Data Sharing in P2P Systems

Rabab Hayek, Guillaume Raschia, Patrick Valduriez, Noureddine Mouaddib

Data Sharing in P2P Systems

Rabab Hayek and Guillaume Raschia and Patrick Valduriez and Noureddine Mouaddib

Abstract In this chapter, we survey P2P data sharing systems. 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*.

Rabab Hayek
LINA, 2 rue de la Houssiniere 44322 Nantes France, e-mail: rabab.hayek@univ-nantes.fr

Guillaume Raschia
LINA, 2 rue de la Houssiniere 44322 Nantes France, e-mail: guillaume.raschia@univ-nantes.fr

Patrick Valduriez
INRIA, 2 rue de la Houssiniere 44322 Nantes France, e-mail: Patrick.Valduriez@inria.fr

Noureddine Mouaddib
LINA, 2 rue de la Houssiniere 44322 Nantes France, e-mail: Noureddine.Mouaddib@univ-nantes.fr

1 Introduction

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 [31]. 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 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.

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.

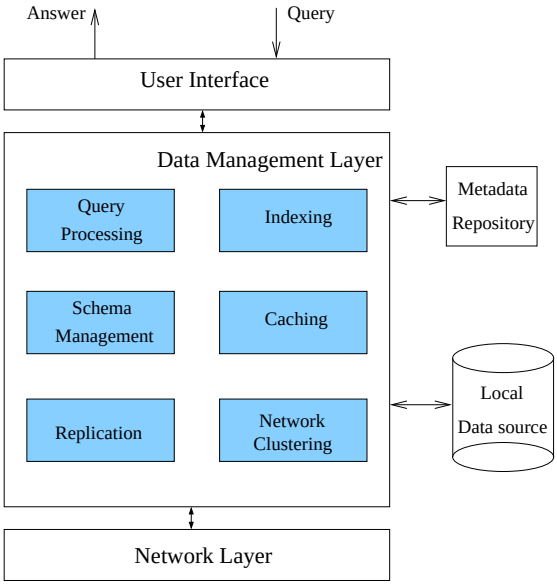

Fig. 1 Peer Generic Architecture

To enhance the search performance, P2P works started to employ *data indexing* techniques (e.g. [6], [82]). 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 3 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 [11, 46, 15].

In Schema-based P2P systems [93], 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. [12], [45]). 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 4.

Network clustering has been also proposed as a viable solution to improve query processing in P2P systems (e.g. [55], [7]). Clustering techniques aim to organize the

network into groups based on some criteria. A clustering criterion may be a physical network parameter (e.g. latency, bandwidth), peer property/behavior (e.g. connectivity, stability), or application-dependent parameter (e.g. similarity of interests). In this chapter, we do not discuss P2P clustering techniques. However, representative works are [81], [83], [7], [56].

Finally, we note that the data management layer presented in Figure 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 require a higher expressiveness level than key-based lookups or keyword searches. Processing complex queries in P2P systems is discussed in Section 5.

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 P2P Networks

P2P systems are application-level virtual networks with their own overlay topologies and routing protocols. The overlay topology defines how the nodes are connected to each other, while the routing protocol defines 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 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 Hybrid Decentralized Architectures

In these networks, a central server facilitates the interaction between nodes by indexing all their shared files (Figure 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 (1*GP*). A well-known example is Napster [4].

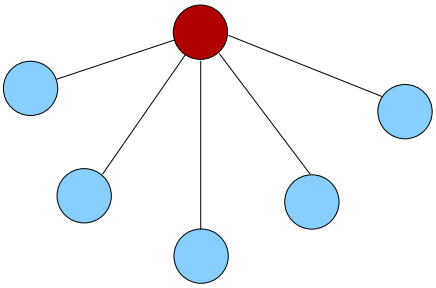

Fig. 2 Hybrid decentralized architecture

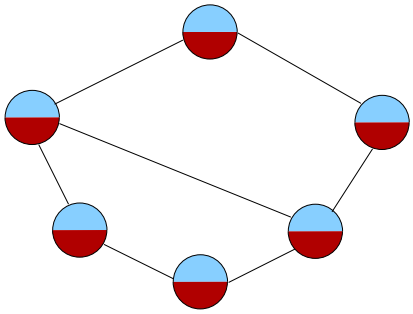

Fig. 3 Pure decentralized architecture

2.1.2 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 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 [11] and routing indices [6]. These mechanisms have direct implications on network scalability.

