Peer-to-Peer Management of XML Data: Issues and Research Challenges

Georgia Koloniari and Evaggelia Pitoura
Computer Science Department, University of Ioannina, Greece
{kgeorgia, pitoura}@cs.uoi.gr

ABSTRACT

Peer-to-peer (p2p) systems are attracting increasing attention as an efficient means of sharing data among large, diverse and dynamic sets of users. The widespread use of XML as a standard for representing and exchanging data in the Internet suggests using XML for describing data shared in a p2p system. However, sharing XML data imposes new challenges in p2p systems related to supporting advanced querying beyond simple keyword-based retrieval. In this paper, we focus on data management issues for processing XML data in a p2p setting, namely indexing, replication, clustering and query routing and processing. For each of these topics, we present the issues that arise, survey related research and highlight open research problems.

1. INTRODUCTION

The popularity of file sharing systems (such as Napster [33] and Gnutella [17]) has resulted in attracting much current research in peer-to-peer (p2p) architectures as an efficient means of sharing data. Peer-to-peer computing [32] refers to a form of distributed computing that involves a large number of autonomous computing nodes (the peers) that cooperate to share resources and services. The peers form logical overlay networks by establishing links to some other peers they know or discover. A user in a p2p system issues queries that describe data of interest. The queries are propagated through the overlay network to locate peers that provide data relevant to the query and any matching results are returned to the user.

Although the best-known application of p2p systems is file sharing (for example, music files in Napster), p2p system applications go beyond data sharing. Peer-to-peer computing is also a way of implementing systems based on the notion of increased decentralization and self-organization of systems, applications, or simply algorithms. By leveraging vast amounts of computing power, storage, and connectivity from personal computers distributed around the world, p2p systems provide a substrate for a variety of applications such as network monitoring and routing, web search and large scale event/notification systems.

XML [54] has evolved as the new standard for the representation and exchange of semistructured data on the Internet. Several application domains for XML already show that XML is inherently distributed on the Web, for example, Web services that use XML-based descriptions in WSDL and exchange XML messages with SOAP, e-commerce and e-business, collaborative authoring of large electronic documents and management of large-scale network directories. All these applications demonstrate that much of the traffic and data available in the Internet are already represented in XML format. Thus, it is natural to assume that much of the data in a p2p system is already represented in XML format. Furthermore, the deployment of XML as the underlying data model for p2p systems can provide a solution to two important issues in current p2p systems: the limited expressiveness of the available query languages and the heterogeneity of data.

In most p2p systems, different users and applications employ various formats and schemas to describe their data. A user is usually unaware of the schemas remote peers use. Moreover, some application domains, such as health-related applications, use sensitive data that are required not to be exposed to all users for privacy reasons. Therefore, there is a need for a query language that can work with incomplete or no-schema knowledge but also capture whatever semantic knowledge is available. The flexibility of XML in representing heterogeneous data that follow different schemas makes it suitable for distributed applications where the data are either native XML documents or XML descriptions of data or services that are represented in various formats in the underlying sources (i.e. in relational databases).

With regards to the query language, in most p2p systems, users specify the data they are interested in through simple keyword-based queries. These keywords are matched against the names of the shared files and any results are returned to the user. Often, most results returned are not relevant to what the user is interested in. Thus, new more expressive languages are needed to describe and query the shared data. XML seems to be a promising candidate in this direction, since it enables more precise search through provision of structural and self-describing metadata that allow for context and category-based search.

Traditionally, research work in XML querying has been following one of the two paths: the structured query approach (XQuery [7]) and the keyword-based approach (XKeyword [21], XSEarch [9]). While structured queries work effectively with the inherent structure of XML data and can convey complex semantic meaning, they require from the user to know the schema (or part of the schema) of the XML data to write the right query. The problem becomes even more difficult when the user has to deal with data from different schemas where the query requires rewriting before it can be evaluated. On the other hand, keyword-based searches do not require any knowledge about the underlying data schema but they also do not allow the users to convey semantic knowledge in their queries.

Motivated by the important role of XML in p2p systems, in this paper, we survey recent work on distributed processing of XML data in p2p systems. We focus on data management issues, such as indexing (Section 3), clustering (Section

4), replication (Section 5) and query processing and routing (Section 6). Although, schema integration of heterogeneous data is one of major issues in p2p processing [20], [22], it is beyond the scope of this survey.

2. P2P DATA MANAGEMENT

In this section, we provide a classification of p2p systems and describe some of their distinctive characteristics.

2.1 P2p characteristics

Scalability: While even in large traditional distributed systems, the number of participating nodes is in the order of hundreds; p2p systems must achieve scalability at the Internet-level.

Decentralization: Peer-to-peer computing is an alternative to the centralized and client-server models of computing, where there is typically a single or small cluster of servers and many clients. In its purest form, the peer-to-peer model has no concept of a server; rather all participants are equal, with each node given both server and client capabilities. Between the centralized and the pure peer-to-peer approach, there are hybrid p2p systems, in which some of the participants, called superpeers, have extended responsibilities and control over the others. The superpeers are often peers that have increased capabilities (storage and processing) and good stability properties.

Autonomy: We distinguish four kinds of autonomy, (i) storage, (ii) execution, (iii) lifetime and (iv) connection autonomy. *Storage autonomy* refers to the freedom of what a node in the system stores. In traditional distributed systems with central administration, the system enforces to the nodes which data items or indexes to store. P2p systems are self-configured: each peer stores its own data according to its interests and needs. Storage autonomy has another dimension related to *ownership* of data. This kind of autonomy allows a peer to determine which other peers in the system can store its own data or index information about its data.

Execution autonomy refers to the ability of a node to answer queries and change its own data. *Lifetime autonomy* refers to the freedom of each node to join and leave the system arbitrarily. In contrast to traditional distributed systems, p2p systems support this kind of dynamism and many issues concerning fault-tolerance, self-maintenance, ad-hoc connectivity and data availability arise from this requirement. Since the peers leave the system very frequently, a p2p system has to provide mechanisms to cope with these disconnections without causing significant problems in its operation. Furthermore, self-maintenance techniques should be used to deal with the frequent changes in the network. Finally, *connection autonomy* refers to the form of the overlay network in a p2p system. It enables a peer to select with how many and which peers it will connect to, based for example on trust or friendship with other users.

Due to the various forms of autonomy, many peers may exhibit selfish behavior. Selfish peers may refuse to evaluate or propagate queries or store index information or data copies. Such selfish peers try to exploit the resources and services provided by the system, without being willing to offer anything back to the peer community. To prevent this kind of behavior, p2p systems must provide incentives to peers for sharing their data and participating in query processing.

Other challenges that p2p systems need to cope with include anonymity, security and administration transparency.

A nice introduction to p2p systems and their basic goals and characteristics can be found in [32]. Some research challenges with emphasis on search are presented in [40], search and security issues are discussed in [13] and initial research ideas on data management in [19].

2.2 Types of p2p data management systems

We classify peer-to-peer data management systems based on (i) the degree of decentralization, (ii) the topology of the overlay network, (iii) the way information is distributed among the nodes and (iv) the type of data they store. Regarding the degree of decentralization, this varies from *pure* p2p systems, where all peers have equal roles, to *hybrid* architectures, where specific peers (the superpeers) are assigned different roles. Topology refers to the way the nodes in the p2p system are interconnected. Example topologies include the star, ring and the grid topology. In hybrid architectures, the topology of the superpeers may differ from that of the other peers. For instance, the superpeers may be fully inter-connected, while each simple peer is only connected with a single superpeer.

With regards to the distribution of information among the peers, we distinguish between structured and unstructured p2p systems. In *unstructured* p2p systems, there is no assumption about the distribution of data to the peers. Unstructured p2p systems can be further distinguished between systems that use indexes and those that are based on flooding and its variations. Topologies in unstructured p2p systems are usually not restricted to some regular structure, however, they may be regulated, for instance, by setting limits on the number of neighbors each peer can have.

