Query processing in P2P systems

Reza Akbarinia, Esther Pacitti, Patrick Valduriez

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Query processing in P2P systems

Reza Akbarinia, Esther Pacitti, Patrick Valduriez

N° ????

January 2007

Thème XXX

ISSN 0249-6399

Query processing in P2P systems

Reza Akbarinia[1], Esther Pacitti[2], Patrick Valduriez[3]

Thème Sym C : Organisation des contenus et de la langue
Projet Atlas

Rapport de recherche n° ???? – Janvier 2007 - 38 pages

Abstract: Peer-to-peer (P2P) computing offers new opportunities for building highly distributed data systems. Unlike client-server computing, P2P is a very dynamic environment where peers can join and leave the network at any time. This yields important advantages such as operation without central coordination, peers autonomy, and scale up to large number of peers. However, providing high-level data management services is difficult. Most techniques designed in distributed database systems which statically exploit schema and network information no longer apply. New techniques are needed which should be decentralized, dynamic and self-adaptive. In this paper, we survey the techniques which have been developed for query processing in P2P systems. We first give an overview of the existing P2P networks, and compare their properties from the perspective of data management. Then, we discuss the approaches which are used for schema mapping. Then, we describe the algorithms which have been proposed for query routing. In particular, we focus on query routing in unstructured networks and DHTs. Finally, we present the techniques which have been proposed for processing complex queries, e.g. top-k queries, in P2P systems, in particular in DHTs.

Keywords: P2P systems, query processing, schema mapping, query routing, complex queries, top-k queries.

[1] INRIA and LINA, University of Nantes – Reza.Akbarinia@univ-nantes.fr
[2] INRIA and LINA, University of Nantes – Esther.Pacitti@univ-nantes.fr
[3] INRIA and LINA, University of Nantes – Patrick.Valduriez@inria.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : +33 2 99 84 71 00 — Télécopie +33 2 99 84 71 71

Traitement de Requêtes dans les Systèmes Pair-à-Pair

Résumé: Le pair-à-pair (P2P) fournit de nouvelles opportunités pour construire des systèmes de gestion de données distribués à grande échelle. Contrairement au client-serveur, le P2P est un environnement très dynamique, où les pairs peuvent rejoindre ou quitter le réseau à tout moment. Ceci offre des avantages importants comme la décentralisation du contrôle, l'autonomie des pairs et le passage à l'échelle en nombre de pairs. Mais le support de services de gestion de gestion de données de haut niveau est difficile car les techniques de gestion de bases de données distribuées qui exploitent statiquement un schéma ne sont pas applicables. Nous avons besoin de nouvelles techniques qui soient décentralisées, dynamiques et adaptatives. Dans cet article, nous étudions les techniques développées pour le traitement de requêtes dans les systèmes P2P. Nous commençons par une introduction aux réseaux P2P et une comparaison de leurs propriétés du point de vue de la gestion de données. Puis nous présentons les approches à la gestion de schémas et les algorithmes proposés pour le routage de requêtes. En particulier, nous nous intéressons aux réseaux non structurés et aux DHTs. Enfin, nous présentons les techniques proposées pour le traitement de requêtes complexes, par ex. top-k, dans les systèmes P2P, en particulier, dans les DHTs.

Mots clés: Systèmes P2P, traitement de requêtes, traduction de schémas, routage de requêtes, requêtes complexes, requêtes top-k.

1 Introduction

Data management in distributed systems has been traditionally achieved by distributed database systems [OV99] which enable users to transparently access and update several databases in a network using a high-level query language (*e.g.* SQL). Transparency is achieved through a global schema which hides the local databases' heterogeneity. In its simplest form, a *distributed database system* is a centralized server that supports a global schema and implements distributed database techniques (query processing, transaction management, consistency management, etc.). This approach has proved effective for applications that can benefit from centralized control and full-fledge database capabilities, *e.g.* information systems. However, it cannot scale up to more than tens of databases. Data integration systems [TV00, TRV98] extend the distributed database approach to access data sources on the Internet with a simpler query language in read-only mode. Parallel database systems [Val93] also extend the distributed database approach to improve performance (transaction throughput or query response time) by exploiting database partitioning in a multiprocessor or cluster system. Although data integration systems and parallel database systems can scale up to hundreds of data sources or database partitions, they still rely on a centralized global schema and strong assumptions about the network.

In contrast, P2P systems adopt a completely decentralized approach to resource management. By distributing data storage, processing, and bandwidth across autonomous peers in the network, they can scale without the need for powerful servers. P2P systems have been successfully used for sharing computation, *e.g.* Seti@home [ACKL+02, Set06] and Genome@home [LSP03, Gen06], communication, *e.g.* ICQ [Icq06] and Jabber [Jab03], internet services, *e.g.* P2P multicast systems [CDKR02, LRSS02, CJKR+03, BKRS+04] and security applications [KR02, JWZ03, VAS04], or data, *e.g.* Gnutella [Gnu06, JAB01, Jov00], KaZaA [Kaz06] and PeerDB [OST03, SOTZ03].

There are several features that distinguish P2P systems from traditional distributed database systems (DDBS) and make it difficult to provide advanced data management services over P2P networks [SOTZ03]:

- Peers are very dynamic and can join and leave the system anytime. However, in DDBS, nodes are added to and removed from the system in a controlled manner.
- Usually there is no predefined global schema for describing the data which are shared by the peers.
- In P2P systems, the answers to queries are typically incomplete. The reason is that some peers may be absent at query execution time. In addition, due to the very large scale of the network, forwarding a query to all peers can be very inefficient.
- In P2P systems, there is no centralized catalog that can be used to determine the peers that hold relevant data to a query. However, such a catalog is an essential component of DDBS.

Initial research on P2P systems has focused on improving the performance of query routing in the unstructured systems which rely on flooding. This work led to structured solutions based on distributed hash tables (DHT), *e.g.* CAN [RFHK+01] and Chord [SMKK+01], or hybrid solutions with super-peers [NSS03].

Although very useful, most of the initial P2P systems are quite simple (*e.g.* file sharing), support limited functions (*e.g.* keyword search) and use simple techniques (*e.g.* resource location by flooding) which have performance problems. In order to overcome these limitations, recent works have concentrated on supporting advanced applications which must deal with semantically rich data (*e.g.* XML documents, relational tables, etc.) using a high-level SQL-like query language, *e.g.* ActiveXML [ABCM+03], Edutella [NWQD+02, NSS03], Piazza [HIMT03, TIMH+03], PIER [HHLT+03].

One of the main services which are needed for supporting advanced P2P applications is a query processing service which deals with schema-based queries. However, providing such a service in P2P systems is quite challenging because of the specific features of these systems, *e.g.* lack of a global schema. Most techniques designed for distributed database systems which statically exploit schema and network information no longer apply. New techniques are needed which should be decentralized, dynamic and self-adaptive. Therefore, novel techniques have been proposed to perform decentralized schema mapping, to route queries to relevant peers without relying on a centralized catalog, and to execute queries, especially complex queries such as top-k queries, in a fully distributed fashion while taking into account the dynamic behavior of peers.

In this paper, we survey the techniques which have been proposed for query processing in P2P systems. In particular, we focus on schema-based queries which essential for advanced data management applications. We first give an overview of the existing P2P networks, and compare their properties from the perspective of data management. Then, we discuss the approaches which are used in P2P systems for schema mapping. Then, we present the algorithms which have been proposed for routing queries to relevant peers. In particular, we focus on query routing in unstructured networks and DHTs. Then, we present the techniques which have been proposed for processing complex queries in P2P systems, in particular in DHTs. Examples of complex queries are top-k queries, join queries, range queries and multi-attribute queries.

The rest of this paper is organized as follows. In Section 2, we present and discuss P2P networks. In Section 3, we present schema mapping in P2P systems. In Section 4, we focus on query routing in P2P systems. Section 5 deals with complex queries. Section 6 concludes.

2 P2P Networks

All P2P systems rely on a P2P network to operate. This network is built on top of the physical network (typically the Internet), and thus referred to as *overlay network*. The degree of centralization and the topology of the overlay network strongly impact the nonfunctional properties of the P2P system, such as fault-tolerance, self-maintainability, performance, scalability, and security. For simplicity, we consider three main classes of P2P networks: unstructured, structured, and super-peer.

2.1 Unstructured

In unstructured P2P networks, the overlay network is created in a nondeterministic (ad hoc) manner and data placement is completely unrelated to the overlay topology. Each peer knows its neighbors, but does not know the resources they have. Query routing is typically done by flood-

ing the query to the peers that are in limited hop distance from the query originator. There are also more sophisticated and efficient query routing techniques in unstructured systems (see Section 4 for more details).

There is no restriction on the manner to describe the desired data (query expressiveness), *i.e.* key look-up, SQL-like query, and other approaches can be used. Fault-tolerance is very high since all peers provide equal functionality and are able to replicate data. In addition, each peer is autonomous to decide which data to store. However, the main problems of unstructured networks are scalability and incompleteness of query results. Query routing mechanisms based on flooding usually do not scale up to a large number of peers because of the huge amount of load which they incur on the network. Also, the incompleteness of the results can be high since some peers containing relevant data may not be reached because they are too far away from the query originator.

Examples of P2P systems supported by unstructured networks include Gnutella [Jov00, JAB01, Gnu06], KaZaA [Kaz06] and FreeHaven [DFM00].

2.2 Structured

Structured networks have emerged to solve the scalability problem of unstructured networks. They achieve this goal by tightly controlling the overlay topology and data placement. Data (or pointers to them) are placed at precisely specified locations and mappings between data and their locations (*e.g.* a file identifier is mapped to a peer address) are provided in the form of a distributed routing table.

Distributed hash table (DHT) is the main representative of structured P2P networks. A DHT provides a hash table interface with primitives put(key,value) and get(key), where key is an object identifier, and each peer is responsible for storing the values (object contents) corresponding to a certain range of keys. Each peer also knows a certain number of other peers, called neighbors, and holds a routing table that associates its neighbors' identifiers to the corresponding addresses. Most DHT data access operations consist of a lookup, for finding the address of the peer p that holds the requested data, followed by direct communication with p. In the lookup step, several hops may be performed according to nodes' neighborhoods.