Representative examples of pure decentralized P2P systems are Gnutella [2], and FreeHaven [79].

2.1.3 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 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 [3], Gnutella2 [1], and Edutella [12]. 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 (2*GP*).

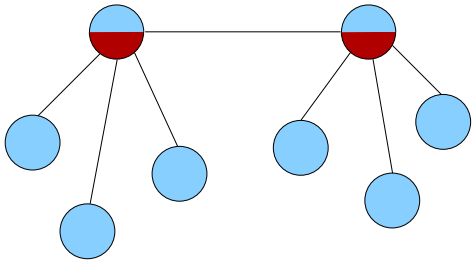

Fig. 4 Partially decentralized architecture

2.2 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 (3*GP*), 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. [44], [82]). Freenet [43] 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.

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

Table 1 A classification of P2P Systems and Infrastructures Based on Network Structure, and Degree of Decentralization [13]

Table 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.

2.3 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 [98]. First, the strict network structure imposes high overhead for handling node join and leave, although some works have defended performance during churn (e.g. [26]). 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 [60] 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 [21], 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 [93]. 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. [20], [39]) 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.

3 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.

3.1 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.

3.1.1 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 [2] 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 [97]. 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 [86].

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 5). Small TTL values reduce the network traffic and the load at peers, but also reduce the chances of a successful search.

Modified BFS [89] 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 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 [11], 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

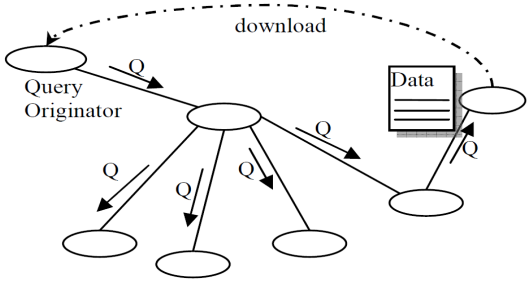

Fig. 5 Example of BFS: the received query is forwarded to all the neighbors

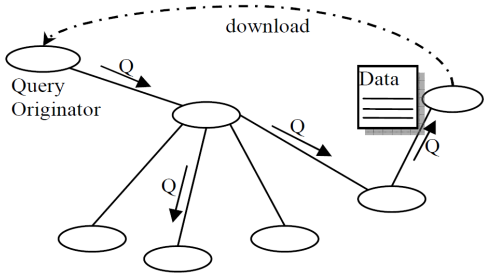

Fig. 6 Example of Modified BFS: the received query is forwarded to a randomly selected set of neighbors

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.

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 [11], 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.

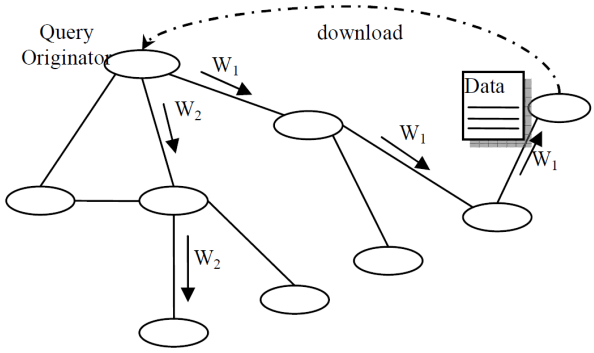

Fig. 7 Example of Random Walks: each received walk is forwarded to only one neighbor

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 [50] 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 [24] 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.

3.1.2 Centralized Index

The index is centralized at dedicated servers, but the described data is distributed. In fact, the centralized schemes [4] 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.

3.1.3 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 [43]. 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.

3.2 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 present the main geometries (i.e. the topology and the associated routing strategies) used for DHT-based systems, and discuss their search efficiency and their robustness.

3.2.1 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 [25] 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.