In *structured* p2p systems, data items (or indexes of data items) are placed at specific nodes. Usually the distribution of data items to the peers is based on distributed hashing (DHTs) (such as in CAN [38] and Chord [45]). In DHTs, each item is associated with a key and each peer is assigned a range of keys and thus items. We make an additional distinction regarding structured p2p systems based on whether they assign to peers actual data items or indexes of items. Most DHT-based structured p2p systems follow a strict topology (such as a ring or torus) in which each peer has a specific number of neighbors. For example, in Chord, a hash function creates an m-bit identifier space. Identifiers are ordered on an identifier circle modulo 2^m, that forms the Chord virtual ring. As new peers join the system, their identifier, produced by hashing their IP address and port, is used for mapping them to the virtual ring. Data keys are also hashed and distributed to the peers according to their hash value, so that each key is assigned to its successor peer, which is the peer with the nearest hash-value traveling the ring clockwise. When a peer n joins the system, certain keys previously assigned to n's successor now become assigned to n. When peer n leaves the network, all of its assigned keys are reassigned to n's successor. Each peer maintains a finger table with its successors. Query routing proceeds by consulting the finger tables to locate the peer with the identifier closer to the search key. Other DHT-based systems exploit a less strict topology. For instance, P-Grid [1] builds a virtual distributed search tree which may be unbalanced.

DHT-based p2p systems support efficient key lookup (e.g., of order O(logn) in Chord). However, in most structured p2p systems, both storage and connection autonomy is compromised. The peers are forced to store information for data

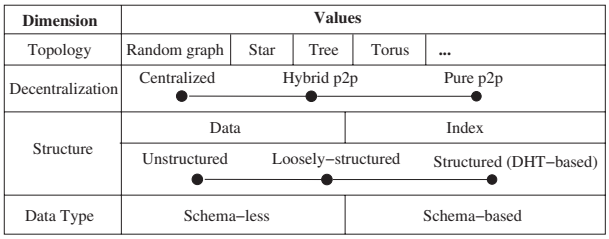


Figure 1: Classification of p2p systems

items assigned to them by distributed hashing and also follow a regulated topology. In addition, structured p2p systems require sophisticated load balancing procedures to cope with the increased demand for popular items and the dynamic behavior of the peers. There are also systems that are not based on distributed hashing, but do impose some structure. Such systems organize the overlay network into groups of peers with similar properties. We call such systems *loosely-structured* or *clustered* p2p systems.

Note that structured, unstructured and loosely structured p2p systems can use either pure or hybrid p2p architectures. For example in the case of structured DHT-based p2p systems, the DHT may be built only upon the superpeers, which have to follow the strict topology imposed by the system, while simple peers may connect to one or more superpeers without following any particular structure.

Finally, we distinguish between schema-less and schema-based p2p systems. In schema-based p2p systems, the peers use explicit schemas to describe their content. Schemas can be heterogeneous. A potential candidate for describing resources in p2p systems is the Resource Description Framework (RDF). RDF [39] is used to annotate resources on the Web, thus providing the means by which computer systems can exchange and comprehend data. RDF schemas are flexible and can evolve over time by allowing the easy extension of schemas with additional properties, thus being suitable for dynamic p2p systems. RDF usually uses an XML-based syntax to represent the metadata of the described resources. Apart from representing RDF descriptions in XML, RDF schemas can be used to provide semantic meaning for XML documents by using ontologies that are also described in RDF [23]. Figure 1 summarizes our taxonomy.

3. DISTRIBUTED INDEXES

In a p2p system, queries are initiated at various peers. These queries may require data that are located at a large number of peers distributed over the system. Traditionally, distributed systems use centralized or distributed indexes (catalogs) to store information about the location of data. For query processing, the indexes are consulted and the queries are sent to the appropriate nodes and evaluated there. Maintaining indexes in p2p systems poses additional requirements. In particular, indexes in p2p systems must support frequent updates, as peers join and leave the system constantly. Furthermore, the indexes need to be highly scalable, since the number of peers reaches Internet-scale, while in traditional distributed systems, the number of participating nodes is much smaller and controlled.

3.1 Types of p2p indexes

There are three basic approaches regarding indexes: maintaining (i) no index, (ii) a centralized index and (iii) a distributed index. When there is *no index* (such as in Gnutella [17]), some form of flooding is used for routing: the peer where the query is initiated contacts its neighbors in the overlay network, which in turn contact their own neighbors until the requested items are located or some system-defined bound is reached. Flooding does not compromise storage autonomy but incurs large network overheads. In the case of a *centralized index* (such as in Napster [33]), information about the contents of all peers in the system is maintained at a single peer. Peers that enter the system publish information about their data in this central index that is consulted when a query is submitted. The drawback of this approach is that the central index server becomes a bottleneck and a single point of failure. Maintaining replicas of the centralized index may increase reliability and scalability but still fails to handle efficiently the huge number of updates.

The distribution of the index depends on the overlay topology and on whether the system is structured or unstructured. In *structured p2p systems*, each peer stores index information for the data assigned to it by the hash function. For the evaluation of a query, its hash value is computed and the query is routed through the peer overlay network towards the peer that is responsible for storing the corresponding value.

In *unstructured p2p systems*, a popular form of distributed indexing is routing indexes [10]. The *routing indexes* of a peer summarize information about the contents of other reachable peers; they are used during routing to direct the queries towards the peers that are expected to hold relevant data. Since knowledge about all peers in the network is infeasible for scalability reasons, *horizons* are used to limit the number of other peers for which each peer stores information. Each index of a peer summarizes information about the data of all other peers that can be reached at a maximum distance h, where h is called the *radius* of the horizon. A peer may have just one such index or one for each of its links, summarizing the information of all peers on the path starting from this link and at a maximum distance h. Another distribution strategy is a hierarchical topology. In this case, the peers form one or multiple hierarchies. Each peer in the hierarchy stores information about the peers in its subtree, and the roots of the hierarchies are interconnected. The peers in the upper layers of the hierarchy assume most of the load and use their indexes to forward the queries to parts of the hierarchy where relevant data may be found.

Distributed indexes can also vary according to the degree of decentralization of the p2p system. In hybrid p2p systems, each superpeer is responsible for a number of other peers for which it stores index information. The superpeers are interconnected with each other following either a structured or an unstructured architecture. Each query is forwarded to a superpeer that is responsible for propagating it to the relevant peers or superpeers using its indexes. Query routing follows different protocols among the superpeers and the peers of the system and different types of indexes are used between superpeers and between superpeers and peers.

When using XML as the underlying data format, additional requirements arise for p2p indexes. In partcular, for XML documents, we need both *value* and *path indexes* for addressing the content as well as the structure of documents. Commonly used path indexes are indexes that assume an unordered tree structured data model and consist of a tree structure that summarizes path information [18]. Evaluating a query consists of traversing the path tree and match-

N1	D1: device/printer/postscript, D2: device/camera
N2	D3: device/printer/postscript, device/local/printer
N3	D4: device/camera
N4	D5: device/local/printer, D6: device/network/printer
N5	D7: printer/laser/color, device/camera
N6	D8: network/printer/laser
N7	D9: book/databases/greek
N8	D10: book/databases/english

Figure 2: Example of XML data distributed at peers

ing the path expression against the tree nodes. Other approaches assume that the data follow an arbitrary graph model. These indexes construct reduced graphs [36] that summarize all paths in the original data graph, by collapsing nodes that are equivalent (two nodes are equivalent if the paths from the root to them are the same).

Next, we describe in detail how indexes are adopted in some structured and unstructured p2p systems to support XML. We shall use the simple example depicted in Fig. 2 of 8 peers and their data, where for example, peer $N1$ stores 2 documents $D1$ and $D2$ containing paths "device/printer/postscript" and "device/camera" respectively.