Queries can be efficiently routed since the routing scheme allows one to find a peer responsible for a key in $O(\log n)$ routing hops, where n is the number of peers in the network. Because a peer is responsible for storing the values corresponding to its range of keys, autonomy is limited. Furthermore, DHT queries are typically limited to exact match keyword search. Active research is on-going to extend the DHT capabilities to deal with more complex queries such as range queries [GS04], join queries [HHLT+03], and top-k queries [APV06b].

Examples of P2P systems supported by structured networks include Chord [SMKK+01], CAN [RFHK+01], Tapestry [ZHSR+04], Pastry [RD01a], Freenet [CMHS+02], PIER [HHLT+03], OceanStore [KBCC+00], Past [RD01b], and P-Grid [ACDD+03, AHA03]. Freenet 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 [AS04]. They use a chain mode propagation approach, where each node makes a local decision about to which node to send the request message next. P-Grid is not supported by a DHT either. It is based on a virtual distributed search tree.

2.3 Super-peer

Unstructured and structured P2P networks are considered "pure" because all their peers provide the same functionality. In contrast, super-peer networks are hybrid between client-server systems and pure P2P networks. Like client-server systems, some peers, the *super-peers*, act as dedicated servers for some other peers and can perform complex functions such as indexing, query processing, access control, and meta-data management. Using only one super-peer reduces to client-server with all the problems associated with a single server. Like pure P2P networks, super-peers can be organized in a P2P fashion and communicate with one another in sophisticated ways, thereby allowing the partitioning or replication of global information across all super-peers. Super-peers can be dynamically elected (*e.g.* based on bandwidth and processing power) and replaced in the presence of failures.

In a super-peer network, a requesting peer simply sends the request, which can be expressed in a high-level language, to its responsible super-peer. The super-peer can then find the relevant peers either directly through its index or indirectly using its neighbor super-peers.

The main advantages of super-peer networks are efficiency and quality of service (*i.e.* the user-perceived efficiency, *e.g.* completeness of query results, query response time, etc.). The time needed to find data by directly accessing indices in a super-peer is very small compared with flooding. In addition, super-peer networks exploit and take advantage of peers' different capabilities in terms of CPU power, bandwidth, or storage capacity as super-peers take on a large portion of the entire network load. In contrast, in pure P2P networks, all nodes are equally loaded regardless of their capabilities. Access control can also be better enforced since directory and security information can be maintained at the super-peers. However, autonomy is restricted since peers cannot log in freely to any super-peer. Fault-tolerance is typically low since super-peers are single points of failure for their sub-peers (dynamic replacement of super-peers can alleviate this problem).

Examples of super-peer networks include Napster [Nap06], Publius [WAL00], Edutella [NSS03, NWQD+02], and JXTA [Jxt06]. A more recent version of Gnutella also relies on super-peers [AS04].

2.4 Comparing P2P Networks

From the perspective of data management, the main requirements of a P2P network are [DGY03]: autonomy, query expressiveness, efficiency, quality of service, fault-tolerance, and security. We describe these requirements in the following. Then, we compare P2P networks based on these requirements.

- **Autonomy**: an autonomous peer should be able to join or leave the system at any time without restriction. It should also be able to control the data it stores and which other peers can store its data, *e.g.* some other trusted peers.
- **Query expressiveness**: the query language should allow the user to describe the desired data at the appropriate level of detail. The simplest form of query is key look-up which is only appropriate for finding files. Keyword search with ranking of results is appropriate for searching documents. But for more structured data, an SQL-like query language is necessary.

- **Efficiency**: the efficient use of the P2P network resources (bandwidth, computing power, storage) should result in lower cost and thus higher throughput of queries, *i.e.* a higher number of queries can be processed by the P2P system in a given time.
- **Quality of service**: refers to the user-perceived efficiency of the P2P network, *e.g.* completeness of query results, data consistency, data availability, query response time, etc.
- **Fault-tolerance**: efficiency and quality of services should be provided despite the occurrence of peers failures.
- **Security**: the open nature of a P2P network makes security a major challenge since one cannot rely on trusted servers. Wrt. data management, the main security issue is access control which includes enforcing intellectual property rights on data contents.

Table 1 summarizes how the requirements for data management are possibly attained by the three main classes of P2P networks. This is a rough comparison to understand the respective merits of each class. For instance, "high" means it can be high. Obviously, there is room for improvement in each class of P2P networks. For instance, fault-tolerance can be made higher in super-peer by relying on replication and fail-over techniques.

Table 1. Comparison of P2P networks

Requirements	**Unstructured**	**Structured**	**Super-peer**
Autonomy	high	low	moderate
Query expressiveness	"high"	low	"high"
Efficiency	low	high	high
QoS	low	high	high
Fault-tolerance	high	high	how
Security	low	low	high

3 Schema Mapping

Semantic heterogeneity is a key problem in any large scale data sharing system, be it a data integration system or a P2P management system [MBDH05]. The data sources involved are typically designed independently, and hence use different schemas. To be able to do meaningful inter-operation between the data of two sources, the system needs a schema mapping, *i.e.* a set of expressions that specify how the data in one source corresponds to the data in the other. In relational systems, schema mapping signifies the process of defining the semantic equivalence between relations and attributes of the two schemas [BAHS+06].

In this section, we first give a brief description of schema mapping in data integration systems. Then, we discuss the approaches which are used in P2P systems for schema mapping.

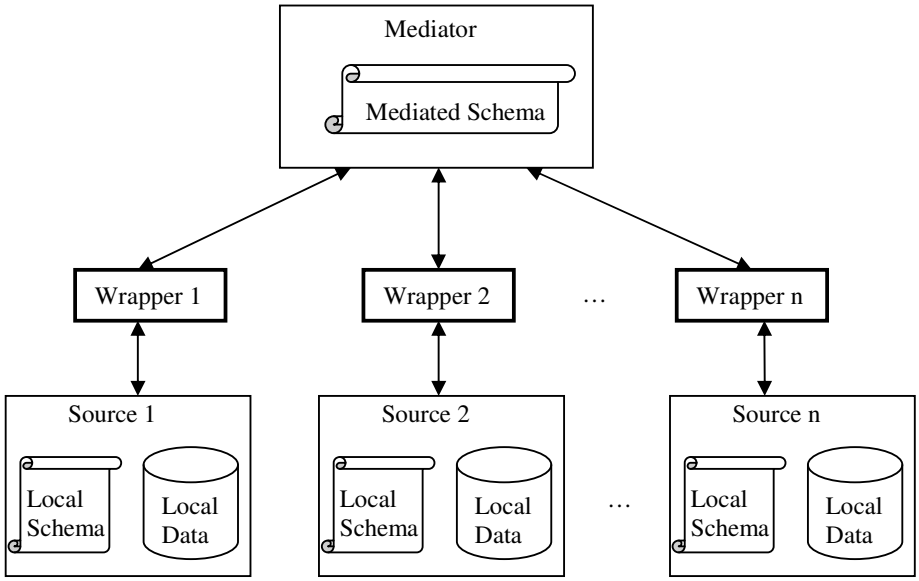

Figure 1. Schema Mapping using a Global Mediated Schema

3.1 Schema Mapping in Data Integration Systems

To be able to query heterogeneous data sources, data integration systems [TV00, TRV98] typically rely on the definition of a global *mediated schema* (see **Figure 1**). The local schemas of the data sources are mapped over the mediated schema, and the mappings are maintained at a mediator. Users issue their queries in terms of the mediated schema, and the mediator reformulates the query into sub-queries which are expressed on local schemas and can be executed at data sources. There is a wrapper close to each data source that provides translation services between the mediated schema and the local query language [Ull97].

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 [Len02]. In GAV, the autonomy of data sources is higher than LAV because they can define their local schemas as they want. However, if any new source is added to a system that uses the GAV approach, considerable effort may be necessary to update the mediator code. Thus, GAV should be favored when the sources are not likely to change. The advantage of a LAV modeling is that new sources can be added with far less work than in GAV. LAV should be favored when the mediated schema is not likely to change, *i.e.* the mediated schema is complete enough that all the local schemas can be described as a view of it.

3.2 Schema Mapping in P2P Systems

Due to specific characteristics of P2P systems, *e.g.* the dynamic and autonomous nature of peers, the approaches that rely on centralized global schemas no longer apply in P2P systems. 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.

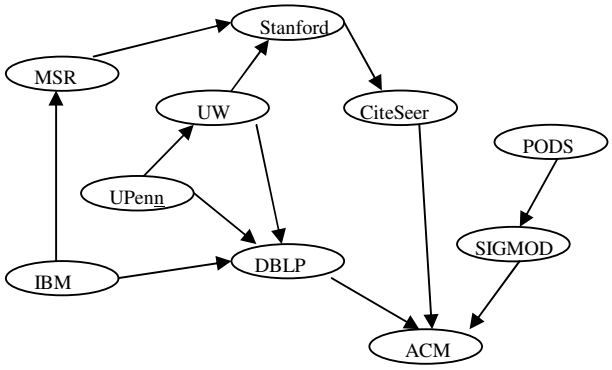

Figure 2. An Example of Pairwise Schema Mapping in Piazza

3.2.1 Pairwise Schema Mapping

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 [TIMH+03] follows this approach (see Figure 2). 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 subtree 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) [BGKM+02] is another example that follows this approach. LRM assumes that the peers hold relational databases, and each peer knows a set of peers with which it can exchange data and services. This set of peers is called *p*'s acquaintances. Each peer must define semantic dependencies and translation rules between its data and the data shared by each of its acquaintances. The defined mappings form a semantic network, which is used for query reformulation in the P2P system.

PGrid also assumes the existence of pairwise mappings between peers, initially constructed by skilled experts [ACH03]. 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.

3.2.2 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 [W3C01]. 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 [DMDD+03] 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.