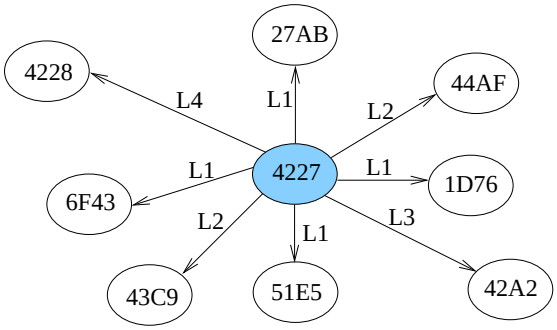

Fig. 8 Tapestry routing mesh from the perspective of a single node. Outgoing neighbor links point to nodes with a common matching prefix. Higher level entries match more digits. Together, these links form the neighbor map [25].

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 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 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 [47]. Pastry [14] 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 [15].

3.2.2 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 [44] 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 [48]. 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.

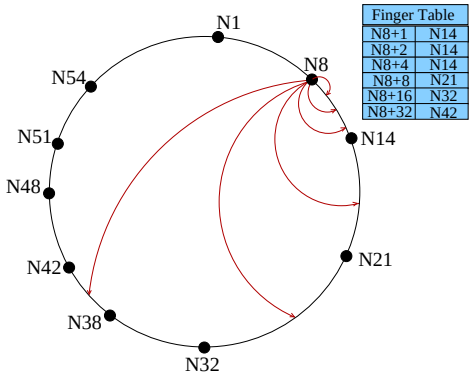

Fig. 9 The *finger table* at node 8 on a Chord ring of 10 nodes, $m = 6$ [44].

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 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.

3.2.3 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 [82] 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 10). When $d = log\ n$, the neighbor sets in CAN are similar to those of a $log\ n$ dimensional hypercube.

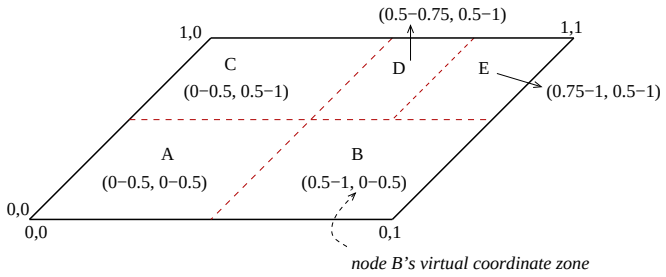

Fig. 10 2-dimensional $[0;1] \times [0;1]$ coordinate space partitioned between 5 CAN nodes [82].

Other DHTs geometries are the *Butterfly* geometry which is used in Viceroy [30], and the XOR geometry which is used by Kademlia [69]. Certainly, two or more geometries can be combined together to provide a hybrid geometry that satisfies better the DHT requirements. To illustrate, Pastry [14] combines the tree and ring geometries in order to achieve more efficiency and flexibility.

3.3 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 5.

3.3.1 Keyword Lookup

Gnutella [2] 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 [16], 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, [24] 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 [6] 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).

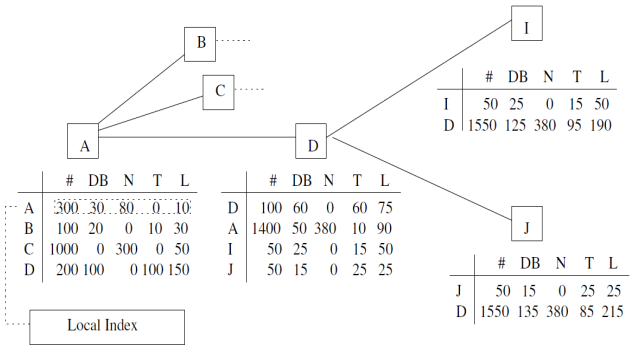

Fig. 11 Example of Routing Indices [6]

Figure 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.

3.3.2 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 [18]. 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 [33] 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 [28] 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 [37]. 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 [85], 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.

3.3.3 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?* [36]. 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 12).

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 4. 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

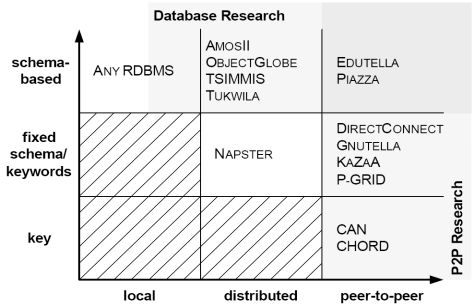

Fig. 12 Schema capabilities and distribution [93]

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 5 discusses complex queries in P2P systems.