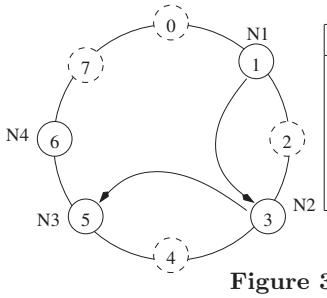
3.2 XML indexes in structured p2p systems

The use of XML as the format for data representation introduces additional problems in structured p2p systems. Most current structured p2p systems use document names as the data keys that are mapped to the underlying virtual network. Recent research [44], [49] has extended structured p2p systems by exploiting the content of documents for determining the keys. In particular, a vector describing each document is extracted and used as the key to map the documents to the virtual multi-dimensional space of the network. These vectors, used in information retrieval applications, typically consist of the document's keywords weighted by the frequencies of their appearance. The extracted vector is of much higher dimension than the dimension of the virtual space of the network. Thus, dimensionality reduction is required. This reduction should cause a minimal distortion, that is, the distance of the initial vectors should be approximated by the distance of the new vectors with the reduced dimensions. Vectors consisting of keywords and their corresponding weights are not suitable for representing XML data, since they do not capture the relationships between the elements (hierarchical structure). Thus, the challenge in this context is mapping the appropriate index keys for XML (both value and path indexes) to the multi-dimensional space created by distributed hashing.

In [15], a distributed catalog framework based on a structured p2p system is proposed. The system uses Chord [45] as the overlay network. The distributed catalog stores sets of key-summaries information for all the peers. The *keys* for XML data are either element or attribute names. The summaries that correspond to each key are either structural or value summaries. The *structural summaries* of a key are all possible paths leading to that key. The type of the *value summary* depends on the domain of the key, i.e. histograms are used for arithmetic keys. For each new peer that enters the system, each key-summary pair is inserted in the system by the Chord protocol according to the hash values of the keys. The class of supported XPath queries are of the form: $p = a_1[b_1]/a_2[b_2]/.../a_n[b_n]$ *op value*, where each a_i is a key and each b_i a path. The structural part of the query is handled using the structural catalog information, while the value predicates use the value summaries. For query routing, first all the simple paths ($sp_i = /a_{i1}/a_{i2}/.../a_{i,mi}$ *op value*) are extracted from the query. The peer responsible for the next $a_{i,mi}$ is found and the set of candidate peers for sp_i are retrieved using the catalog of that peer. The intersection of all the candidate peers that are produced after all the simple paths are processed is the set of peers that should receive the query.

Figure 3 shows an example of routing in Chord and its extension in [15]. The circles correspond to peers, while dotted circles correspond to logical positions (nodes) in the Chord ring that are not currently occupied by actual peers. The numbers within the circles correspond to the identifier of each logical node in the virtual space, while the labels next to them show which peers occupy the positions. The system has four peers $N1$ to $N4$ whose data are presented in Fig. 2. The table presents the index information that each peer holds for both Chord and its extension. The first column presents the finger tables of each peer, i.e., $N1$ knows that the successor of identifier 2 is peer 3, of identifier 3 again peer 3 and of identifier 5 peer 5. The finger tables are the same for both systems. For Chord, the index keys are the names of the files shared by the peers. Therefore, the second column of the table shows which files are assigned to each peer after the hash function is applied to their names. In the extended Chord (third column of the table), the indexed keys are the elements found in the documents along with their structural summaries denoted by S_i. Each S_i contains all the possible paths that lead to the corresponding indexed key in the contents of peer Ni. Let us assume that $N1$ issues the query: "local/printer". This query is not supported in Chord, therefore to retrieve this information the user must know the names of the files that contain it and pose a query using these names. On the other hand, the extended Chord is designed to support exactly this kind of queries. The arrows show the route of this query in the extended Chord.

XP2P [4] also extends Chord to support XML data. The system assumes that each peer stores a set of XML fragments (subtrees of XML data). In addition, each peer stores the local content of the user's fragments and their related path expressions that are the lists of each fragment's child fragments (path expressions stored as PCDATA within subtags in the fragment) and their super fragment (a path expression of the fragment which is the ancestor of the current fragment). These expressions are hashed into the Chord virtual space. The hashing technique used is different from that used in Chord. In particular, a fingerprinting technique is proposed based on [37]. The produced fingerprints are shorter than the hash keys used in Chord and support a concatenation property that allows the computation of the tokens associated with path expressions to proceed incrementally. Partial and full match lookups are supported, where in the first case, a match to a fragment is returned without unfolding its child fragments, while in the latter case, all the sub tags of the fragment are unfolded and the corresponding child fragments are retrieved. The queries are fingerpinted as well and when the fingerprint of a query (either in full or partial lookup) matches the fingerpint of a data fragment, the results are located by the lookup functionality of Chord. If the system cannot find a match, for instance if some peers are temporarily unavailable, additional techniques based on gradually pruning the query path are deployed to provide the user with at least a partial match.



Peer	Finger Table	Keys (Chord)	Keys–Summaries (extended Chord)
N1	2:3, 3:3, 5:5	D1	camera–S1{device}, S3{device}
N2	4:5, 5:5, 7:1	D2, D3	printer–S1{device}, S2{device, device/local} S4{device/network, device/local}
N3	6:6, 5:5, 7:1	D4, D5	–
N4	7:1, 0:1, 1:1	D6	postscript–S1{device/printer}

Figure 3: Indexes in structured p2p systems

RDFPeers [6] are based on MAAN (Multi-Attribute Addressable Network) which extends Chord to answer multi-attribute and range queries. Each RDF is viewed as a (subject, predicate, object) triple. Each triple is hashed and stored for each of its values in the corresponding positions in the Chord ring. For arithmetic attributes, MAAN uses order preserving hash functions so as to place close values to neighboring peers in the ring for the evaluation of range queries. Each query is transformed into triples and each value is searched as in Chord.

A DHT-based approach based on CAN is presented in [53]. The system presumes that the schema of the data is known by all peers. An overlay network similar to CAN is built where each dimension in the virtual multi-dimensional space corresponds to either a path level (a level of the path expression corresponding to some element name) or a unique attribute name on a specific path level. The dimensionality of the virtual space depends on the maximum depth of the path expressions and the number of distinct attributes each path level has. The virtual space is viewed as a hyper-rectangle and each distinct path corresponds to a logical node in the overlay network. The overall hyper-rectangles are disjointly partitioned among sub hyper-rectangles with exactly one logical node corresponding to each one of them. Each piece of XML data is mapped to a logical node according to its coordinates that are derived by hashing each element name and attribute that corresponds to each of the dimensions. Each peer keeps catalog information about all the paths that are mapped to it along with its own coordinates and its corresponding hyper-rectangle. Additionally, each peer keeps a routing table where it stores tuples of the form (coordinate, hyper-rectangle, address) for its neighbors in the virtual space. When a query is issued, it is also hashed to provide the query coordinates. When the queries consist of absolute location paths with only parent-child axis, the routing can be easily done by using the CAN routing mechanism which is enhanced in order to find the closest neighbor for propagating the query (finding the closest hyper-rectangle). If the query is more complex, it is transformed into one or more absolute location paths and the same mechanism is deployed.

In [34], a hybrid structured (non-DHT) p2p architecture is presented. The superpeers are organized into a hypercube topology that supports efficient broadcasting, while (non-super) peers connect to superpeers in a star-like fashion where each peer connects to only one superpeer. RDF metadata are used to describe the content of peers and to build routing indexes. Queries and answers to queries are also represented using RDF metadata. Two kinds of indexes are maintained at the superpeers: superpeer/peer indexes (SP/P) and superpeer/superpeer (SP/SP) indexes. SP/P indexes at a superpeer store information about metadata usage at each peer connected to it. This includes schema information such as schemas or attributes used, as well as possibly conventional indexes on attribute values. SP/SP indexes contain the same kind of information as SP/Ps, but refer to the direct superpeer neighbors of a superpeer. Queries are forwarded to superpeer neighbors according to the SP/SP indexes and then sent to the connected peers based on the SP/P indexes.

3.3 XML indexes in unstructured p2p systems

In unstructured p2p systems, research efforts focus on building space efficient routing indexes for XML documents. Most approaches build path indexes with the use of aggregation and suitable encoding schemes for the paths.