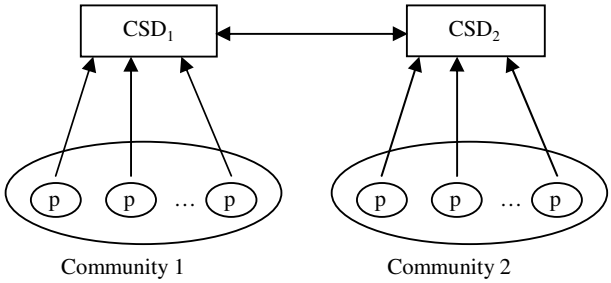

Figure 3. Common Agreement Schema Mapping in APPA

3.2.3 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 [AMPV06a, AM06, AMPV04] 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 3). 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. Given two CSD relation definitions r_1 and r_2, an example of peer mapping at peer p is: $p{:}r(A,B,D) \subseteq csd{:}r_1(A,B,C),\ csd{:}r_2(C,D,E)$. In this example, the relation $r(A,B,D)$ which is shared by peer p is mapped to relations $r_1(A,B,C)$, $csd{:}r_2(C,D,E)$ which are involved in 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 [MP04] 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.

3.2.4 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 [OST03] 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 [NSS03] 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 super-peers. 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.

4 Query Routing

The main problem for query processing in P2P systems is how to route the query to relevant peers, *i.e.* those that hold some data related to the query, [LW06]. Once the query is routed to relevant peers, it is executed at those peers and the answers are returned to the query originator.

In this section, we describe the approaches for query routing in unstructured systems, DHTs, and super-peer systems.

4.1 Query Routing in Unstructured Systems

The approaches used in unstructured systems for query routing can be classified in the following groups [TR03b]: Breath-First Search (BFS), Iterative deepening, random walks, adaptive probabilistic search, local indices, bloom filter based indices, and distributed resource location protocol.

4.1.1 BFS

This approach, which is used originally by Gnutella for data discovery, floods the query to all accessible peers within a TTL (Time To Live) hop distance as follows. Whenever a query with a TTL is issued at a peer, called query originator, it is forwarded to all its neighbors. Each peer, which receives the query, decreases the TTL by one and if it is greater than one sends the query and TTL to its neighbors. By continuing this procedure, all accessible peers whose hop distance from the query originator is less than or equal to TTL receive the query. Each peer that receives the query executes it locally and returns the answers directly to the query originator (see Figure 4).

Modified BFS [KGZ02] is a variation of the BFS approach in which the peers randomly choose only a ratio of their neighbors and forward the query only to these neighbors (see Figure 5). Although this approach reduces the number of messages needed for query routing, it may lose many of the good answers which could be found by BFS.

Intelligent BFS [KGZ02] is another variation. For each recently answered query, peers maintain the query and the number of answers which are found via each of their neighbors. When a peer receives a query, it identifies all queries similar to the received query, *e.g.* using a query similarity metric, and sends the query to a set of its neighbors that have returned most answers for the similar queries. If an answer is found for the query at a peer, a message is sent to the peers over the reverse path in order to update their statistics. Like standard BFS, each peer that receives the query decreases the TTL by one, and if it is equal to zero, the query is discarded. Compared to modified BFS, intelligent BFS can find many more answers. However, it produces more routing messages, *i.e.* because of update messages. In addition, it can not be easily adapted to the peer departures and data deletions.

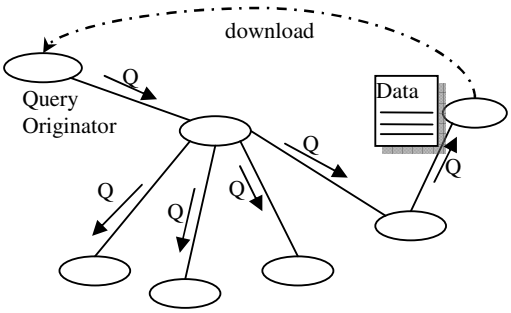

Figure 4. Example of BFS: the received query is forwarded to all the neighbors

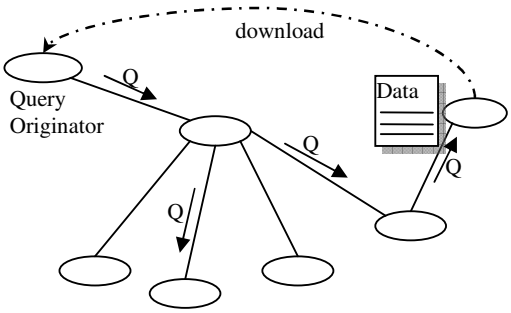

Figure 5. Example of Modified BFS: the received query is forwarded to a randomly selected set of neighbors

4.1.2 Iterative Deepening

Iterative deepening [YG02, LCCL+02] is used when the user is satisfied by only one (or a small number) of the closest answers. In this algorithm, the query originator performs consecutive BFS searches such that the first BFS has a low TTL, *e.g.* 1, and each new BFS uses a TTL greater than the previous one. The algorithm ends when the required number of answers is found or a BFS with the predefined maximum TTL is done. For the cases where a sufficient number of answers are available at the peers that are close to the query originator, this algorithm achieves

good performance gains compared to the standard BFS. In other cases, its overhead and response time may be much higher than the standard BFS.

4.1.3 Random Walks

In Random Walks [LCCL+02], for each query, the query originator forwards *k* query messages to *k* of its randomly chosen neighbors. Each of these messages follows its own path, having intermediate peers forward it to a randomly chosen neighbor at each step (see Figure 6). These messages are known as *walkers*. When the TTL of a walker reaches zero, it is discarded. Each walker periodically contacts the query originator, asking whether the termination condition is held or not. If the response is positive, the walker terminates.

Let *k* be the number of walkers. The main advantage of the Random Walks algorithm is that it produces $k \times TTL$ routing messages in the worst case, a number which does not depend on the underlying network. Simulation results in [LCCL+02] show that routing messages can be reduced significantly compared to the standard BFS. The main disadvantage of this algorithm is its highly variable performance, because success rates and the number of found answers vary greatly depending on network topology and the random choices. Another drawback of this method is that it cannot learn anything from its previous successes or failures.

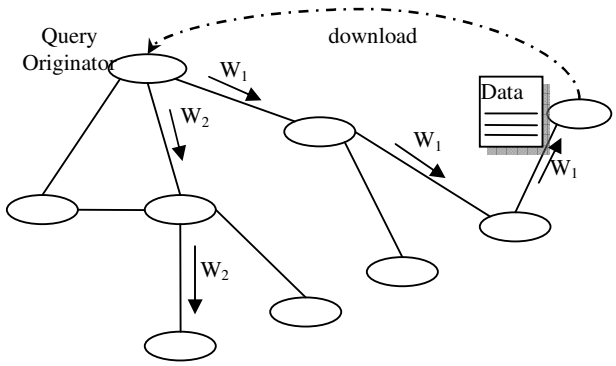

Figure 6. Example of Random Walks: each received walk is forwarded to only one neighbor

4.1.4 Adaptive Probabilistic Search

In Adaptive Probabilistic Search (APS) [TR03a], for each recently requested data, the peers maintain the data identifier and probability of returning the data by each of their neighbors. Given a query asking for a data, the query originator establishes *k* independent walkers and sends them to its neighbors. Each intermediate peer, which receives a walker, sends it to the neighbor that has the highest probability to return the requested data. Initially equal for all neighbors, the probability values are updated using either an optimistic approach or a pessimistic approach. In the optimistic approach, when a peer sends a walker to a neighbor, it increases in advance the corresponding probability value. However, if the walker terminates without the requested data, a message is sent over the walker path to decrease the corresponding probability values. The pessimistic approach makes the assumption that the data cannot be found, so it decreases the corresponding probability value after sending the walker to a neighbor. If the walker finds the data, all peers over the walker path update their probability values by increasing them.

To remember a walker's path, each peer appends its ID in the query message during query forwarding. If a walker w_2 passes by a peer where another walker w_1 stopped before, the walker w_2 terminates unsuccessfully. APS has very good performance as it is bandwidth-efficient: the number of routing messages produced by it is very close to that of Random Walks. In spite of this, the probability of finding the requested data by APS is much higher than that of Random Walks. However, if the topology of the P2P system changes quickly, the ability of APS to answer queries reduces significantly.

4.1.5 Local Indices

In this approach [YG02, CG02], each peer *p* indexes the data shared by all peers which are within a certain radius *r*, i.e. the peers whose hop-distance from *p* is less than or equal to *r*. 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, …. 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.

4.1.6 Bloom Filter based Indices

In [RK02], the indexing of data is done using Bloom filters. Each peer holds *d* bloom filters for each neighbor, such that the *i*th filter summarizes the data that can be found *i* hops away through that specific neighbor. When a peer receives a query requesting a data, it checks its local data, if the data is found it is returned to the query originator. Otherwise, the peer forwards the query to the neighbor who has the minimum numbered filter that represents the data among its members.

The advantage of representing the indexed data by Bloom filters [Blo70] is that they are space efficient, *i.e.* with a small space, one can index a large number of data. However, it is not possible to remove a data from a Bloom filter, so they are not easily adaptable to highly dynamic environments. In addition, it is possible that a Bloom filter gives a false positive answer, *i.e.* the Bloom filter wrongly returns a positive answer in response to a question asking the membership of a data item.

4.1.7 Distributed Resource Location Protocol

In Distributed Resource Location Protocol (DRLP) [MK02], the peers index the location of all data which are the answer for recently issued queries. The indexing is done gradually as follows. Peers with no information about the location of a requested data, forward the query to a set of randomly chosen neighbors. If the data is found at some peer, a message is sent over the reverse path to the query originator, storing the data location at those peers. In subsequent requests, peers with indexed location information forward the query directly to the relevant peers. This algorithm initially spends many routing messages for query processing. In subsequent requests,

it might take only one message to discover it. Thus, if a query is issued frequently, this approach is very efficient.

