

[64] J. M. Hellerstein. Toward network data independence. SIGMOD Rec, 32:200–3, 2003.

[65] H.Jagadish, B.Ooi, and Q.Vu. Baton: A balanced tree structure for peer-to-peer networks. In VLDB, 2005.

[66] H.Jagadish, B.Ooi, Q.Vu, R.Zhang, and A.Zhou. Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In ICDE, page 34, 2006.

[67] H.Jagadish, R.Ng, B.Ooi, and A.Tung. Itcompress: An iterative semantic compression algorithm. In 20th International Conference on Data Engineering, page 646, 2004.

[68] H.Shen, Y.Shu, and B.Yu. Efficient semantic-based content search in p2p network. IEEE Transactions on Knowledge and Data Engineering, 16(7), 2004.

[69] I.Chrysakis, D.Plexousakis, I.Chrysakis, and D.Plexousakis. Semantic query routing and distributed top-k query processing in peer-to-peer networks. Technical report, Department of Computer Science, University of Crete, 2006.

[70] I.Clarke, S.Miller, T.Hong, O.Sandberg, and B.Wiley. Protecting free expression online with freenet. IEEE Internet Computing, 6(1):40–49, 2002.

[71] Y. Ioannidis. The history of histograms (abridged). In VLDB '2003: Proceedings of the 29th international conference on Very large data bases, pages 19–30. VLDB Endowment, 2003.

[72] I.Stoica, R.Morris, D.Karger, M.F.Kaashoek, and H.Balakrishnan. Chord: A scalabale peer-to-peer lookup service for internet applications. In Proc ACM SIGCOMM, 2001.

[73] I.Tartinov and *et al.* The Piazza peer data management project. In SIGMOD, 2003.

[74] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: a decentralized peer-to-peer web cache. In PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing, pages 213–222, 2002.

[75] J.Kubiatowicz, D.Bindel, Y.Chen, S.Czerwinski, P.Eaton, D.Geels, R.Gummadi, S.Rhea, H.Weatherspoon, C.Wells, and B.Zhao. Oceanstore: an architecture for global-scale persistent storage. SIGOPS Oper. Syst. Rev., 34(5):190–201, 2000.

[76] h. JXTA for J2ME.

[77] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, pages 654–663, 1997.

[78] K.Hose, C.Lemke, and K.Sattler. Processing relaxed skylines in pdms using distributed data summaries. In CIKM, pages 425–434, 2006.

[79] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan. A cluster-based approach for routing in dynamic networks. SIGCOMM Comput. Commun. Rev., 27(2):49–64, 1997.

[80] K.Thompson and P.Langley. Concept formation in structured domains. In Concept formation: Knowledge and experience in unsupervised learning, pages 127–161. Morgan Kaufmann.

[81] L.Adamic and *et* al. Search in power law networks. Physical Review E, 64:46135–46143, 2001.

[82] L.A.Zadeh. Fuzzy sets. Information and Control, 8:338–353, 1965.

[83] L.A.Zadeh. Concept of a linguistic variable and its application to approximate reasoning-I. Information Systems, 8:199–249, 1975.

[84] D. H. Lee and M. H. Kim. Database summarization using fuzzy ISA hierarchies. IEEE Trans. on Systems, Man and Cybernetics-Part B: Cybernetics, 27:68–78, fvrier 1997.

[85] M. Lenzerini. Data integration: a theoretical perspective. In PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 233–246, 2002.

[86] Y. Li, L. Lao, and J.-H. Cui. Sdc: A distributed clustering protocol for peer-to-peer networks. In Networking, pages 1234–1239, 2006.

[87] W. Litwin, M.-A. Neimat, and D. A. Schneider. Lh* - linear hashing for distributed files. In SIGMOD Conference, pages 327–336, 1993.

[88] W. Litwin, M.-A. Neimat, and D. A. Schneider. Rp*: A family of order preserving scalable distributed data structures. In VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases, pages 342–353, 1994.