Figure 4 shows an example of query routing when using simple routing indexes such as in [10] and when XML-based routing indexes are used. The peers $N1$ to $N4$ hold the same data as in Fig. 2. Let us assume that the horizon is of radius 2. The gray circles represent the peers that are within peers $N1$ horizon. The table shows the routing indexes of peers $N1$ and $N3$ and their edges, for both simple routing indexes and path-based ones. Assume that $N1$ issues the same query as in the example of Fig. 3, that is "local/printer". If path indexes are not supported, the user must know the names of the files that contain the requested path expression. The arrows show the route the query follows when path-based routing indexes are used.

Two architectures for distributing the routing indexes, namely, the open and the agreement model, that differ in the degrees of shared knowledge among the peers, are proposed in [27]. In the open model, each peer is allowed to know about and potentially communicate with every other peer, while in the agreement model, each peer enters into bilateral agreements with some other peers called its neighbors. Path indexes are used as the internal organization of the routing indexes which maintain pointers from each path to the corresponding peers that contain it. Since the information included in a path index can grow excessively, aggregating paths with common prefixes is proposed to accommodate the routing index in a given space overhead.

Processing of containment queries in p2p systems is presented in [16]. Containment queries exploit the structure of XML data (i.e. book contains author contains name = "John Smith"). XML elements and text words are treated uniformly as index keys. Local indexes at each peer consist of inverted lists, which map keywords to XML documents stored at the peer. In addition to its local inverted lists, each peer also maintains routing indexes, called peer inverted indexes, that map keywords to the identifiers of remote peers. A query is forwarded to remote peers by using the peer inverted index and set operations are used to minimize the number of relevant destinations. Indexes are built when a

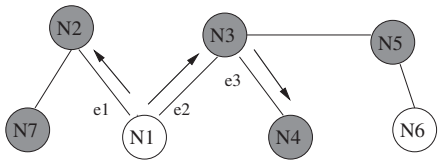
peer joins the system by exchanging information with other peers. These indexes are smaller than local indexes, since a peer only exchanges a small subset of its keywords, such as words that are often found in queries or that are representative of its local data. The result is a p2p system in which each peer has a summary of important data of all other peers. Horizons are used to limit the number of peers for which a peer has summarized information. A peer maps keywords outside of its horizon to peers on the boundary of the horizon that are closer to them.

In [26], each peer maintains a local index, summarizing its local content and one or more merged indexes summarizing the contents of its neighbors. The peers form hierarchies in which each peer stores summarized data for the peers belonging to its subtree. The roots are interconnected and store additional summaries for all other roots. Each peer that receives a query first checks its local index for any matches. Then, if it is an internal peer, it checks its merged index and if there is a match it forwards the query to its subtree. Furthermore, it sends the query to its parent or if it is a root peer to the other matching roots. The indexes used are based on Bloom filters that are compact data structures for the representation of a set of elements. To support the evaluation of regular XPath expressions, multi-level Bloom filters are introduced that preserve hierarchical relationships between the inserted elements. These relationships are preserved by inserting the elements of the XML tree to a different level of the filter according to their depth in the tree (Breadth Bloom filters), or by using paths of different lengths as keys and inserting them to the corresponding level of the filter according to their length (Depth Bloom filters).

A secure service discovery protocol for p2p systems is described in [12]. Service providers use the Service Discovery Service (SDS) to advertise descriptions and metadata of services expressed in XML. Clients use the SDS to locate the services they are interested in. The SDS servers are organized into hierarchies, which can be modified according to each server's workload. Each server is responsible for a particular domain; it receives advertisements and queries from a specific part of the network. When a server becomes overloaded, one or more child servers are spawned and assigned part of their parent domain. Each internal peer of the hierarchies stores summaries of the descriptions of its children, which are used for query routing. Summaries consist of a single Bloom filter. To insert a description in the filter, the description is divided to all possible subsets of the elements and attributes up to a certain threshold. Each query is split to all possible subsets and each one is checked in the index. The system emphasizes on security and access control issues and uses cryptography and authentication to ensure them.

4. CLUSTERING

Data clustering refers to grouping data items together to form clusters (groups) of items with common attributes or properties. In centralized systems, query performance may be improved by an appropriate placement of clustered data or indexes in main memory or in disk so that the I/O cost during query evaluation is minimized. In a distributed setting, clustering may improve query performance by reducing the communication cost through placing similar data at neighboring nodes. In a p2p setting, we further distinguish between (i) clustering similar data items (or indexes of similar data items) so that similar data (or indexes of similar data) are placed in neighboring peers and (ii) clustering peers with similar data items, so that their distance in the overlay network is small. By grouping similar peers, a query that reaches a peer in the cluster finds all other peers with relevant data nearby. Each form of clustering provides a different degree of storage autonomy. Data (or index) clustering violates storage autonomy, since it enforces peers to store specific items.



Peer	Edge	Keys	Keys (XML−based)
N1	e1	D3, ...	device/local/printer, device/camera, ...
	e2	D4, D5, D6, ...	device/camera, device/local/printer, device/network/printer ...
N3	e3	D5, D6, ...	device/local/printer, device/netwrok/printer, ...

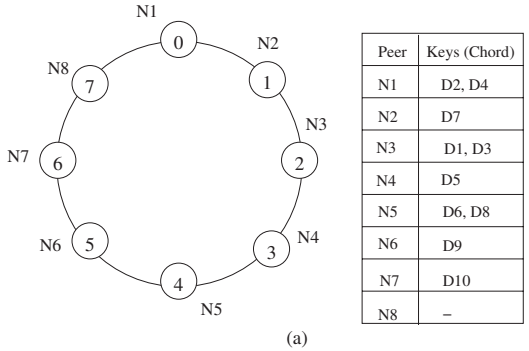
Figure 4: Indexes in unstructured p2p systems

In structured p2p systems, if the hash function is order-preserving, similar documents are stored at the same or neighboring peers. Order preserving hash functions are those hash functions that for similar inputs produce outputs close in the identifier space. Then, content-based clustering can be achieved by using as input to the hash function not just the name of the document but a semantic vector describing its content and structure.

Clustering the peers affects the topology of the p2p overlay. Issues of interest are how the peers within a cluster are interconnected (*intra-cluster organization*) and how the clusters are connected to each other (*inter-cluster organization*). Usually, query routing proceeds in two steps: first the appropriate cluster is identified and then the query is routed inside the cluster. Different mechanisms for intra-cluster and inter-cluster routing are often required.

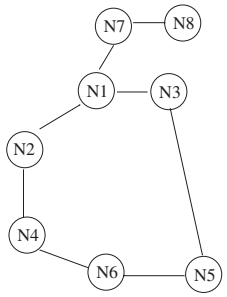
In hybrid clustered p2p systems, superpeers act as cluster representatives. Each cluster contains at least one superpeer, which is in charge of the management of the cluster: query processing and peer information management. The superpeers collect their peers' schemas and communicate with each other for query evaluation.

Figure 5 illustrates the two techniques for clustering in p2p systems, data or index clustering (Fig. 5(a)) and peer clustering (Fig. 5(b)). The documents stored at each peer are described in Fig. 2. In Fig. 5(a), the peers form a structured p2p system that follows a ring topology. We assume that the vectors extracted from each of the documents are given as input to an order preserving hash function. The output of the hash function maps each document to the virtual ring. The table shows which documents are mapped to each peer. Documents with the same or similar content are mapped to the same or neighboring peers. For example, documents $D2$ and $D4$ that have the same content are both mapped to logical node 0, while documents $D3$ and $D5$ that share some content (i.e., the path expression "device/local/printer") are mapped to neighboring peers. On the other hand, in Fig. 5(b) where peers clustering is performed what is affected is the overlay network. In particular, two peers that have similar content are connected in the overlay network. For example, peer $N3$ is connected to both



Peer	Keys (Chord)
N1	D2, D4
N2	D7
N3	D1, D3
N4	D5
N5	D6, D8
N6	D9
N7	D10
N8	–

(a)



(b)

Figure 5: (a) Data (index) clustering and (b) peer clustering

$N1$ and $N5$ with which it has similar content (document $D2$ of $N1$ has the same content with $D4$ of $N3$, while document $D7$ of $N5$ shares a common path expression with $D4$).