4.2 Query routing in DHTs

Distribute Hash Tables (DHTs) provide an efficient and scalable solution for data location in P2P systems. While there are significant implementation differences between DHTs, they all map a given key onto a peer *p*, called responsible for the key, using a hash function and can lookup *p* efficiently, usually in O(*log n)* routing hops where *n* is the number of peers [HHHL+02]. DHTs typically provide an operation *put(key, data)* that stores the data at the peer that is responsible for key. For requesting a data, there is an operation *get(key)* that routes the key to the peer that is responsible for it, and retrieves the requested data.

The way by which a DHT routes the keys to their responsible depends on the DHT's *routing geometry* [GGGR+03], *i.e.* the topology which is used by the DHT for arranging peers and routing queries over them. The routing geometries in DHTs can be [GGGR+03]: tree, hypercube, butterfly, XOR and hybrid. In the following, we describe these geometries and discuss query routing.

4.2.1 Tree

Tree is the first geometry which is used for organizing the peers of a DHT and routing queries among them. In this approach, peer identifiers constitute the leaves of a binary tree of depth *log n* where *n* is the number of nodes of the tree. The responsible for a given key is the peer whose identifier has the highest number of prefix bits which are common with the key. Let *h(p, q)* be the height of the smallest common sub-tree between two peers *p* and *q*. For each $1 \leq i \leq log\ n$, each peer *p* knows the address of a peer *q* such that $h(p, q)=i$. This means that, for each $1 \leq i \leq log\ n$, the peer *p* knows a peer *q* such that the number of common prefix bits in the identifiers of *p* and *q* is *i*. The routing of a key proceeds by doing a longest prefix match at each intermediate peer until reaching to the peer which has the most common prefix bit with the key. The basic routing algorithms in Tapestry [ZHSR+04] is rather similar to this algorithm. In Tapestry, each identifier is associated with a node that is the root of a spanning tree used to route messages for the given identifier. The Tapestry routing geometry is very closely associated to a tree structure and we classify it as such.

The tree geometry gives a great deal of freedom to peers in choosing their neighbors: each peer has $2^i - 1$ options in choosing a neighbor in a sub-tree with height *i*. Thus, in total, each peer has about $2^{\log n * (\log n - 1) / 2}$ options to select all its neighbors. Therefore, the tree geometry has good neighbor selection flexibility. 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.

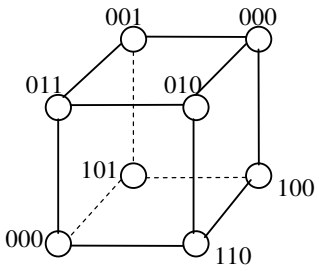

Figure 7. Example of Hypercube Routing Geometry

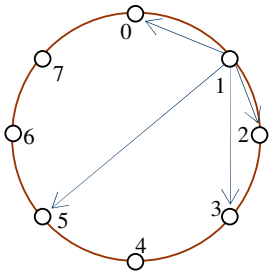

Figure 8. Example of Ring Routing Geometry: the fleshes point out to the neighbors of the peer #1.

4.2.2 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 of the hypercube. Each peer *p* has *log n* neighbors such that the identifier of the *i*th neighbor and *p* differ only in the *i*th bit. Thus, there is only one different bit between the identifier of *p* and each of its neighbors (see Figure 7). 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) * … * (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, for selecting its neighbors, a peer has only one choice. Thus, the hypercube geometry has no flexibility in the neighbor selection. This is the opposite of what occurs with the tree, which has much neighbor selection flexibility but no route selection flexibility.

The routing geometry used in CAN [RFHK+01] 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. When *d* = *log n*, the neighbor sets in CAN are similar to those of a *log n* dimensional hypercube.

4.2.3 Ring

The Ring geometry is based on a one dimensional cyclic space such that the peers are ordered on the circle clockwise with respect to their identifiers (see Figure 8). Chord [SMKK+01] is a DHT protocol that relies on this geometry for query routing. In Chord, each peer has an m-bit identifier, and the responsible for a key is the first peer whose identifier is equal or follows *k*.

Each peer *p* maintains the address of *log n* other peers on the ring such that the *i*th neighbor is the peer whose distance from *p* clockwise in the circle is $2^{i-1} \bmod n$. Hence, any peer can route a given key to its responsible in *log n* hops because each hop cuts the distance to the destination by half.

Although the original Chord protocol defines a specific set of neighbors for each peer, the ring geometry does not necessarily needs such rigidity in neighbor selection. In the ring geometry, *p* can select its *i*th neighbor from the peers whose distance from *p* clockwise in the circle is in the range $[2^{i-1} \bmod n \ , \ 2^{i} \bmod n)$. This implies that in the ring geometry, each peer has 2^{i-1} options in selecting its *i*th neighbor. Thus in terms of flexibility of neighbor selection, there are a total of approximately $2^{(\log n - 1)*(\log n - 1)/2}$ options for each peer.

The flexibility in route selection can be computed as follows. For communicating a message between two peers with a *log n* distance, the first peer has approximately *log n* of its neighbors that make progress towards the destination. The next peer has approximately *(log n) - 1* options for selecting the next hop, and so on. This yields a total of approximately *(log n)!* possible routes for a typical path.

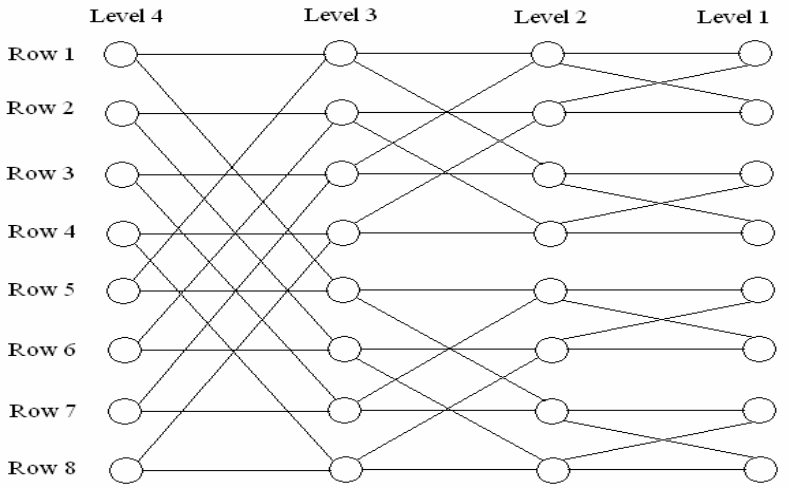

Figure 9. Example of the Butterfly Routing Geometry

4.2.4 Butterfly

The Butterfly geometry is an extension of the traditional butterfly network that supports the scalability requirements of DHTs. Viceroy [MNR02] is a DHT that uses this geometry for efficient data location. The peers of a butterfly with size *n* are portioned into log *n* levels and *n / log n* rows (see Figure 9). The peers of each row are subsequently connected to each other using successor/predecessor links. The number of peers in each row is *log n*, thus a sequential lookup in each row is done in *O(log n)*. In addition to successor/predecessor links, each peer has some links to the peers of other rows. The inter-row links are arranged in such a way that the distance between a peer in Level 1 of any row to any other row is *log n*. Routing a query in the Butterfly is done in three steps as follows. First, the query is sequentially forwarded to the peer that is at Level l of the same row as the query originator. This is done in *O(log n)* routing hops. 2)

Second, from Level 1, the query is routed in *O(log n)* routing hops to the row to which the destination peer belongs. Third, at the destination row, the query is traversed sequentially to the destination peers. Each of these steps is done in *O(log n)* routing hops, thus the total time of query routing is *O(log n)*. The advantages of the Butterfly geometry is that the size of the routing table per peer, *i.e.* the number of neighbors of each peer, is a small constant number whereas, in most of other geometries, this size is *O(log n)*. However, the Butterfly geometry has poor neighbor and route selection flexibility, *i.e.* there is only one choice for selecting the neighbors or the route.

4.2.5 XOR

The XOR approach uses a symmetric unidirectional tree topology for structuring the peers of the P2P network. Kademlia [MM02] is a DHT that uses the XOR geometry for query routing. In Kademlia, the distance between two peers is computed as the numeric value of the exclusive OR (XOR) of their identifiers. Each peer *p* has *log n* neighbors, where the *i*th neighbor is any peer whose XOR distance from *p* is a value in $[2^i , 2^{i+1})$. Query routing proceeds by greedily reducing via intermediate peers the XOR distance from the destination peer. Kademlia provides the same neighbor selection flexibility as the tree geometry. In addition, in terms of route selection, the XOR geometry can reduce the bit differences in any order, and does not require strict left-to-right bit fixing as the tree geometry. Thus, it has a great route selection flexibility which is comparable to that of the ring geometry.

4.2.6 Hybrid

Hybrid geometries use a combination of geometries. Pastry [RD01a] combines the tree and ring geometries in order to achieve more efficiency and flexibility. Peer identifiers are maintained as both the leaves of a binary tree and as points on a one-dimensional circle. In Pastry, the distance between a given pair of nodes is computed in two different ways: the tree distance between them and the ring distance between them. Peers have great flexibility of neighbor selection. For selecting their neighbors, peers take into account the proximity properties, *i.e.* they select the neighbors that are close to them in the underlying physical network. The route selection is also very flexible because, to route a message, peers have the possibility to choose one of the hops that do make progress on the tree or on the ring.

4.2.7 Comparing DHT Routing Geometries

Depending on their routing geometries, DHTs have different routing properties. Table 2 compares the DHT routing geometries from the point of view of the following routing properties:

- **Route selection flexibility.** This is defined as the number of options which a peers has for routing the query to the next hop. The higher this flexibility is, the more resilient the routing procedure is to dynamic behavior of peers.
- **Neighbor selection flexibility.** This is defined as the number of options which a peer has for selecting its neighbors. The higher this flexibility is, the more possibilities peers have for choosing neighbors which are close to them in the underlying network.

- **Routing table size.** This is the number of peers about which a peer needs to maintain routing information, *e.g.* their address. The larger the routing table size is, the higher the overhead of maintaining the overlay network is.