4 Schema Management in P2P Systems

Semantic heterogeneity is a key problem in large scale data sharing systems [57]. 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 [38] (see Figure 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 [88].

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 [54]. 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

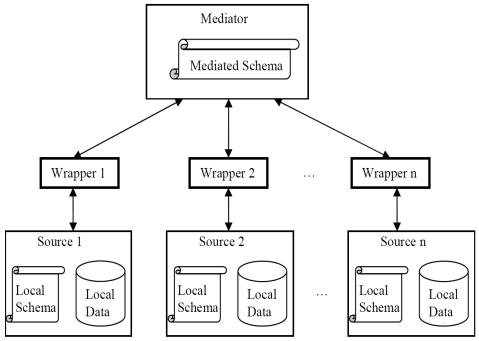

Fig. 13 Schema Mapping using a Global Mediated Schema

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.

4.1 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 [45] follows this approach (see Figure [45]). 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.

The Local Relational Model (LRM) [66] 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

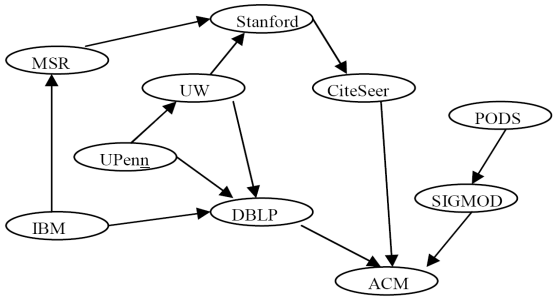

Fig. 14 An Example of Pairwise Schema Mapping in Piazza

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 [10]. 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.

4.2 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 [5]. 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 [8] 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.

4.3 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 [76] 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 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 [70] 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.

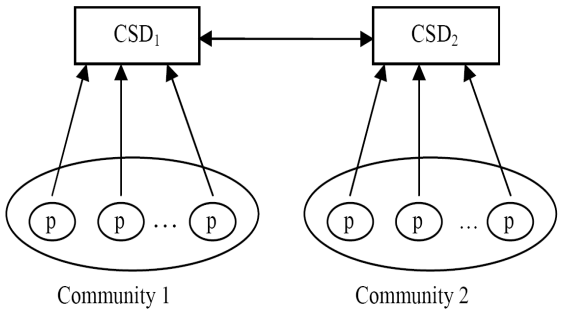

Fig. 15 Common Agreement Schema Mapping in APPA

4.4 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 [62] 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 [12] 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 [99]. 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.

5 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".

5.1 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 three types of complex queries: range, multi-attribute and join 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.

5.1.1 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 [9]. 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* [9] 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 [64] 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 [27]. 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 [63] 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 [10] 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 [40] is a balanced binary search tree with in-level links for efficiency, fault-tolerance, and load-balancing. VBI-tree [41] 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.

5.1.2 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 [80], 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 [72], 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.

5.1.3 Multi-Attributes Queries

There has been some work on multi-attribute P2P queries. The Multi-Attribute Addressable Network (MAAN) [59] 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 [58]. 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.

5.1.4 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) that maintains some patient records in a given hospital[1].

Suppose now that a doctor requires information about young patients diagnosed with Malaria (i.e. the information need). In a conventional SQL query, we must