When compared to distributed clustering, the autonomy of the peers makes the application of distributed clustering in a p2p context challenging. Peers may join and leave the system or change their content very often. The clusters in the system must adapt to reflect the new setting without requiring the application of the clustering procedure from scratch. Thus, the allocation of similar data to neighboring peers must be dynamic and incremental. Furthermore, clustering algorithms for centralized applications rely on having global knowledge of the data or the schema that the data follow and use this information to define the clusters or the categories to group or classify the data. However, the changing environment and the lack of global knowledge in p2p systems prohibits the use of predefined clusters or categories and clustering must be applied as an adaptive distributed procedure among peers using only partial knowledge about the data and the system's topology. Moreover, using a fixed number of clusters creates unbalanced clusters that cannot cope with changes in the system workload. Thus, load-balancing should also be taken into account in the clustering procedure, since some clusters may become over-populated and assume most of the query workload, while others may be almost empty.

In most current research, with regards to peer clustering, the number or the description of the clusters is predefined and fixed while global knowledge of this information is required. With Semantic Overlay Networks (SONs), peers with semantically similar content are logically linked to form overlay networks based on a classification hierarchy of their documents, which is defined a priori [11]. Queries are processed by identifying which SONs are better suited to answer them. In [52], clusters of peers are again formed based on the semantic categories of their documents. Sophisticated procedures are proposed for both inter-cluster and intra-cluster load balancing. Similarly in [3], peers are partitioned into topic segments based on their documents. A fixed set of C clusters is assumed, each one corresponding to a topic segment. Knowledge of the C centroids is global.

To perform clustering in a p2p system that uses XML as the underlying model, we need to identify which peers have similar content or which XML documents are similar and therefore should be grouped together. Thus, we need to define appropriate similarity measures for XML documents. Clustering for XML documents in centralized applications mostly relies on structural information. For the classification of schema-less data, the authors of [51] combine text terms, structural information in the form of twigs and paths and also ontological knowledge (WordNet [14]) to construct more expressive feature spaces that are then used for the classification. XRules [55] assigns the documents to categories through a rule based classification approach that relates the presence of a particular structural pattern in an XML document to its likelihood of belonging to a particular category. S-GRACE [29] is a hierarchical algorithm for clustering XML documents with a distance metric based on the notion of a structure graph, which is a minimal summary of edge containment in the data. Finally, XClust [28] addresses clustering when schema information in the form of DTDs is available in contrast with the previous methods that can be applied to schema-less data. XClust clusters DTDs based on the semantics, immediate descendants and leaf-context similarity of DTD elements. These centralized methods for XML clustering cannot be directly applied to p2p systems since they require global knowledge of the data or of their schema. They need to be adapted to work with incomplete local knowledge acquired by the cooperation of the peers.

In [26], a form of peer clustering is applied to an unstructured p2p system of XML peers. The peers are organized into hierarchies according to their content similarity. Content similarity is derived from the similarity of their routing indexes. Similarity takes into account both the structure and the content of data and is efficiently calculated from the routing indexes without requiring any knowledge of the schemas or the data of the other peers. Upon entering the system, each peer sends its index to the roots of the hierarchies that compare it with their own indexes. The peer attaches to the most similar hierarchy, so that peers with similar content are organized into the same hierarchy. The number of the hierarchies, i.e. clusters, is not fixed and changes according to the content of the new peers that enter the system. An adaptation of this clustering procedure for non-hierachical architectures is presented in [25].

The systems we describe next, assume that schema information is available. In structured p2p systems this information can be used to provide for an appropriate mapping of data items or peers to the virtual space. If the schema is known a priori, the virtual address space can be split to sub-spaces each one corresponding to a different part of the global schema. Then, upon entering the system, each peer (or data item) can be mapped to the sub-space of the virtual space that corresponds to its schema, thus creating clusters of peers that follow the same schema.

Along this line, in [43], the system assumes that all peers share a common topic ontology, and organizes them into concept clusters that are described by a logical combination

of ontology concepts. These concept clusters are organized into a hypercube topology. A hypercube topology is followed within the concept clusters as well. In particular, a peer in an ontology-based hypercube carries an address, which concatenates a set of concept coordinates (outer hypercube) and a set of storage coordinates (inner hypercube). In [35], the assumption is that the data of the peers belong to some categorization hierarchies relevant to a domain, called a multihierarchic namespace. Each hierarchy is called a dimension. The coordinates of a data item in the system are expressed as n-tuples. Groups of peers choose what data to host based on their own interests, thus defining their interest areas. Interest areas are defined as subsets of the cross product of some dimensions and are divided into interest cells. The interest areas describe index coverage of other groups' data and are encoded into URNs. The associated index servers to each interest area are contacted to find relevant data.

The next three systems are hybrid clustered p2p systems that use global schema information to organize peers into clusters. In SQPeer [24], peers are grouped based on their RDF-schema similarity. The peers that hold RDF descriptions conforming to the same RDF schema are clustered together. The system uses active schemas that are fine-grained schema advertisements that contain only information about what is actually stored in a peer. In XPeer [42], peers are logically organized into clusters that are also formed on a schema-similarity basis, whenever this is possible. Superpeers are organized to form a tree, where each peer hosts schema information about its children; superpeers having the same parent form a group. In both SQPeer and XPeer, the procedure that creates the clusters is not described and the authors focus on the exploitation of the clusters when such clusters exist. The system in [30], is based on RDF schemas that are assumed to be globally known. Rule-based clustering is used. Peers are registered and grouped in clusters based on cluster specific rules that describe the properties that each peer in the cluster should possess. These rules are provided by the cluster's administrator.

5. REPLICATION

The goals of replication in a p2p system do not differ much from those in a non p2p distributed system. Replication is basically used to improve system performance and to increase data availability in case of peer failures. Performance can be improved with replication either through balancing the load among the peers or by increasing locality, i.e. placing copies of data closer to their requestors. Replication refers either to the data items or their indexes. Issues of interest in both p2p and non p2p distributed systems include determining which items to replicate, where to place the replicas and how to keep them consistent. However, replication in p2p systems creates new requirements. The lifetime autonomy of the peers makes replication essential for keeping data available even when some of the peers storing them go offline. On the other hand, this makes enforcing consistency very difficult in p2p systems. In traditional distributed systems, we have either *eager* or *lazy replication*. Eager replication keeps all replicas exactly synchronized at all nodes. Lazy replication propagates updates asynchronously. Most applications of p2p systems do not require strict consistency, thus only lazy replication is usually applied, because of the high cost of eager replication. Furthermore, the lack of any central administration and of global knowledge of both the data and the query workload make the decision of which items to replicate and where much more complicated. Handling these issues depends both on the overlay topology and the routing mechanism in use.

With regards to the number of replicas, there are various approaches. At one extreme, *uniform replication* creates the same number of replicas for all data items irrespectively of their popularity. At the other extreme, *proportional replication* creates for each data item a number of C replicas, where C is proportional to its popularity, i.e. the number of queries that concern the given item, assuming that the popularity, i.e. the query workload, is globally known. While in proportional replication, popular queries are satisfied very efficiently since a large number of copies for the requested data is available across the system, unpopular data require many search steps thus compromising the overall system performance. On the other hand, with uniform replication, a large portion of the system resources is wasted in replicating data that appear in queries very rarely. Between these two extremes, *square-root replication* [8] creates copies so that for any two data items the ratio of replication is the square root of the ratio of their query rates. Square-root replication provides better results by leveraging the efforts for finding popular and unpopular data items.