Table 2. Comparison of DHT Routing Geometries

DHT routing Geometry	Neighbor selection flexibility	Route selection flexibility	Routing table size
Tree	$O(2^{\log n * \log n})$	O(1)	O(log n)
Hypercube	O(1)	O((log n) !)	O(log n)
Ring	$O(2^{\log n * \log n})$	O((lon n) !)	O(log n)
Butterfly	O(1)	O(1)	O(1)
XOR	$O(2^{\log n * \log n})$	O((log n) !)	O(log n)

4.3 Super-peer

Super-peer networks typically rely on some powerful and highly available peers, called super-peers, to index the data shared by peers which are connected to the system. Edutella is one of the most known super-peer networks. In Edutella, super-peers are arranged in the HyperCup topology [SSDN02] (see Figure 10), so messages can be communicated between any two super-peers in *O(log m)* routing hops, where *m* is the number of super-peers. The process of joining a super-peer to the network consists of two parts: taking the appropriate position in the HyperCuP topology and announcing itself to its neighbors. Each ordinary peer joins the system by connecting to a super-peer.

To support efficient query routing, at each super-peer, two kinds of routing indices are maintained: *super-peer / peer* (SP/P) *indices* and *super-peer / super-peer (SP/SP) indices*. Queries are routed over super-peers by using the SP/SP indices, and to ordinary peers based on the SP/P indices.

In the SP/P indices, each super-peer stores information about the characteristics of the data which are shared by the peers that are connected to it. These indices are used to route a query from the super-peer to its connected peers. At join time, peers provide their metadata information to their super-peer by publishing an advertisement. To index the provided metadata, Edutella uses the schema-based approaches which have successfully been used in the context of mediator-based information systems (*e.g.* [Wie92]). To ensure that the indices are always up-to-date, peers notify super-peers when their data change. When a peer leaves the network, all references to this peer are removed from the indices. If a super-peer fails, its formerly connected peers must connect to another super-peer chosen at random, and provide their metadata to it.

The second type of indices is SP/SP indices which are essentially summaries (possibly also approximations) of SP/P indices. Update of SP/SP indices is triggered after any modification to SP/P indices as follows. When a super-peer changes its SP/P index, *e.g.* due to a peer's join/leave, it broadcasts an announcement of update to the super-peer network by using the HyperCuP protocol. The other super-peers update their SP/SP indices accordingly. Although such a broadcast is not optimal, it is not too costly either because the number of super-peers is much less than the number of all peers. Furthermore, if peers join/leave frequently, the super-peer can

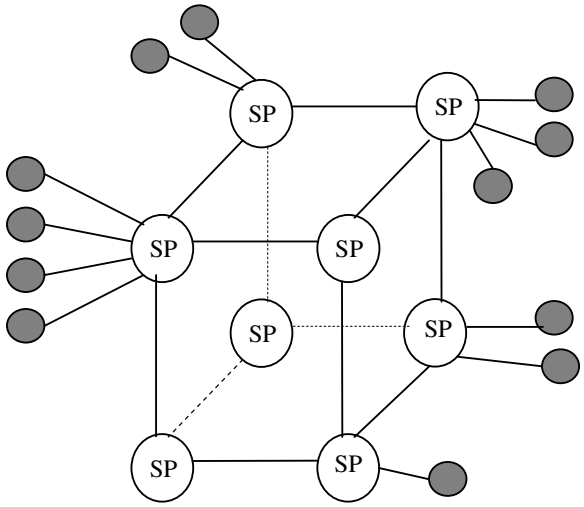

Figure 10. Edutella architecture

send a summary announcement periodically instead of sending a separate announcement for each join/leave.

The query routing in Edutella is done as follows. When a peer receives a query issued by the user, it sends the query to its super-peer. At the super-peer, the metadata used in the query are matched against the SP/P indices in order to determine local peers which are able to answer the query. If the query cannot be satisfied by local peers, it is forwarded to other super-peers using SP/SP indices.

5 Complex Query Processing

In the previous section, we discussed the techniques for routing queries to relevant peers. These techniques allow P2P systems to provide good support for exact-match queries. However, supporting more complex queries cannot be done easily in P2P systems, particularly in DHTs. Examples of complex queries are: top-k queries, join queries, range queries and multi-attribute queries. In this section, we discuss the techniques which have been proposed to support complex queries in P2P systems.

5.1 Top-k Queries

Top-k queries have attracted much attention in many fields of information technology such as network and system monitoring [BO03, KOTZ04, CW04], information retrieval [MTW05, BNST05, PZS96], multimedia databases [CGM04, NR99, DMNK02], spatial data analysis [BBK01, CP02, HS03], etc. With a top-k query, the user can specify a number k of the most relevant answers to be returned by the system. The degree of relevance (*score*) of the answers to the query is determined by a scoring function. Top-k queries are very useful for data management in P2P systems [APV06a], in particular when the number of all the answers is very large. For example, consider a P2P system with medical doctors who want to share some (restricted) patient data for an epidemiological study. Assume that all doctors agreed on a common Patient

description in relational format. Then, one doctor may want to submit the following query to obtain the 10 top answers ranked by a scoring function over height and weight:

```
SELECT  *
FROM    Patient P
WHERE   (P.disease = "diabetes") AND
        (P.height < 170) AND (P.weight > 70)
ORDER BY scoring-function(height, weight)
STOP AFTER 10
```

The scoring function specifies how closely each data item matches the conditions. For instance, in the query above, the scoring function could be *(weight - (height - 100))* which computes the amount of overweight.

Efficient execution of Top-k queries in large-scale P2P systems is difficult. In this section, we first discuss the most efficient techniques proposed for top-k query processing in distributed systems. Then, we present the techniques proposed for P2P systems.

5.1.1 Top-k queries in Distributed Systems

The most efficient approaches for top-k query processing in distributed systems are based on the Threshold Algorithm (TA) [FLN03, GKB00, NR99]. TA is applicable for queries where the scoring function is monotone, *i.e.*, any increase in the value of the input does not decrease the value of the output. Many of the popular aggregation functions, *e.g.* Min, Max, Average, are monotone. Given m lists of n data items such that each data item has a local score in each list and the lists are sorted according to the local scores of their data items, assume the overall score of a data item is computed based on the local scores of the data item in all lists using the scoring function. TA finds k data items whose overall scores are the highest as follows. TA goes down the sorted lists in parallel, one position at a time, and for each data item, retrieves its local scores in all lists, and computes the overall score. This process continues until finding k data items whose overall scores are greater than a threshold which is the overall score of a virtual data item whose local scores are the local scores which are at current position in the lists.

The TA algorithm can be applied to relational top-k query processing by viewing each column of a table as a list, thus sorting the columns according to their values, and using the TA algorithm for finding the k tuples whose overall scores are the highest.

Several TA-style algorithms, *i.e.* extensions of TA, have been yet proposed for distributed top-k query processing. In the context of the KLEE framework [MTW05], the authors propose a TA-style algorithm which aims at minimizing the communication cost of top-k query processing in distributed systems. This algorithm yields much performance gains at the expense of a small reduction in answer completeness. In KLEE, the m sorted lists are distributed over m nodes, and each node divides its list into c cells and maintains statistical information describing the cells, *e.g.* lower, upper, average and frequency of local scores which fall in the cell. KLEE uses Bloom filters to compactly represent, for each cell, the set of data items whose local scores fall in the cell. This information on cells along with the Bloom filters contribute at reducing the number of local scores which should be communicated over network, so reducing the communication cost.

The Three Phase Uniform Threshold (TPUT) [CW04] is a TA-Style algorithm that executes top-k queries in three round trips. TPUT assumes that each list is held by one node, which we call the list holder. TPUT works in three phases as follows.

1. Each list holder sends to the query originator its k top data items, *i.e.* the k data items whose local scores in the list are the highest. Let f be the scoring function, d be a received data item, and $s_i(d)$ be the local score of d in list i, then the partial sum of d is defined as $p_{sum}(d) = s_1'(d) + s_2'(d) + \ldots + s_m'(d)$ where $s_i'(d) = s_i(d)$ if d has been sent to the query originator from the holder of list i, otherwise $s_i'(d) = 0$. The query originator calculates the partial sums for all received data items and identifies the items with the k highest partial sums. The partial sum of the kth data item is called phase-1 bottom and denoted by λ_1.
2. The query originator sends a threshold value $\tau = \lambda_1/m$ to every list holder. Then, each list holder sends to the query originator all its data items whose local scores are not less than τ. The intuition is that if a data item is not reported by any node in this phase, its score must be less than λ_1, so it cannot be one of the top-k data items. Let D be the set of data items received from list holders, the query originator calculates the new partial sums for the data items involved in D, and identifies the items with the k highest partial sums. The partial sum of the kth data item is called phase-2 bottom, and denoted by λ_2. Let the upper bound score of a data item d be defined as $u_{score}(d) = u_1(d) + u_2(d) + \ldots + u_m(d)$ where $u_i(d) = s_i(d)$ if d has been received from the holder of list i, otherwise $u_i(d) = \tau$. For each data item $d \in D$, if the upper bound score of d is less than λ_2, it is removed from D. The data items which remain in D are called the top-k candidate data items.
3. The query originator sends the set of top-k candidate data items to each list holder which returns back the scores of these items. Then, the query originator calculates the overall score, extracts the k data items with highest scores, and returns the answer to the user

As shown in [CW04], for the cases where the number of lists (*i.e.* m) is high, the response time of TPUT is much better than that of the basic TA algorithm.

5.1.2 Top-k Queries in P2P systems

We now discuss the techniques for top-k query processing in each of the three classes of P2P systems.

5.1.2.1 Top-k Queries in Unstructured Systems

One possible approach for processing top-k queries in unstructured systems 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, as shown in [AMPV06b], this approach is not efficient in terms of response time and communication cost.