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

Table 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*

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:

```
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 [51].

The application of gradual predicates, such as the YOUNG predicate presented in Figure 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.

In [77], 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

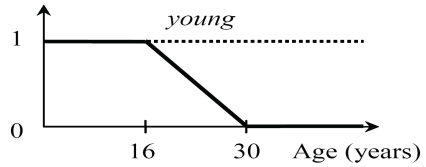

Fig. 16 Gradual predicate on attribute AGE

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 [94]. In [68], 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 [68] 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.

5.2 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.

5.2.1 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 [33] 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 [74]. 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 [92], 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 [12]. 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 [75]. 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.

5.2.2 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.

[73] is the first attempt on progressive processing of skyline queries on a P2P network such as CAN [82]. 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* [32] 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 [84] has proposed an efficient solution for skyline query processing in the context of BestPeer. BestPeer [94] is a P2P platform that supports both structured and unstructured overlays. The solution proposed in [84] is called *Skyline Space Partitioning* (SSP), and is implemented in the BestPeer's structured network, called BATON [40]. 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.

5.3 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 5.1, 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 [49] in which the notion of *relaxed skyline* is introduced, and [42] in which the well-known TA algorithm [87] 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 [17].

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 multi-

ple 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. [19], [95]) 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 ([67], [61]). 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 [17]. 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.

Note that aggregation queries are not flexible, i.e. they are precisely formulated with specific operators. However, the aggregate values returned to the user provide information about tendencies within data. For example, the data cube [35], 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 17).

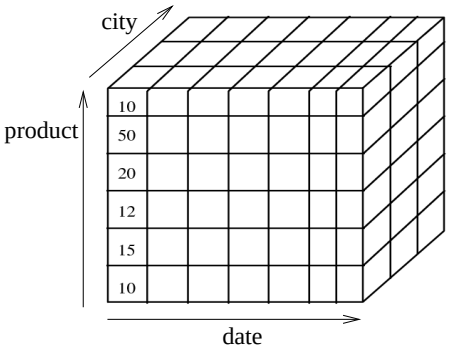

Fig. 17 Data cube

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 [90].

The literature also offers studies of how to express concepts or needs through constructs such as operators or linguistic variables [52]. One of the main challenges of extending query languages is to enrich query formulation without drastically reducing the performance of the query evaluation process. Linguistic summaries, studied by Yager *et al* in [96], 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 [71, 78] use fuzzy quantifiers in addition to linguistic terms to describe the data. For instance, in SummarySQL [78], 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 [23] or by exploiting fuzzy functional dependencies [22, 29]. They produce, in the case of gradual rules of Bosc *et al* [23], 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 [53] and the SaintEtiQ model [34] 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 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 [65, 91], however, such techniques have not been studied in P2P environments yet. The next chapter proposes a solution for managing fuzzy summaries in P2P systems to support DB applications with approximate query facilities.

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

References

1. http://www.gnutella2.com
2. http://www.gnutella.com
3. http://www.kazaa.com
4. http://www.napster.com
5. http://www.w3.org/2001/sw/
6. A.Crespo, H.G.Molina: Routing indices for peer-to-peer systems. In: Proc. of the 28 tn Conference on Distributed Computing Systems (2002)