[89] W. Litwin, H. Yakouben, and T. Schwarz. Lh*rsp2p: a scalable distributed data structure for p2p environment. In NOTERE '08: Proceedings of the 8th international conference on New technologies in distributed systems, pages 1–6, 2008.

[90] X. Liu, Y. Liu, L. Xiao, F.-L. M. Ni, and X. Zhang. Location awareness in unstructured peer-to-peer systems. IEEE Trans. Parallel Distrib. Syst., 16(2):163–174, 2005.

[91] L.Ramaswamy, B.Gedik, and L.Liu. A distributed approach to node clustering in decentralized peer-to-peer networks. IEEE Transactions on Parallel and Distributed Systems, 16(9):814–829, 2005.

[92] J. Madhavan, P. A. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In ICDE '05: Proceedings of the 21st International Conference on Data Engineering, pages 57–68, 2005.

[93] V. Martins, R. Akbarinia, E. Pacitti, and P. Valduriez. Reconciliation in the appa p2p system. In ICPADS (1), pages 401–410, 2006.

[94] M.Bechchi, G.Raschia, and N.Mouaddib. Merging distributed database summaries. In ACM Sixteenth Conference on Information and Knowledge Management (CIKM), 2007.

[95] M.Cai and M.Frank. Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network. In WWW, pages 650–657, 2004.

[96] M.Cai, M.Frank, J.Chen, and P.Szekely. Maan: A multi-attribute addressable network for grid information services. In GRID, 2003.

[97] M.Castro, M.Costa, and A.Rowstron. Should we build gnutella on a structured overlay? SIGCOMM Comput. Commun. Rev., 34(1):131–136, 2004.

[98] A. B. Mcdonald and T. Znati. A mobility based framework for adaptive clustering in wireless ad-hoc networks. IEEE Journal on Selected Areas in Communications, 17:1466–1487, 1999.

[99] M.Espil and A.Vaisman. Aggregate queries in peer-to-peer olap. In DOLAP, 2004.

[100] M.Gerla, T. J.Kown, and G.Pei. On demand routing in large ad hoc wireless networks with passive clustering. In Proceedings of IEEE WCNC 2000, 2000.

[101] M.Ripeanu, I.Foster, and A.Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. IEEE Internet Computing Journal, 6(1), 2002.

[102] W. Ng, B.Ooi, K-L.Tan, and A.Zhou. Peerdb: A p2p-based system for distributed data sharing. In ICDE, pages 633–644, 2003.

[103] N.Harvey, M.Jones, S.Saroiu, M.Theimer, and A.Wolman. Skipnet: A scalable overlay network with practical locality properties. In USENIX Symposium on Internet Technologies and Systems, 2003.

[104] N.Sarshar, P.Boykin, and V.Roychowdhury. Percolation search in power law networks: Making unstructured peer-to-peer networks scalable. p2p, 00:2–9, 2004.

[105] D. O.Sahin, A.Gupta and A.El-Abbadi. Query processing over peer-to-peer data sharing systems. Technical report, University of California, Santa Barbara, 2002.

[106] H.-Y. Paik, N. Mouaddib, B. Benatallah, F. Toumani, and M. Hassan. Building and querying e-catalog networks using p2p and data summarisation techniques. J. Intell. Inf. Syst., 26(1):7–24, 2006.

[107] P.Bernstein, F.Giunchiglia, A.Kementsietsidis, J.Mylopoulos, L.Serafini, and I.Zaihrayeu. Data management for peer–to–peer computing: A vision. In Proc. of the 5th International Workshop on the Web and Databases (WebDB), 2002.

[108] P.Ganesan, Q.Sun, and H.G.Molina. Adlib: a self-tuning index for dynamic peer-to-peer systems. In Int. Conference on Data Engineering (ICDE), 2005.

[109] P.Kalnis, W.Ng, B.Ooi, D.Papadias, and K.Tan. An adaptive peer-to-peer network for distributed caching of olap results. In SIGMOD, 2002.

[110] P.Kalnis, W.Ng, B.Ooi, and K.Tan. Answering similarity queries in peer-to-peer networks. Inf. Syst., 31(1):57–72, 2006.