PlanetP [CPMN03] is an unstructured P2P system that supports top-k queries. In PlanetP, a content-addressable publish/subscribe service replicates global documents across P2P communities up to ten thousand peers. 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 per-

formance 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, we proposed a fully distributed (FD) framework to execute top-k queries in unstructured P2P systems [APV06a, AMPV06b]. FD involves a family of algorithms that are simple but effective. FD's execution is completely distributed 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. Given a top-k query Q with a specified TTL, the basic algorithm of FD is performed in four phases as follows:

1. **Query forward**. In this phase, using one of the yet proposed algorithms, Q is forwarded to the accessible peers whose hop-distance from the query originator is less than TTL.
2. **Local query execution and wait**. Each peer p that receives Q, executes it locally, *i.e.* accesses the local data items that match the query predicate, scores them using a scoring function, selects the k top data items and saves them as well as their scores locally. Then p waits to receive its neighbors' results. However, since some of the neighbors may leave the P2P system and never send a score-list to p, for the wait time there is a limit which is computed for each peer based on the received TTL, network parameters and peer's local processing parameters.
3. **Merge-and-backward.** In this phase, the top scores are bubbled up to the query originator using a tree based algorithm as follows. After its wait time has expired, p merges its k local top scores with those received from its neighbors and sends the result to its *parent* (the peer from which it received Q) in the form of a *score-list*. In order to minimize network traffic, FD does not bubble up the top data items (which could be large), only their scores and addresses. A score-list is simply a list of k couples *(a, s)*, such that a is the address of the peer owning the data item and s its score.
4. **Data retrieval.** After receiving the score-lists sent from its neighbors, the query originator makes the *final score-list* which is gained by merging its k local top scores with the merged score-lists received from its neighbors. Then it directly retrieves the k top data items from the peers which hold them.

We proposed several techniques to reduce FD's communication cost. We also proposed solutions to the problems which may occur during the execution of the top-k queries due to the dynamic behavior of peers [APV06a]. In particular, we addressed the following problems: peers becoming inaccessible in the merge-and-backward phase; peers that hold top data items becoming inaccessible in the data retrieval phase; late reception of score-lists by a peer, after its wait time has expired.

5.1.2.2 Top-k Queries in Super-peer Systems

In [BNST05], the authors propose a top-k query processing algorithm for Edutella, a super-peer network. 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. The top-k query processing algorithm works as follows. Given a query Q, the query originator sends Q to its super-peer, and sends Q to other super-peers. The super-peers forward Q to the relevant peers

connected to them. Each peer that has some data items relevant to Q scores them and sends its maximum scored data item to its super-peer. Each super-peer chooses the overall maximum scored item from all received data items. For determining the second best item, it only asks one peer, the one which has returned the first top item, to return its second top scored item. Then, the super-peer selects the overall second top item from the previously received items and the newly received item. Then, it asks the peer which has returned the second top item and so on until all k top items will be retrieved. Finally the super-peers send their top items to the super-peer of the query originator, to extract overall k top items, and to send them to the query originator.

5.1.2.3 Top-k Queries in DHTs

The main functionality of a DHT is to map a set of keys to the peers of the P2P system and lookup efficiently the responsible of any given key which is involved in the set. This offers efficient and scalable support for exact match queries. However, it is quite challenging to support top-k queries on top of DHTs [BAHS+06]. A simple solution is to retrieve all tuples of the relations involved in the query, compute the score of each retrieved tuple, and finally return the k tuples whose scores are the highest. However, this solution cannot scale up to a large number of stored tuples. Another 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, which is a network independent P2P system [AMPV06a], we proposed an efficient solution for processing top-k queries in DHTs [AMPV06b]. The solution is based on the TA algorithm [FLN03, GKB00, NR99] which is widely used in distributed systems. It can scale up to large numbers of peers and avoids any centralized data storage. The solution is based on a data storage mechanism that stores the shared data in the DHT in a fully distributed fashion. We describe this solution below.

Data storage mechanism

In the APPA data storage mechanism, peers store their relational data in the DHT with two complementary methods: tuple storage and attribute-value storage. With the tuple storage method, each tuple of a relation is stored in the DHT using its tuple identifier (*e.g.* its primary key) as the storage key. This enables looking up a tuple by its identifier.

Attribute value storage stores individually the attributes that may appear in a query's equality predicate or in a query's scoring function in the DHT. Thus, like database secondary indices, it allows looking up the tuples using their attribute values. The attribute value storage method has two important properties. 1) after retrieving an attribute value from the DHT, peers can retrieve easily the corresponding tuple of the attribute value; 2) attribute values that are relatively "close" are stored at the same peer. To provide the first property, the key, which is used for storing the entire tuple, is stored along with the attribute value. The second property is provided by using the concept of domain partitioning as follows. Consider an attribute a and let D_a be its domain of values. Assume there is a total order $<$ on D_a, *e.g.* D_a is numeric, string, date, etc. D_a is partitioned into n nonempty sub-domains $d_1, d_2, \ldots, d_n$ such that their union is equal to D_a, the intersection of any two different sub-domains is empty, and for each $v_1 \in d_i$ and $v_2 \in d_j$, if $i<j$ then

we have $v_1<v_2$. Upon attribute-value storage, the hash function is applied on the sub-domain of the attribute value. Thus, for the attribute values that fall in the same sub-domain, the storage key is the same; thereby they will be stored at the same peer. To avoid attribute storage skew, *i.e.* skewed distribution of attribute values within sub-domains, domain partitioning is done in such a way that attribute values are uniformly distributed in sub-domains. This technique uses histogram-based information that describes the distribution of values of the attribute.

Top-k query processing algorithm

Let Q be a given top-k query, f be its scoring function, and p_{int} be the peer at which Q is issued. For simplicity, let us assume that f is a monotonic scoring function. Let *scoring attributes* be the set of attributes that are passed to the scoring function as arguments. The top-k query processing algorithm, called DHTop, starts at p_{int} and proceeds in two phases: (1) prepare ordered lists of candidate sub-domains; (2) continuously retrieve candidate attribute values and their tuples until finding the k top tuples.

1. For each scoring attribute α, p_{int} prepares the list of sub-domains and orders them according to their positive impact on the scoring function, *i.e.* the sub-domains for which the scoring function is higher are at the beginning of the list. For each list, p_{int} removes from the list the sub-domains of which no member can satisfy *Q's* conditions. For instance, if there is a condition that enforce the scoring attribute to be equal to a constant, *e.g.* $\alpha = u$, then p_{int} removes from the list all its sub-domains but the sub-domain to which the constant value belongs. The list prepared in this phase for a scoring attribute α is denoted by CDL_α.
2. For each scoring attribute α in parallel, p_{int} proceeds as follows. It sends Q and α to the peer, say p, that is responsible for storing the values of the first sub-domain of CDL_α, and requests it to return the values of α that are stored at it. The values are returned to p_{int} in order of their positive impact on the scoring function. After receiving each attribute value, p_{int} retrieves its corresponding tuple, computes the tuple's score, and keeps it if its score is one of the k highest scores yet computed. This process continues until having k tuples whose scores are higher than a threshold which is computed based on the attribute values retrieved yet. If the attribute values that p returns to p_{int} are not sufficient for determining the k top tuples, p_{int} sends Q and α to the responsible of the second sub-domain of CDL_α.

The threshold is computed as follows. Let $\alpha_1, \alpha_2, \ldots, \alpha_m$ be the scoring attributes. Let $v_1, v_2, \ldots, v_m$ be the last values retrieved respectively for attributes $\alpha_1, \alpha_2, \ldots, \alpha_m$. The threshold is defined to be $\delta = f(v_1, v_2, \ldots, v_m)$. A main feature of the DHTop algorithm is that after retrieving each new attribute value the value of the threshold decreases. Thus, after retrieving a certain number of attribute values and their tuples, the threshold gets less than k of the retrieved tuples and the algorithm ends. It is analytically proved that DHTop works correctly for monotonic scoring functions and also for a large group of non-monotonic functions.

Although very effective, DHTop has the following limitations. First, it assumes top-k queries with no join operation, thus it is needed to extend this algorithm to support top-k join queries. Second, there are many scoring functions which are not supported by DHTop, *e.g.* many of polynomial functions.

5.2 Other complex queries

In this section, we deal with three kinds of complex queries: range queries, multi-attribute queries, and join queries. The processing of such queries in unstructured and super-peer networks does not raise any difficulty, *i.e.* range queries and multi-attribute queries can be processed like simple selection queries, and join queries can be processed as in distributed systems. However, processing these queries in DHTs is much more challenging. In the rest of this section, we focus on processing these complex queries in DHTs.

5.2.1 Range queries

Range queries are difficult to process in DHTs. With a range query, the user requests the data such that their attribute values are in a specific range, *e.g.* patients whose weight is between 50 and 60 kilograms. The main problem for supporting range queries in DHTs is that the hash function, which is used by the DHT to map data on peers, does not maintain data proximity, *i.e.* the hash of two close data may be two far numbers. There have been several proposals to support range queries on top of DHTs. In [GAA03], the authors rely on locality sensitive hashing to ensure that, with high probability, nearby values 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.

SkipNet [HJST+03] 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.

In [GS04], an adaptive mechanism for efficient processing of range queries in DHTs is proposed. It builds a logical Range Search Tree (RST) on top of a DHT, and stores the attribute values in the RST. A given range query is decomposed into a small number of sub-queries which are sent to a small number of peers in of the RST, and the query's answer is the union of the answers to the sub-queries. However, the problem is that the load is not uniformly distributed over the peers, *i.e.* the closer a peer is to the root of RST, the higher is its load.

Sahin et al. [SGAA02] extend the CAN system for d=2 such that a virtual hash space is constructed for each attribute as a 2-dimensional square bounded by the lower and higher values of the attribute's domain. The space is further partitioned into zones, and each zone is assigned to a peer. That peer stores the attribute values which are in the partition it owns, as well as routing information about adjacent zones. A given range query is routed to the peers which maintain the domain partitions involved in the range. Another extension of CAN is proposed in [AX02] which maps nearby ranges to nearby CAN zones. A subset of peers, called interval keepers, is responsible for sub-domains of the attribute's domain. When a range query is issued, it is routed to interval keepers, and via them to the peers which maintain the relevant data to the query. However, the two above methods are devised for 2-dimensional CAN systems, and it is hard to apply them to other DHTs such as Chord.