7. A.Crespo, H.G.Molina: Semantic overlay networks for p2p systems. Tech. rep., Computer Science Department, Stanford University (2002)
8. A.Doan, J.Madhavan, R.Dhamankar, P.Domingos, A.Halevy: Learning to match ontologies on the semantic web. The VLDB Journal **12**(4), 303–319 (2003)
9. A.Gupta, D.Agrawal, A.El-Abbadi: Approximate range selection queries in peer-to-peer systems. In: CIDR (2003)
10. *et* al, K.: P-grid: a self-organizing structured p2p system. SIGMOD Rec. **32**(3), 29–33 (2003)
11. *et* al, Q.: Search and replication in unstructured peer-to-peer networks. In: ACM Int. conference on Supercomputing (2002)
12. *et* al, W.: Edutella: a p2p networking infrastructure based on rdf. In: WWW'02 (2002)
13. Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution technologies. ACM Comput. Surv. **36**(4), 335–371 (2004)
14. A.Rowstron, 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)
15. A.Rowstron, P.Druschel: Storage management and caching in PAST, a large–scale, persistent peer-to-peer storage utility. In: Proc.SOSP (2001)
16. A.Singla, C.Rohrs: Ultrapeers: another step towards gnutella scalability. Tech. rep. (2002)
17. B.Arai, G.Das, D.Gunopulos, V.Kalogeraki: Approximating aggregation queries in peer-to-peer networks. In: ICDE (2006)
18. Bawa, M., Manku, G.S., Raghavan, P.: 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, pp. 306–313 (2003)
19. B.Babcock, S.Chaudhuri, G.Das: Dynamic sample selection for approximate query processing. In: SIGMOD (2003)
20. B.Cooper, H-G.Molina: Ad hoc, self-supervising peer-to-peer search networks. ACM Trans. Inf. Syst. **23**(2), 169–200 (2005)
21. B.Maniymaran, M.Bertier, 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, p. 33. IEEE Computer Society, Washington, DC, USA (2007)
22. Bosc, P., Dubois, D., Prade, H.: Fuzzy functional dependencies and redundancy elimination. JASIS **49**(3), 217–235 (1998)
23. Bosc, P., Pivert, O., Ughetto, L.: On data summaries based on gradual rules. In: Fuzzy Days, pp. 512–521 (1999)
24. B.Yang, H-G.Molina: Improving search in peer-to-peer networks. In: Proc of the 22 nd International Conference on Distributed Computing Systems (ICDCS) (2002)
25. B.Zhao, J.Kubiatowicz, A.Joseph: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. rep., Computer Science Division, U. C.Berkeley (2001)
26. B.Zhao, L.Huang, J.Stribling, S.Rhea, A.Joseph, J.Kubiatowicz: Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications **22**, 41–53 (2004)
27. C.Schmidt, M.Parashar: Enabling flexible queries with guarantees in p2p systems. IEEE Internet Computing **08**(3), 19–26 (2004)

28. C.Tang, Z.Xu, M.Mahalingam: Peersearch: Efficient information retrieval in peer-to-peer networks. Tech. Rep. HPL-2002-198, HP Labs (2002)
29. Cubero, J.C., Medina, J.M., Pons, O., Miranda, M.A.V.: Data summarization in relational databases through fuzzy dependencies. Information Sciences **121**(3-4), 233–270 (1999)
30. D.Malkhi, M.Naor, D.Ratajczak: Viceroy: a scalable and dynamic emulation of the butterfly. In: Proc of the twenty-first annual symposium on Principles of distributed computing, pp. 183–192 (2002)
31. D.Milojicic, *et al*: Peer-to-peer computing. Tech. rep., HP labs (2002)
32. D.Papadias, Y.Tao, G.Fu, B.Seeger: An optimal and progressive algorithm for skyline queries. In: ACM SIGMOD, pp. 467–478 (2003)
33. F.Cuenca-Acuna, C.Peery, R.Martin, T.Nguyen: Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. In: HPDC-12 (2003)
34. G.Raschia, N.Mouaddib: A fuzzy set-based approach to database summarization. Fuzzy sets and systems 129(2) pp. 137–162 (2002)
35. Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., Pirahesh, H.: 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)
36. Gribble, S., Halevy, A., Ives, Z., Rodrig, M., Suciu, D.: What can databases do for peer-to-peer? In: WebDB Workshop on Databases and the Web (2001)
37. G.Skobeltsyn, T.Luu, Žarko, I.P., M.Rajman, K.Aberer: Query-driven indexing for scalable peer-to-peer text retrieval. Infoscale p. 14 (2007)
38. G.Wiederhold: Mediators in the architecture of future information systems. IEEE Computer **25**, 38–49 (1992)
39. Hellerstein, J.M.: Toward network data independence. SIGMOD Rec **32**, 200–3 (2003)