In unstructured p2p systems, two different strategies are mainly used for placing replicas: path and owner replication. With *owner replication*, whenever a peer issues a successful query about a specific data item, a replica of the item is created at that peer. With *path replication*, copies of the requested data are stored at all peers along the path in the overlay network from the requestor peer to the provider peer [31]. Although path replication outperforms owner replication, it tends to replicate data items to peers that are topologically along the same path, which somewhat hurts the system performance. For updating the copies, the authors in [41] propose an investment policy where each individual peer propagates an update only if it estimates a benefit from doing so, i.e., an investment return.

For structured p2p systems, the issue that is addressed by replication is mostly load balancing and data availability. Bringing the data items closer to the requestors is not an issue since these systems already require only a small number of search steps to locate each item. Distributed hashing that assigns data items to peers does not take into consideration the popularity of each item, thus some peers may be assigned with many popular items. Such peers are burdened with satisfying most of the query workload and become hot spots. CAN [38] tries to solve this problem by using multiple hash functions to assign each item to more than one peers or by creating multiple coordinate spaces. As far as data availability is concerned, if a peer in a structured p2p system fails, all the items assigned to it become inaccessible even if they are stored at other peers, since the routing protocol cannot operate correctly. For this reason, in Chord [45], each peer stores a list of k of its successor peers, so if its successor fails, it can contact the first available successor in its list. Finally, an additional requirement for structured p2p systems that map indexes of data items to peers and replicate items instead of indexes, is that the index entries must be extended to maintain the identifiers of all the peers that hold copies of the indexed item.

The problem of replicating XML data or indexes in p2p systems has not received much attention yet. An important

issue that arises is the granularity of replication and distribution for XML. The issue is discussed in [5] but for a non p2p distributed XML repository. There, a global conceptual schema is used by a simplified structure called RepositoryGuide, which is a tree-structured index that resembles a DataGuide [18]. The system supports XML-path and tree pattern queries. The fragmentation scheme decomposes the RepositoryGuide into a disjoint and complete set of tree-structured fragments that preserve data semantics. A sublanguage of XPath is used for data fragmentation that supports vertical fragmentation, which is solely based on the selection of node types through path properties. The language can also be extended to support horizontal fragmentation, which includes conditions and branching, although some consistency issues arise. The allocation phase consists of three steps: determining which fragments to allocate at which system nodes, placing schema structures at local nodes and suitable instances of fragments at each node. For the first step, existing methods from distributed databases are used. For the second step, the RepositoryGuide is fully distributed among the nodes. Finally, for the third step, the global context of each fragment is kept by storing the data path from the global root node to the root of the local fragment. To this end, three indexes are used: a path index that encodes the global context of local fragments, a term index that allows processing of queries that include conditions on terms and an address index that stores the physical addresses of the fragments. Space efficient path indexes are constructed with the use of a path identification scheme. Because of their small size, path indexes are replicated at all system nodes, while term and address indexes are distributed among them.

The replication of XML indexes in structured p2p systems for load balance is described in [15]. Since some objects may be more popular, thus creating a considerable load to the peers that store them, two methods are proposed for splitting this load with other peers, namely the split-replicate and the split-toss methods that split catalog information among peers. A peer increases the level of a popular index key, where level is the number of XML-tree path steps contained in the key (initially set to 1), and either replicates the new keys at the corresponding peers (according to the hash function) in the Chord ring (split-replicate), or only sends them there and then discards them (split-toss). The peer also creates a mapping in its catalog for the new locations of the key, which it hands over to peers that request it. Once they obtain these mappings, they query one of the new locations in a round-robin fashion.

In [2], in contrast with [15], the actual XML documents, called dynamic XML documents, and not their indexes are replicated or distributed in an unstructured p2p system. Dynamic XML documents are XML documents that contain materialized XML data that are part of the document and intentional data that can be produced by service calls. Since dynamic documents may contain calls to services on other peers, some form of distribution is inherently part of the model. External edges are added to the XML document to point to peers that store other parts of the documents to allow for a higher form of distribution. A cost model is introduced and used for query processing and for deciding whether to replicate some parts of other peers' documents to increase the efficiency of query execution.

6. QUERY PROCESSING

Query processing in traditional distributed systems can be divided into four phases: query decomposition, data localization, global and local query optimization. Firstly, a relational query is decomposed into an algebraic query on global relations using techniques from centralized databases. Secondly, data distribution information is exploited to localize the algebraic query. The global query optimization phase receives an algebraic query on data fragments and finds an optimal strategy for its execution taking into account selectivity estimations and communication costs. At the last phase, each node receiving a query subplan tries to optimize it locally using centralized algorithms. The first three phases are performed centrally with global knowledge, while the fourth one is executed on each participating node with the knowledge available locally. For scalability reasons and to preserve autonomy, query processing in p2p systems must be coordination free. Furthermore, the lifetime autonomy of peers suggests that the execution plan must be constructed dynamically, since the number of participating peers and the available data changes with time.

We consider two generic approaches to this issue. The first approach is based on evaluating the query *incrementally*. A peer initiates the execution of a query, and the query is routed among the peers using the indexes. The execution plan evolves by accumulating partial results evaluated at each peer and locating the next peer with relevant data that needs to process the query. The other approach resembles the traditional approach with the known four phases but localization of data is achieved by using the indexes placed at each peer. In particular, the initiator of a query transforms the query into a path list and sends this list to remote peers where with the use of index-joins (similar to semi-joins) between the list and the peers' path indexes, the peers that store data relevant to the query are retrieved. In a way, this method relies on *index shipping* rather than query or data shipping that is common in traditional distributed systems. If the system is a hybrid p2p system, the queries may be sent to the superpeers that are responsible for constructing the execution plan and coordinating query evaluation.

Figure 6 illustrates the different scenarios for the distributed processing of the query illustrated in Fig. 6(a) issued by peer $N1$. The query asks for all the printers in a building that have printing quality larger than 600x450 dpi. Let us assume that the information about the available printers in the building is stored in peer $N4$ and the information about various printers characteristics in peer $N2$. The query can then be decomposed in Q_1 that retrieves all the printers with the desired quality from $N2$ and Q_2 that retrieves all the available printers in the building from $N4$. The lines between nodes correspond to overlay network connections, while the arrows show the route the query takes. In Fig. 6(b), we illustrate the traditional client-server scenario where the query is forwarded to a central server that has the available information for decomposing, localizing and optimizing globally the query. The server constructs the distributed query plan and forwards the two sub-queries to the corresponding peers for local optimization and evaluation. The results are returned to $N1$ that performs the final join. In Fig. 6(c), the superpeer based scenario is illustrated. Peer $N1$ sends the query to the superpeer it is connected to, namely SP_1. SP_1 interacts with the other superpeers in the network (dotted arrows) and gathers the re-

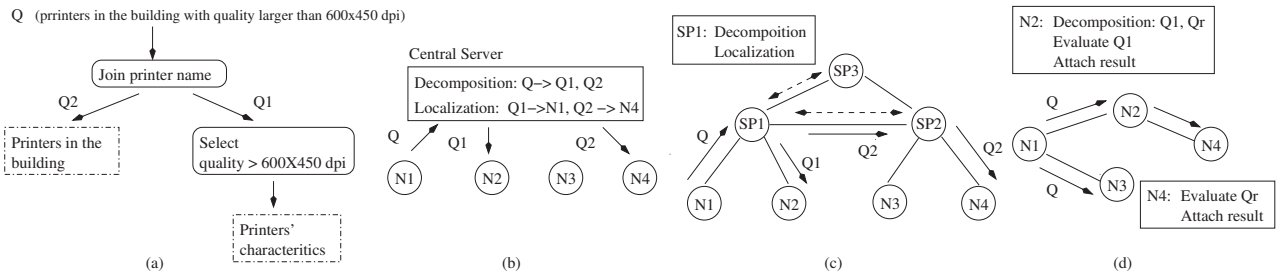


Figure 6: Distributed query processing scenarios

quired information to construct the distributed query plan. It then forwards the sub-query Q_1 to $N2$ that is attached to it, and Q_2 to SP_2 that in turn forwards the query to $N4$. The results are returned to SP_1 that performs the final join. Finally, in Fig. 6(d) we demonstrate the fully-distributed incremental query evaluation strategy. $N1$ issues the query and forwards it to its neighbors, following a flooding-based approach. When $N2$ receives the query, it realizes that it can evaluate a part of it (Q_1). It evaluates this part and attaches to the query plan the computed result. It then forwards the query further to its own neighbor $N4$. $N4$ evaluates the remaining part of the query and it either performs the final join itself or just attaches its own partial result and forward sit to $N1$ for the evaluation of the final join.