[111] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In SPAA, pages 311–320, 1997.

[112] P.Maymounkov and D.Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In Int. Workshop on Peer-to-Peer Systems (IPTPS), pages 53–65, 2002.

[113] P.McBrien and A.Poulovassilis. Defining peer-to-peer data integration using both as view rules. In DBISP2P, pages 91–107, 2003.

[114] H. Prade and C. Testemale. Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries. Inf. Sci., 34(2):115–143, 1984.

[115] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Numerical recipes in C (2nd ed.): the art of scientific computing. Cambridge University Press, New York, NY, USA, 1992.

[116] P.Triantafillou and T.Pitoura. Towards a unifying framework for complex query processing over structured peer-to-peer data networks. In DBISP2P, pages 169–183, 2003.

[117] P.Wu, C.Zhang, Y.Feng, B.Zhao, D.Agrawal, and A.El-Abbadi. Parallelizing skyline queries for scalable distribution. In EDBT, pages 112–130, 2006.

[118] R.Akbarinia, E.Pacitti, and P.Valduriez. Reducing network traffic in unstructured p2p systems using top-k queries. Distrib. Parallel Databases, 19(2-3):67–86, 2006.

[119] R.Akbarinia, E.Pacitti, and P.Valduriez. Processing top-k queries in distributed hash tables. In Euro-Par, pages 489–502, 2007.

[120] R.Akbarinia, V.Martins, E.Pacitti, and P.Valduriez. Design and implementation of appa. In Global Data Management (Eds. R. Baldoni, G. Cortese and F. Davide). IOS press, 2006.

[121] L. Ramaswamy, B. Gedik, and L. Liu. Connectivity based node clustering in decentralized peer-to-peer networks. In P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing, page 66, 2003.

[122] R.Aringhieri, E.Damiani, S.Vimercati, S.Paraboschi, and P.Samarati. Fuzzy techniques for trust and reputation management in anonymous peer-to-peer systems. Journal of the American Society for Information Science and Technology, 57(4), 2006.

[123] D. Rasmussen and R. R. Yager. SummarySQL - a fuzzy tool for data mining. Intelligent Data Analysis, 1:49–58, 1997.

[124] R.Dingledine, M.Freedman, and D.Molnar. The free haven project: distributed anonymous storage service. In International workshop on Designing privacy enhancing technologies, pages 67–95, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[125] R.Huebsch, J.Hellerstein, N.Lanham, B.Thau, L.Shenker, and I.Stoica. Querying the internet with pier. In VLDB, 2003.

[126] R.Saint-Paul, G.Raschia, and N.Mouaddib. General purpose database summarization. In Proc VLDB, pages 733–744, 2005.

[127] E. H. Ruspini. A new approach to clustering. Information and Control, 15:22–32, 1969.

[128] S.Babu, G.Minos, and R.Rajeev. Spartan: A model-based semantic compression system for massive data tables. In Proc. of the 2001 ACM Intl. Conf. on Management of Data (SIGMOD), pages 283–295, 2001.

[129] S.Dongen. A new cluster algorithm for graphs. Technical report, Amsterdam, 1998.

[130] S.Ratnasamy, M.Handley, R.Karp, and S.Shenker. Topologically-aware overlay construction and server selection. In Proceedings of IEEE INFOCOM'02, 2002.

[131] S.Ratnasamy, P.Francis, M.Handley, R.M.Karp, and S.Shenker. A scalable content–addressable network. In SIGCOMM, 2001.

[132] K. Sripanidkulchai, B. M. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In INFOCOM, 2003.

[133] S.Saroiu, P.Gummadi, and S.Gribble. A measurement study of peer-to-peer file sharing systems. In Proc of Multimedia Computing and Networking (MMCN), 2002.

[134] S.Wang, B.Ooi, A.Tung, and L.Xu. Efficient skyline query processing on peer-to-peer networks. In ICDE, 2007.