40. H.Jagadish, B.Ooi, Q.Vu: Baton: A balanced tree structure for peer-to-peer networks. In: VLDB (2005)
41. H.Jagadish, B.Ooi, Q.Vu, R.Zhang, A.Zhou: Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In: ICDE, p. 34 (2006)
42. I.Chrysakis, D.Plexousakis, I.Chrysakis, D.Plexousakis: Semantic query routing and distributed top-k query processing in peer-to-peer networks. Tech. rep., Department of Computer Science, University of Crete (2006)
43. I.Clarke, S.Miller, T.Hong, O.Sandberg, B.Wiley: Protecting free expression online with freenet. IEEE Internet Computing **6**(1), 40–49 (2002)
44. I.Stoica, R.Morris, D.Karger, M.F.Kaashoek, H.Balakrishnan: Chord: A scalabale peer-to-peer lookup service for internet applications. In: Proc ACM SIGCOMM (2001)
45. I.Tartinov, *et al*: The Piazza peer data management project. In: SIGMOD (2003)
46. Iyer, S., Rowstron, A., Druschel, P.: Squirrel: a decentralized peer-to-peer web cache. In: PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing, pp. 213–222 (2002)
47. J.Kubiatowicz, D.Bindel, Y.Chen, S.Czerwinski, P.Eaton, D.Geels, R.Gummadi, S.Rhea, H.Weatherspoon, C.Wells, B.Zhao: Oceanstore: an architecture for global-scale persistent storage. SIGOPS Oper. Syst. Rev. **34**(5), 190–201 (2000)
48. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: 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, pp. 654–663 (1997)
49. K.Hose, C.Lemke, K.Sattler: Processing relaxed skylines in pdms using distributed data summaries. In: CIKM, pp. 425–434 (2006)
50. L.Adamic, *et* al: Search in power law networks. Physical Review E **64**, 46,135–46,143 (2001)
51. L.A.Zadeh: Fuzzy sets. Information and Control **8**, 338–353 (1965)
52. L.A.Zadeh: Concept of a linguistic variable and its application to approximate reasoning-I. Information Systems **8**, 199–249 (1975)
53. Lee, D.H., Kim, M.H.: Database summarization using fuzzy ISA hierarchies. IEEE Trans. on Systems, Man and Cybernetics-Part B: Cybernetics **27**, 68–78 (1997)

54. Lenzerini, M.: Data integration: a theoretical perspective. In: PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 233–246 (2002)
55. Li, Y., Lao, L., Cui, J.H.: Sdc: A distributed clustering protocol for peer-to-peer networks. In: Networking, pp. 1234–1239 (2006)
56. L.Ramaswamy, B.Gedik, 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)
57. Madhavan, J., Bernstein, P.A., Doan, A., Halevy, A.: Corpus-based schema matching. In: ICDE '05: Proceedings of the 21st International Conference on Data Engineering, pp. 57–68 (2005)
58. M.Cai, M.Frank: Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network. In: WWW, pp. 650–657 (2004)
59. M.Cai, M.Frank, J.Chen, P.Szekely: Maan: A multi-attribute addressable network for grid information services. In: GRID (2003)
60. M.Castro, M.Costa, A.Rowstron: Should we build gnutella on a structured overlay? SIGCOMM Comput. Commun. Rev. **34**(1), 131–136 (2004)
61. M.Espil, A.Vaisman: Aggregate queries in peer-to-peer olap. In: DOLAP (2004)
62. Ng, W., B.Ooi, K-L.Tan, A.Zhou: Peerdb: A p2p-based system for distributed data sharing. In: ICDE, pp. 633–644 (2003)
63. N.Harvey, M.Jones, S.Saroiu, M.Theimer, A.Wolman: Skipnet: A scalable overlay network with practical locality properties. In: USENIX Symposium on Internet Technologies and Systems (2003)
64. O.Sahin A.Gupta, D., A.El-Abbadi.: Query processing over peer-to-peer data sharing systems. Tech. rep., University of California, Santa Barbara (2002)
65. Paik, H.Y., Mouaddib, N., Benatallah, B., Toumani, F., Hassan, M.: Building and querying e-catalog networks using p2p and data summarisation techniques. J. Intell. Inf. Syst. **26**(1), 7–24 (2006)
66. P.Bernstein, F.Giunchiglia, A.Kementsietsidis, J.Mylopoulos, L.Serafini, 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)
67. P.Kalnis, W.Ng, B.Ooi, D.Papadias, K.Tan: An adaptive peer-to-peer network for distributed caching of olap results. In: SIGMOD (2002)
68. P.Kalnis, W.Ng, B.Ooi, K.Tan: Answering similarity queries in peer-to-peer networks. Inf. Syst. **31**(1), 57–72 (2006)
69. P.Maymounkov, D.Mazieres: Kademlia: A peer-to-peer information system based on the xor metric. In: Int. Workshop on Peer-to-Peer Systems (IPTPS), pp. 53–65 (2002)