5.2.2 Join queries

Due to the specific method of DHTs for data distribution, *i.e.* using hash functions for choosing the location where the data should be stored, the execution of join queries over these systems is quite challenging. In [HHLT+03], the authors addressed the problem of join query processing in PIER. A key is constructed from the "namespace" and the "resource ID". There is a namespace for each relation and the resource ID is the primary key for base tuples in that relation. Queries are multicast to all peers in the two namespaces (relations) to be joined. Two algorithms are used for processing equi-join queries. 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.

To avoid multicasting the query to large numbers of peers, the authors in [TP03] propose to allocate a limited number of special powerful peers, called range guards, for the task of join query processing. They divide the domain of the join attributes, and each partition is dedicated to a range guard. Join queries are sent only to range guards, where the query is executed. Efficient selection of range guards and performance evaluation of their proposal were left as future work. However, the disadvantage of this solution is that the existence of the required special peers is not evident in many P2P networks.

5.2.3 Multi-Attribute queries

There has been some work for supporting multi-attribute queries on top of DHTs [BAHS+06]. Multi-Attribute Addressable Network (MAAN) [CFCS03] is a P2P system that addresses both multi-attribute and range queries in DHTs. It is built on top of Chord and is the first system that supports both query types in DHTs. MAAN executes multi-attribute range queries in $O(\log n + n * s_{min})$ 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. MAAN makes the assumption that a locality preserving hash function would ensure load balance. The authors acknowledge that there is a selectivity breakpoint at which full flooding becomes more efficient than their technique. Another drawback of MAAN is that it requires a fixed global schema which is known in advance to all peers.
In RDFPeers [CF04] the authors extend this solution to allow heterogeneity in peers schemas. Each peer contains RDF based data items described as triples <subject, predicate, object>. The triples are hashed onto MAAN peers. The experimental results show improvement in load balance, but no test for skewed query loads was done.

6 Conclusion

In this paper, we discussed schema based query processing in P2P systems. First, we introduced the three main kinds of P2P networks: unstructured, structured and super-peer. We briefly described each of these P2P networks, and compared them based on the main requirements for data management: autonomy, query expressiveness, efficiency, quality of service, fault-tolerance and security. Each kind of P2P network provides partial support for these requirements, *i.e.* there is no P2P network that supports all of the requirements. For example structured networks, *i.e.* DHTs, have low query expressiveness, super-peer networks are not fault-tolerant, and unstructured networks are usually inefficient.

Second, we discussed schema mapping in P2P systems. Due to their specific characteristics, *e.g.* the dynamic and autonomous nature of peers, P2P systems cannot rely on centralized global schemas which have been proposed for distributed database systems. We presented the techniques used in P2P systems. Although there has been significant progress, there are still many limitations for P2P schema mapping. In particular, these techniques typically need some kind of human intervention and the quality of the mappings is not very high.

Third, we presented the techniques for routing queries to relevant peers. We first described the algorithms of query routing in unstructured systems. The main concern in unstructured systems is how to route the query to obtain high quality answers while minimizing the communication cost. Usually, the algorithms where peers maintain some kind of statistics outperform the others. However, for highly dynamic systems, these algorithms may incur a high communication overhead without significant gains in answers' quality. We also discussed the problem of query routing in structured systems, particularly in DHTs. We presented the main routing geometries which are used in DHTs. We analyzed the routing properties of these geometries and compared them from the point of view of these properties.

Finally, we described the techniques for processing complex queries in P2P systems, in particular in DHTs. We focused on top-k queries, join queries, range queries, and multi-attribute queries.

Although significant progress has been made in P2P schema-based query processing, there are still many open issues. In particular, there are some limitations with the solutions proposed for schema mapping and query processing in DHTs.

Bibliography

[ABCM+03] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. *ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, 527-538, 2003.

[ACDD+03] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: a self-organizing structured P2P system. *ACM SIGMOD Record*, 32(3), 29-33, 2003.

[ACH03] K. Aberer, P. Cudré-Mauroux and M. Hauswirth. The chatty Web: emergent semantics through gossiping. *Int. Conf. on World Wide Web (WWW)*, 197-206, 2003.

[ACKL+02] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky and D. Werthimer. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11), 56-61, 2002.

[AHA03] D. Anwitaman, M. Hauswirth, and K. Aberer. Updates in highly unreliable, replicated peer-to-peer systems. *IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, 76-85, 2003.

[AM06] R. Akbarinia and V. Martins. Data management in the APPA P2P system. *Journal of Grid Computing*, To appear, 2007.

[AMPV06a] R. Akbarinia, V. Martins, E. Pacitti, and P. Valduriez. Design and implementation of APPA. *Global Data Management* (Eds. R. Baldoni, G. Cortese and F. Davide), IOS Press, 2006.

[AMPV06b] R. Akbarinia, V. Martins, E. Pacitti and P. Valduriez. Top-k query processing in the APPA P2P system. *Int. Conf. on High Performance Computing for Computational Science (VecPar)*, LNCS 4395, Springer, 158-171, 2007.

[AMPV04] R. Akbarinia, V. Martins, E. Pacitti and P. Valduriez. Replication and query processing in the APPA data management system. *Int. Workshop on Distributed Data and Structures (WDAS)*, 2004.

[APV06a] R. Akbarinia, E. Pacitti and P. Valduriez. Reducing network traffic in unstructured P2P systems using top-k queries. *Journal of Distributed and Parallel Databases*, 19(2-3), 67-86, 2006.

[APV06b] R. Akbarinia, E. Pacitti and P. Valduriez. An efficient mechanism for processing top-k queries in DHTs. *Journées Bases de Données Avancées (BDA)*, 2006.

[AS04] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4), 335-371, 2004.

[AX02] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *Proc. of the IEEE Int. Conf. on P2P computing*, 33-40, 2002.

[BAHS+06] R. Blanco, N. Ahmed, D. Hadaller, L.G.A. Sung, H. Li and M.A. Soliman. A survey of data management in peer-to-peer systems. Technical Report CS-2006-18, University of Waterloo, 2006.

[BBK01] C. Böhm, S. Berchtold and D.A. Keim. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3), 322-373, 2001.

[BGKM+02] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini and I. Zaihrayeu. Data management for peer-to-peer computing: a vision. In *Proc of the Int. Workshop on the Web and Databases (WebDB)*, 89-94, 2002.

[BKRS+04] A. Bhargava, K. Kothapalli, C. Riley, C. Scheideler, and M. Thober. Pagoda: a dynamic overlay network for routing, data management, and multicasting. In *Proc. of the ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, 170-179, 2004.

[Blo70] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Communications of the ACM*, 13(7), 422-426, 1970.

[BNST05] W.-T Balke, W. Nejdl, W. Siberski and U. Thaden. Progressive distributed top-k retrieval in peer-to-peer networks. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, 174-185, 2005.

[BO03] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, 28-39, 2003.

[CDKR02] M. Castro, P. Druschel, A-M. Kermarrec, and A. Rowstron. SCRIBE: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication (JSAC)*, 20(8), 1489-1499, 2002.

[CF04] M. Cai and M. Frank. RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. *Int. Conf. on World Wide Web (WWW)*, 650-657, 2004.

[CFCS03] M. Cai, M. Frank, J. Chen, and P. Szekely. MAAN: a multi-attribute addressable network for grid information services. *Int. Workshop on Grid Computing*, 184-191, 2003.

[CG02] A. Crespo, and H. Garcia-Molina. Routing indices for peer-to-peer systems. *IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, 23-33, 2002.

[CGM04] S. Chaudhuri, L. Gravano and A. Marian. Optimizing top-k selection queries over multimedia repositories. *IEEE Transactions on Knowledge and Data Engineering*, 16(8), 992- 1009, 2004.

[CJKR+03] M. Castro, M.B. Jones, A-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. *Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM)*, 1510-1520, 2003.

[CMHS+02] I. Clarke, S. Miller, T.W. Hong, O. Sandberg, and B. Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1), 40-49, 2002.

[CP02] P. Ciaccia and M. Patella. Searching in metric spaces with user-defined and approximate distances. *ACM Transactions on Database Systems*, 27(4), 398-437, 2002.

[CPMN03] F.M. Cuenca-Acuna, C. Peery, R.P. Martin and T.D. Nguyen. PlanetP: using gossiping to build content addressable peer-to-peer information sharing communities. *IEEE Int. Symp. on High Performance Distributed Computing (HPDC)*, 236-249, 2003.

[CRBL+03] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P systems scalable. *ACM SIGCOMM Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 407-418, 2003.

[CW04] P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. *ACM Symp. on Principles of Distributed Computing (PODC)*, 206-215, 2004.

[DFM00] R. Dingledine, M. Freedman, and D. Molnar. The FreeHaven project: distributed anonymous storage service. *Workshop on Design Issues in Anonymity and Unobservability*, 67-95, 2000.

[DGY03] N. Daswani, H. Garcia-Molina, and B. Yang. Open problems in data-sharing peer-to-peer systems. *Int. Conf. on Database Theory (ICDT)*, 1-15, 2003.

[DMDD+03] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos and A. Halevy. Learning to match ontologies on the semantic web. *VLDB Journal*, 12(4), 303-319, 2003.

[DMNK02] A.P. DeVries, N. Mamoulis, N. Nes and M.L. Kersten. Efficient k-NN search on vertically decomposed data. *ACM Int. Conf. on Management of Data (SIGMOD)*, 322-333, 2002.

[FLN03] R. Fagin, J. Lotem and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4), 614-656, 2003.

[GAA03] A. Gupta, D. Agrawal and A. El Abbadi. Approximate range selection queries in peer-to-peer systems. *First Biennial Conference on Innovative Data Systems Research (CIDR)*, 141-151, 2003.

[Gen06] Genome@Home. http://genomeathome.stanford.edu/.

[GGGR+03] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. *ACM Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 381-394, 2003.

[GKB00] U. Güntzer, W. Kießling and W.-T Balke. Optimizing multi-feature queries for image databases. *Int. Conf. on Very Large Databases (VLDB)*, 419-428, 2000.

[Gnu06] Gnutella. http://www.gnutelliums.com/.