[135] K. *et* al. P-grid: a self-organizing structured p2p system. SIGMOD Rec., 32(3):29–33, 2003.

[136] Q. *et* al. Search and replication in unstructured peer-to-peer networks. In ACM Int. conference on Supercomputing, 2002.

[137] W. *et* al. Edutella: a p2p networking infrastructure based on rdf. In WWW'02, 2002.

[138] T.Luu, G.Skobeltsyn, F.Klemm, M.Puh, I.-P. Žarko, M.Rajman, and K.Aberer. Alvisp2p: Scalable peer-to-peer text retrieval in a structured p2p network. In Proc VLDB, 2008.

[139] M. Tsangou, S. Ndiaye, M. Seck, and W. Litwin. Range queries to scalable distributed data structure rp*. In Fifth workshop on Distributed Data and Structures (WDAS), 2003.

[140] D. Tsoumakos and N. Roussopoulos. A comparison of peer-to-peer search methods. In Int.Workshop on the Web and Databases (WebDB), pages 61–66, 2003.

[141] U.Guntzer, W.Balke, and W.Kieβling. Optimizing multi-feature queries for image databases. In VLDB, 2000.

[142] J. D. Ullman. Information integration using logical views. In ICDT '97: Proceedings of the 6th International Conference on Database Theory, pages 19–40, 1997.

[143] V.Kalogeraki, D.Gunopulos, and D.Yazti. A local search mechanism for peer-to-peer networks. In Proc CIKM, USA, 2002.

[144] A. Voglozin, G. Raschia, L. Ughetto, and N. Mouaddib. Handbook of Research on Fuzzy Information Processing in Databases, volume 1, chapter From User Requirements to Evaluation Strategies of Flexible Queries in Databases, pages 115–142. 2008.

[145] W.Aiello, F.Chung, and L.Lu. A random graph model for massive graphs. In STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing, pages 171–180, 2000.

[146] W.A.Voglozin, G.Raschia, L.Ughetto, and N.Mouaddib. Querying the SaintEtiQ summaries–a first attempt. In Int.Conf.On Flexible Query Answering Systems (FQAS), 2004.

[147] W.Balke, W.Nejdl, W.Siberski, and U.Thaden. Progressive distributed top-k retrieval in peer-to-peer networks. In ICDE, 2005.

[148] W.Lou and J.Wu. Efficient broadcast with forward node set in clustered mobile ad hoc networks. In Proc. IEEE 11th Conf. Computer Comm. and Networks, 2002.

[149] W.Nejdl and W.Siberski. Design issues and challenges for rdf- and schema-based peer-to-peer systems. SIGMOD Record, 32:2003, 2003.

[150] W.Ng, B.Ooi, and K.Tan. Bestpeer: A self-configurable peer-to-peer system. In ICDE, 2002.

[151] J. Wu and W. Lou. Forward-node-set-based broadcast in clustered mobile ad hoc networks. Wireless Communications and Mobile Computing, 3(2), 2003.

[152] X.Li, Y.J.Kim, R.Govindan, and W.Hong. Multidimensional range queries in sensor networks. In SENSYS, 2003.

[153] R. R. Yager. On linguistic summaries of data. In Knowledge Discovery in Databases, pages 347–366. MIT Press, 1991.

[154] B. Yang, P. Vinograd, and H. Garcia-Molina. Evaluating guess and non-forwarding peer-to-peer search. In ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), pages 209–218, 2004.

[155] Y.Chawathe, S.Ratnasamy, L.Breslau, N.Lanham, and S.Shenker. Making gnutella-like p2p systems scalable. In In Proc. ACM SIGCOMM, 2003.

[156] Y.Halevy, G.Ives, D.Suciu, and I.Tatarinov. Schema mediation for large-scale semantic data sharing. The VLDB Journal, 14(1):68–83, 2005.

[157] T. Zahn, R. Winter, and J. Schiller. Simple and efficient peer-to-peer overlay clustering in mobile, ad hoc networks. In In: Proc. of the 12th IEEE Int Conf. on Networks, 2004.