70. P.McBrien, A.Poulovassilis: Defining peer-to-peer data integration using both as view rules. In: DBISP2P, pp. 91–107 (2003)
71. Prade, H., Testemale, C.: Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries. Inf. Sci. **34**(2), 115–143 (1984)
72. P.Triantafillou, T.Pitoura: Towards a unifying framework for complex query processing over structured peer-to-peer data networks. In: DBISP2P, pp. 169–183 (2003)
73. P.Wu, C.Zhang, Y.Feng, B.Zhao, D.Agrawal, A.El-Abbadi: Parallelizing skyline queries for scalable distribution. In: EDBT, pp. 112–130 (2006)
74. R.Akbarinia, E.Pacitti, P.Valduriez: Reducing network traffic in unstructured p2p systems using top-k queries. Distrib. Parallel Databases **19**(2-3), 67–86 (2006)
75. R.Akbarinia, E.Pacitti, P.Valduriez: Processing top-k queries in distributed hash tables. In: Euro-Par, pp. 489–502 (2007)
76. R.Akbarinia, V.Martins, E.Pacitti, P.Valduriez: Design and implementation of appa. In: Global Data Management (Eds. R. Baldoni, G. Cortese and F. Davide). IOS press (2006)
77. R.Aringhieri, E.Damiani, S.Vimercati, S.Paraboschi, 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)

78. Rasmussen, D., Yager, R.R.: SummarySQL - a fuzzy tool for data mining. Intelligent Data Analysis **1**, 49–58 (1997)
79. R.Dingledine, M.Freedman, D.Molnar: The free haven project: distributed anonymous storage service. In: International workshop on Designing privacy enhancing technologies, pp. 67–95. Springer-Verlag New York, Inc., New York, NY, USA (2001)
80. R.Huebsch, J.Hellerstein, N.Lanham, B.Thau, L.Shenker, I.Stoica: Querying the internet with pier. In: VLDB (2003)
81. S.Ratnasamy, M.Handley, R.Karp, S.Shenker: Topologically-aware overlay construction and server selection. In: Proceedings of IEEE INFOCOM'02 (2002)
82. S.Ratnasamy, P.Francis, M.Handley, R.M.Karp, S.Shenker: A scalable content–addressable network. In: SIGCOMM (2001)
83. Sripanidkulchai, K., Maggs, B.M., Zhang, H.: Efficient content location using interest-based locality in peer-to-peer systems. In: INFOCOM (2003)
84. S.Wang, B.Ooi, A.Tung, L.Xu: Efficient skyline query processing on peer-to-peer networks. In: ICDE (2007)
85. T.Luu, G.Skobeltsyn, F.Klemm, M.Puh, Žarko, I.P., M.Rajman, K.Aberer: Alvisp2p: Scalable peer-to-peer text retrieval in a structured p2p network. In: Proc VLDB (2008)
86. Tsoumakos, D., Roussopoulos, N.: A comparison of peer-to-peer search methods. In: Int.Workshop on the Web and Databases (WebDB), pp. 61–66 (2003)
87. U.Guntzer, W.Balke, W.Kieβling: Optimizing multi-feature queries for image databases. In: VLDB (2000)
88. Ullman, J.D.: Information integration using logical views. In: ICDT '97: Proceedings of the 6th International Conference on Database Theory, pp. 19–40 (1997)
89. V.Kalogeraki, D.Gunopulos, D.Yazti: A local search mechanism for peer-to-peer networks. In: Proc CIKM. USA (2002)
90. Voglozin, A., Raschia, G., Ughetto, L., Mouaddib, N.: Handbook of Research on Fuzzy Information Processing in Databases, vol. 1, chap. From User Requirements to Evaluation Strategies of Flexible Queries in Databases, pp. 115–142 (2008)
91. W.A.Voglozin, G.Raschia, L.Ughetto, N.Mouaddib: Querying the SAINTETIQ summaries–a first attempt. In: Int.Conf.On Flexible Query Answering Systems (FQAS) (2004)
92. W.Balke, W.Nejdl, W.Siberski, U.Thaden: Progressive distributed top-k retrieval in peer-to-peer networks. In: ICDE (2005)
93. W.Nejdl, W.Siberski: Design issues and challenges for rdf- and schema-based peer-to-peer systems. SIGMOD Record **32**, 2003 (2003)
94. W.Ng, B.Ooi, K.Tan: Bestpeer: A self-configurable peer-to-peer system. In: ICDE (2002)
95. X.Li, Y.J.Kim, R.Govindan, W.Hong: Multidimensional range queries in sensor networks. In: SENSYS (2003)
96. Yager, R.R.: On linguistic summaries of data. In: Knowledge Discovery in Databases, pp. 347–366. MIT Press (1991)
97. Yang, B., Vinograd, P., Garcia-Molina, H.: Evaluating guess and non-forwarding peer-to-peer search. In: ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), pp. 209–218 (2004)
98. Y.Chawathe, S.Ratnasamy, L.Breslau, N.Lanham, S.Shenker: Making gnutella-like p2p systems scalable. In: In Proc. ACM SIGCOMM (2003)
99. Y.Halevy, G.Ives, D.Suciu, I.Tatarinov: Schema mediation for large-scale semantic data sharing. The VLDB Journal **14**(1), 68–83 (2005)