In the non p2p distributed XML repositories of [47], distribution is implemented by having links from the local XML data to XML objects at remote nodes. The data is represented by a rooted labeled graph. The model distinguishes between local links that point to local objects and cross-links that point to remote objects. Every node determines which of its data have incoming edges from other nodes (input data nodes) and which have outgoing edges to remote objects (output data nodes). Copies of the external data nodes are added to the node's graph. Given a query, an automaton is computed and sent to every node. Each node traverses only its local graph starting at every input data node and with all the states in the automaton. When the traversal reaches an output data node, it constructs a new output data node with the given state. Similarly, new input data nodes are also constructed. Once the result fragments, which consist of an accessibility graph that has the input and output data nodes and edges between them if and only if they are connected in the local fragment, are computed they are sent to the origin of the query. The client at the origin of the query assembles these fragments by adding missing cross-links, and computes all the data nodes accessible from the root. The algorithm requires only four communication steps and the size of the data exchanged depends only on the number of cross links and the size of the query answer.

Optimizing the cost of communication in answering XPath queries over distributed data based on the client-server model is considered in [48]. Minimal views that describe a query's results are used to avoid the redundancy met in such results where the same data may appear many times. The system leaves part of the evaluation of the query to the client that may have to extract all the answers from the minimal view to obtain the results to the initial queries.

Query processing in [5] is based on shipping index entries among nodes and efficiently evaluating chains of local joins of indexes. The node that issues a query determines the partitions of the query into path indexes lists that need to be sent to other nodes and instructs these nodes to compute the intermediate results. Then, the index results produced at the first node are sent to the second one to compute the join, and the resulting index is send to the third and so on. The final result is sent to the initial node where with the help of the RepositoryGuide the nodes that store the XML fragments defined by the resulting path index are retrieved. Similarly in the distributed RDF repository of [46], the main issue is to find the optimal ordering for a chain of joins of simple path expressions that are extracted from each query.

Mutant Query Plans (MQPs) [35] extend the weak query capabilities and limitations in index scalability and result quality of current p2p systems. A mutant query plan is an algebraic query plan graph, encoded in XML that may also include verbatim XML-encoded data, references to resource locations (URLs), and references to abstract resource names (URNs). Each MQP is tagged with a target, the peer that needs the results. An MQP starts as a regular query operator tree at the client peer and is then passed around among peers accumulating partial results, until it is fully evaluated into a constant piece of XML data. A peer can choose to mutate an MQP either by resolving a URN to one or more URLs, or a URL to its corresponding data. The peer can also reduce the MQP by evaluating a subgraph of the plan that contains only data at the leaves, and substituting the results in place of the subgraph. Resolving URLs is done either by connecting to the specified peer or by sending the MQP to it. For the URN the system's catalog is used. The cost model introduced in [2] for dynamic XML documents, is also used by a peer to decide on a query execution plan that minimizes its own observed cost. Each query is split to subqueries. A peer tries to satisfy locally as much of the query as it can and then forwards the remaining subqueries to the associated peers that follow the same procedure.

In [50], data is represented in XML and peers schemas in XML Schema. Query evaluation is incremental with an additional logical-level search where data are located based on schema-to-schema mappings. Instead of a local index, each peer maintains mappings between its own schema and the schemas of its immediate neighbors. Mappings are described as query expressions using a subset of XQuery. Peers are considered as connected through semantic paths of such mappings. Peers may store mappings, data or both. Query processing starts at the issuing peer and is reformulated over its immediate neighbors, which, in turn, reformulate the query over their immediate neighbors and so on. Whenever the reformulation reaches a peer that stores data, the appropriate query is posed on that peer, and additional results may be appended to the query result. Various optimizations are considered regarding the query reformulation process such as pruning semantic paths based on XML query containment, minimizing reformulations and precomputing some of the semantic paths.

Table 1: Issues and challenges in p2p management of XML

<table>
<tr><th></th><th></th><th>Structured P2P</th><th>Unstructured P2P</th><th>Non P2P</th></tr>
<tr><td rowspan="2">Indexing</td><td>Pure P2P</td><td>Mapping of paths onto the multi-dimensional virtual network [15], [6], [44], [49], [4], [53]</td><td>Use of space efficient routing path indexes through aggregation and encoding [26], [16], [27]</td><td rowspan="2">Less strict autonomy requirements, smaller scale</td></tr>
<tr><td>Hybrid P2P (superpeers)</td><td colspan="2">Superpeers responsible for routing hold all index information and propagate the queries to their peers
Different routing protocols and index structures between superpeers and between superpeers and peers [34], [12]</td></tr>
<tr><td rowspan="2">Clustering</td><td>Pure P2P</td><td>Clustered index
DHT-based clustering of peers based on their content by adopting the input of the hash function to split the multi-dimensional space into regions of peers with similar content [35], [43]
Non-DHT clustering based on schema similarity</td><td>Formation of peer clusters based on schema or content similarity, with the use of routing indexes
Different mechanism used for inter-cluster and intra-cluster routing [26]</td><td rowspan="2">Centralized techniques assume global knowledge</td></tr>
<tr><td>Hybrid P2P (superpeers)</td><td>Clustered index built only upon superpeers</td><td>Superpeers used as cluster representatives
Inter-cluster communication between superpeers [42], [24], [30]</td></tr>
<tr><td rowspan="2">Replication</td><td>Pure P2P</td><td rowspan="2">Data allocation with the use of distributed hashing
Replication requires additional mappings among peers responsible for the data (or indexes) and the peers that hold the replicas [15]</td><td rowspan="2">Replication increases data availability and performance [2]</td><td rowspan="2">Fragmentation-allocation of XML [5]</td></tr>
<tr><td>Hybrid P2P (superpeers)</td></tr>
<tr><td rowspan="2">Query Processing</td><td>Pure P2P</td><td colspan="2">Coordination-free based on the type of index
No clear distinction among query processing phases
Dynamic construction of the execution plan
Query passed around the peers accumulating partial results [35], [2], [50]</td><td rowspan="2">Distinct phases: query decomposition, localization, global and local optimization [47], [48]
Index shipping [5]
Ordering of joins [46]</td></tr>
<tr><td>Hybrid P2P (superpeers)</td><td colspan="2">Superpeers responsible for the construction and the coordination of the execution plan [42], [24]</td></tr>
</table>

In XPeer [42], peers export a tree-shaped DataGuide description of their data, which is automatically inferred by a tree search algorithm. The query language supported is the FLWR subset of XQuery without universally quantified predicates and sorting operations. Query compilation is performed in two phases by the superpeers. First, the peer that issues the query translates it into a location-free algebraic expression. Then, the query is sent to the superpeer network for the compilation of a location assignment. After the location assignment is completed, the query is passed back to the peer that issued it for execution to minimize the load of the superpeer network. The peer applies common algebraic rewriting and then starts the query execution: the query is split into single-location subqueries that are sent to the corresponding peers. Subqueries are locally optimized and the results are returned to the initial peer which executes operations such as joins involving multiple sources.

In SQPeer [24], queries are posed in RQL, according to the RDF schemas that are known to each peer. When a peer receives a query, it parses it and by obtaining the involved path creates the corresponding query pattern graph, which describes the schema information employed by the query. For each path pattern, the peers that contain the required data to answer it are discovered. If the schema of the peer is subsumed by the selected query path, then this query path is annotated with the name of the given peer. This is done for all known schemas that belong to remote peers. The result is an annotated query pattern graph with information for all peers that need to be contacted to answer the query. The query processing algorithm receives as input the annotated query graph and outputs the execution plan according to the underlying data distribution. Channels are used by the peers to exchange query plans and results.