[GS04] J. Gao and P. Steenkiste. An adaptive protocol for efficient support of range queries in DHT-based systems. *IEEE Int. Conf. on Network Protocols (ICNP)*, 239-250, 2004.

[HHHL+02] M. Harren, J.M. Hellerstein, R. Huebsch, B.T. Loo, S. Shenker and I. Stoica. Complex queries in DHT-based peer-to-peer networks. *Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 242-259, 2002.

[HHLT+03] R. Huebsch, J. Hellerstein, N. Lanham, B. Thau Loo, S. Shenker, and I. Stoica. Querying the internet with PIER. *Int. Conf. on Very Large Databases (VLDB)*, 321-332, 2003.

[HIMT03] A. Halevy, Z. Ives, P. Mork, and I. Tatarinov. Piazza: data management infrastructure for semantic web applications. *Int. Conf. on World Wide Web (WWW)*, 556-567, 2003.

[HJST+03] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: a scalable overlay network with practical locality properties. *USENIX Symp. on Internet Technologies and Systems*, 113-126, 2003.

[HS03] G.R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4), 517-580, 2003.

[Icq06] ICQ. http://www.icq.com/.

[JAB01] M. Jovanovic, F. Annexstein, and K. Berman. Scalability issues in large peer-to-peer networks: a case study of Gnutella. Technical report, ECECS Department, University of Cincinnati, 2001.

[Jab03] Jabber. http://www.jabber.org/.

[Jov00] M. Jovanovic. Modelling large-scale peer-to-peer networks and a case study of Gnutella. Master's thesis, ECECS Department, University of Cincinnati, 2000.

[JWZ03] R. Janakiraman, M. Waldvogel, and Q. Zhang. Indra: a peer-to-peer approach to network intrusion detection and prevention. *IEEE Int. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 226-231, 2003.

[Jxt06] JXTA. http://www.jxta.org/.

[Kaz06] KaZaA. http://www.kazaa.com/.

[KBCC+00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao. OceanStore: an architecture for global-scale persistent storage. *ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 190-201, 2000.

[KGZ02] V. Kalogeraki, D. Gunopoulos, D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. *ACM Int. Conf. on Information and Knowledge Management (CIKM)*, 300-307, 2002.

[KOTZ04] N. Koudas, B.C. Ooi, K.L. Tan and R. Zhang. Approximate NN queries on streams with guaranteed error/performance bounds. *Int. Conf. on Very Large Databases (VLDB)*, 804-815, 2004.

[KR02] A.V.M. Keromytis and D. Rubenstein. SOS: secure overlay services. *ACM SIGCOMM Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 61-72, 2002.

[LCCL+02] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. *ACM Int. Conf. on Supercomputing (ICS)*, 84-95, 2002.

[Len02] M. Lenzerini. Data integration: a theoretical perspective. *ACM Symp. on Principles of Distributed Computing (PODC)*, 233-246, 2002.

[LRSS02] K. Lakshminarayanan, A. Rao, I. Stoica, and S. Shenker. Flexible and robust large scale multicast using i3. Technical report CSD-02-1187, University of California, Berkeley, 2002.

[LSP03] S. Larson, C. Snow, and V. Pande. Folding@Home and Genome@Home: using distributed computing to tackle previously intractable problems in computational biology. *Modern Methods in Computational Biology*. (Ed. R. Grant). Horizon Press, 2003.

[LW06] X. Li and J. Wu. Searching techniques in peer-to-peer networks. *Handbook of Theoretical and Algorithmic Aspects of Ad Hoc, Sensor, and Peer-to-Peer Networks.* (Eds. W. Zheng, X. Liu, S. Shi, J. Hu and H. Dong). Auerbach Publications, 2006.

[MAPV06] V. Martins, R. Akbarinia, E. Pacitti, and P. Valduriez. Reconciliation in the APPA P2P system. *IEEE Int. Conf. on Parallel and Distributed Systems (ICPADS)*, 401-410, 2006.

[MBDH05] J. Madhavan, P.A. Bernstein, A. Doan, A. Halevy. Corpus-based schema matching. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, 57-68, 2005.

[MK02] D. Menascé and L. Kanchanapalli. Probabilistic scalable P2P resource location services. *SIGMETRICS Performance Evaluation Review*, 30(2), 48-58, 2002.

[MM02] P. Maymounkov and D. Mazieres. Kademlia: a peer-to-peer information system based on the XOR metric. *Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 53-65, 2002.

[MNR02] D. Malkhi, M. Naor and D. Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. *ACM Symp. on Principles of Distributed Computing (PODC)*, 183-192, 2002.

[MP04] P. McBrien and A. Poulovassilis. Defining peer-to-peer data integration using both as view rules. *Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing*, 91-107, 2004.

[MTW05] S. Michel, P. Triantafillou and G. Weikum. KLEE: a framework for distributed top-k query algorithms. *Int. Conf. on Very Large Databases (VLDB)*, 637-648, 2005.

[Nap06] Napster. http://www.napster.com/.

[NR99] S. Nepal and M.V. Ramakrishna. Query processing issues in image (multimedia) databases. *Int. Conf. on Data Engineering (ICDE)*, 22-29, 1999.

[NSS03] W. Nejdl, W. Siberski, and M. Sintek. Design issues and challenges for RDF- and schema-based peer-to-peer systems. *ACM SIGMOD Record*, 32(3), 41-46, 2003.

[NWQD+02] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. Edutella: a P2P networking infrastructure based on RDF. *Int. Conf. on World Wide Web (WWW)*, 604-615, 2002.

[OST03] B. Ooi, Y. Shu, and K-L. Tan. Relational data sharing in peer-based data management systems. *ACM SIGMOD Record*, 32(3), 59-64, 2003.

[OV99] T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. 2nd Ed., Prentice Hall, 1999.

[PRR97] C.G. Plaxton, R. Rajaraman, and A.W. Richa. Accessing nearby copies of replicated objects in a distributed environment. *ACM Symp. on Parallel Algorithms and Architectures* (*SPAA)*, 311-320, 1997.

[PS00] E. Pacitti and E. Simon. Update propagation strategies to improve freshness in lazy master replicated databases. *VLDB Journal*, 8(3-4), 305-318, 2000.

[PZS96] M. Persin, J. Zobel and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society for Information Science*, 47(10), 749-764, 1996.

[RD01a] A. Rowstron and P. Druschel. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware)*, 329-350, 2001.

[RD01b] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *ACM Symp. on Operating Systems Principles (SOSP)*, 188-201, 2001.

[RFHK+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *ACM SIGCOMM Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 161-172, 2001.

[RK02] S. Rhea and J. Kubiatowicz. Probabilistic location and routing. *Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM)*, 1248- 1257, 2002.

[Set06] Seti@home. http://setiathome.ssl.berkeley.edu.

[SGAA02] O. Sahin, A. Gupta, D. Agrawal, and A. El Abbadi. Query processing over peer-to-peer data sharing systems. Technical Report 2002-28, University of California, Santa Barbara, 2002.

[SMKK+01] I. Stoica, R. Morris, D.R. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. *ACM Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 149-160, 2001.

[SOTZ03] W. Siong Ng, B. Ooi, K-L. Tan, and A. Zhou. PeerDB: a P2P-based system for distributed data sharing. *Int. Conf. on Data Engineering (ICDE)*, 633-644, 2003.

[SSDN02] M. Schlosser, M. Sintek, S. Decker and W. Nejdl. HyperCuP. Technical Report, Stanford University, 2002.

[TIMH+03] I. Tatarinov, Z.G. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyska, G. Miklau, and P. Mork. The Piazza peer data management project. *ACM SIGMOD Record*, 32(3), 47-52, 2003.

[TP03] P. Triantafillou and T. Pitoura. Towards a unifying framework for complex query processing over structured peer-to-peer data networks. *Int.Workshop on Databases, Information Systems and Peer-to-Peer Computing*, 169-183, 2003.

[TR03a] D. Tsoumakos and N. Roussopoulos. Adaptive probabilistic search (APS) for peer-to-peer networks. Technical Report, University of Maryland, 2003.

[TR03b] D. Tsoumakos and N. Roussopoulos. A Comparison of peer-to-peer search methods. *Int. Workshop on the Web and Databases (WebDB)*, 61-66, 2003.

[TRV98] A. Tomasic, L. Raschid, and P. Valduriez. Scaling access to heterogeneous data sources with DISCO. *IEEE Transactions on Knowledge and Data Engineering*, 10(5), 808-823, 1998.

[TV00] A. Tanaka and P. Valduriez. The Ecobase environmental information system: applications, architecture and open issues. *ACM SIGMOD Record*, 30(3), 70-75, 2000.

[Ull97] J.D. Ullman. Information integration using logical views. In *Proc. of the Int. Conf. on Database Theory (ICDT)*, 19-40, 1997.

[Val93] P. Valduriez. Parallel database systems: open problems and new issues. *Distributed and Parallel Databases*, 1(2), 137-165, 1993.

[VAS04] V. Vlachos, S. Androutsellis-Theotokis, and D. Spinellis. Security applications of peer-to-peer networks. *Computer Networks Journal*, 45(2), 195-205, 2004.

[W3C01] W3C: World Wide Web consortium on semantic Web activities. http://www.w3.org/2001/sw, 2001.

[WA91] A.N. Wilschut and P.M.G. Apers. Dataflow query execution in a parallel main-memory environment. *Int. Conf. on Parallel and Distributed Information Systems (PDIS)*, 68-77, 1991.

[WAL00] M. Waldman, R. AD, and C. LF. Publius: a robust, tamper-evident, censorship-resistant, web publishing system. *USENIX Security Symp.*, 59-72, 2000.

[Wie92] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3), 38-49, 1992.

[XCK06] Y. Xia, S. Chen, and V. Korgaonkar. Load balancing with multiple hash functions in peer-to-peer networks. *IEEE Int. Conf. on Parallel and Distributed Systems (ICPADS)*, 411-420, 2006.

[YG02] B. Yang, H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proc. of the IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, 5-14, 2002.

[ZHSR+04] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications (JSAC),* 22(1), 41-53, 2004.