7. CONCLUSIONS

In this paper, we present data management issues that arise in p2p systems that use XML as the underlying data model. Table 1 summarizes related issues and challenges in handling XML data. Regarding indexing, the main challenge is handling both value and path indexes. In structured p2p systems, the central problem is deriving appropriate mappings of documents to peers, while in unstructured p2p systems, the main issue is constructing space and update efficient routing indexes. Clustering in p2p refers to both (i) clustering index and data items and (ii) clustering peers with similar data so that their distance in the overlay network is small. Since most centralized XML clustering approaches assume global knowledge and relatively static environments, clustering in p2p systems poses new challenges. For replicating XML data in p2p systems, research is needed to cross-fertilize recent research results in (non XML) p2p replication and (non p2p) methods for XML fragmentation. Finally, processing XML queries in p2p systems is still an open issue. Techniques that handle the autonomy, decentralization, scale and dynamicity of p2p systems are required.

8. REFERENCES

[1] K. Aberer, P. Cudre-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: a Self-Organizing Structured P2P System. *SIGMOD Record*, 32(3), 2003.

[2] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic XML Documents with Distribution and Replication. In *SIGMOD*, 2003.

[3] M. Bawa, G. S. Manku, and P. Raghavan. SETS: Search Enhanced by Topic Segmentation. In *SIGIR*, 2003.

[4] A. Bonifati, U. Matrangolo, A. Cuzzocrea, and M. Jain. XPath Lookup Queries in P2P Networks. In *WIDM*, 2004.

[5] J. M. Bremer and M. Gertz. On Distributing XML Repositories. In *WebDB*, 2003.
[6] M. Cai and M. Frank. RDFPeers: A Scalable Distributed Repository based on a Structured Peer-to-Peer Network. In *WWW*, 2004.
[7] D. Chamberlin. XQuery: An XML query language. *IBM System Journal*, 41, 2003.
[8] E. Cohen and S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *SIGCOMM*, 2002.
[9] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A semantic Search Engine for XML. In *VLDB*, 2003.
[10] A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In *ICDCS*, 2002.
[11] A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, Computer Science Department, Stanford University, 2002.
[12] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An Architecture for a Secure Service Discovery Service. In *Mobicom*, 1999.
[13] N. Daswani, H. Garcia-Molina, and B. Yang. Open Problems in Data-Sharing Peer-to-Peer Systems. In *ICDT*, 2003.
[14] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.
[15] L. Galanis, Y. Wang, S. Jeffery, and D. DeWitt. Locating Data Sources in Large Distributed Systems. In *VLDB*, 2003.
[16] L. Galanis, Y. Wang, S. Jeffery, and D. DeWitt. Processing Queries in a Large Peer-to-Peer System. In *CAiSE*, 2003.
[17] Knowbuddy's gnutella faq. http://www.rixsoft.com/Knowbuddy/gnutellafaq.html.
[18] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *VLDB*, 1997.
[19] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What Can Databases Do for P2P? In *WebDB*, 2001.
[20] A. Y. Halevy, Z. G. Ives, and D. Suciu. Schema Mediation in Peer Data Management Systems. In *ICDE*, 2003.
[21] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Key-word Proximity Search on XML Graphs. In *ICDE*, 2003.
[22] A. Kementsietsidis, M. Arenas, and R. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *SIGMOD*, 2003.
[23] M. Klein. *Interpreting XML via an RDF Schema. Chapter in Knowledge Annotation for the Semantic Web*. IOS Press, Amsterdam, 2003.
[24] G. Kokkinidis. and V. Christophides. Semantic Query Routing and Processing in P2P Database Systems: ICS-FORTH SQPeer Middleware. In *EDBT Workshop on P2P and DB*, 2004.
[25] G. Koloniari, Y. Petrakis, and E. Pitoura. Content-Based Overlay Networks for XML Peers Based on Multi-level Bloom Filters. In *DBISP2P*, 2003.
[26] G. Koloniari and E. Pitoura. Content-Based Routing of Path Queries in Peer-to-Peer Systems. In *EDBT*, 2004.
[27] N. Koudas, M. Rabinovich, D. Srivastava, and T. Yu. Routing XML Queries. In *ICDE*, 2004.
[28] M. L. Lee, L. Yang, W. Hsu, and X. Yang. XClust: Clustering XML Schemas for Effective Integration. In *CIKM*, 2002.
[29] W. Lian, D. Cheung, N. Mamoulis, and S. Yiu. An Efficient and Scalable Algorithm for Clustering XML Documents by Structure. *IEEE TKDE*, 16(1), 2004.
[30] A. Loser, W. Siberski, M. Wolpers, and W. Nejdl. Information Integration in Schema-Based Peer-To-Peer Networks. In *CAiSE*, 2003.
[31] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *ICS*, 2002.
[32] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Technical report, HP Laboratories Palo Alto, TR HPL-2002-57, 2002.
[33] Napster. http://www.napster.com/.
[34] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Loser. Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-to-Peer Networks. In *WWW*, 2003.
[35] V. Papadimos, D. Maier, and K. Tufte. Distributed Query Processing and Catalogs for Peer-to-Peer Systems. In *CIDR*, 2003.
[36] N. Polyzotis and M. Garofalakis. Structure and Value Synopses for XML Data Graphs. In *VLDB*, 2002.
[37] M. Rabin. Fingerprinting by Random Polynomials. Technical report, CRCT TR-15-81, Harvard University, 1981.
[38] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM*, 2001.
[39] World-Wide Web Consortium: Resource Description Framework. http://www.w3.org/RDF.
[40] J. Risson and T. Moors. Survey of Research Towards Robust Peer-to-Peer Networks: Search Methods. Technical report, University of New South Wales, UNSW-EE-P2P-1-1, September 2004.
[41] M. Roussopoulos and M. Baker. CUP: Controlled Update Propagation in Peer-to-Peer Networks. In *USENIX*, 2003.
[42] C. Sartiani, P. Manghi, G. Ghelli, and G. Conforti. XPeer: A Self-organizing XML P2P Database System. In *EDBT Workshop on P2P and DB*, 2004.
[43] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services. In *P2P*, 2002.
[44] C. Schmidt and M. Parashar. Flexible Information Discovery in Decentralized Distributed Systems. In *HPDC*, 2003.
[45] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *SIGCOMM*, 2001.
[46] H. Stuckenschmidt, R. Vdovjak, G. Houben, and J. Broekstra. Index Structures and Algorithms for Querying Distributed RDF Repositories. In *WWW*, 2004.
[47] D. Suciu. Distributed Query Evaluation on Semistructured Data. *TODS*, 27(1), March 2002.
[48] K. Tajima and Y. Fukui. Answering XPath Queries over Networks by Sending Minimal Views. In *VLDB*, 2004.
[49] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. In *SIGCOMM*, 2003.
[50] I. Tatarinov and A. Halevy. Efficient Query Reformulation in Peer Data Management Systems. In *SIGMOD*, 2004.
[51] M. Theobald, R. Schenkel, and G. Weikum. Exploiting Structure, Annotation, and Ontological Knowledge for Automatic Classification of XML Data. In *WebDB*, 2003.
[52] P. Triantafillou, C. Xiruhaki, M. Koubarakis, and N. Ntarmos. Towards High Performance Peer-to-Peer Content and Resource Sharing Systems. In *CIDR*, 2003.
[53] Q. Wang and M. Oszu. A Data Locating Mechanism for Distributed XML Data over P2P Networks. Technical report, CS-2004-45, University of Waterloo, School of Computer Science, Waterloo, Canada, October 2004.
[54] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/REC-xml.
[55] M. J. Zaki and C. Aggarwal. XRules: An Effective Structural Classifier for XML Data. In *SIGKDD*, 2003.