Utilizing XML Clustering for Efficient XML Data Management on P2P Networks

Panagiotis Antonellis[1], Christos Makris[1] and Nikos Tsirakis[1]

[1] Computer Engineering and Informatics Department,
University of Patras, Rio 26500, Greece
{adonel, makri, tsirakis}@ceid.upatras.gr

Abstract. Peer-to-Peer (P2P) data integration combines the P2P infrastructure with traditional scheme-based data integration techniques. Some of the primary problems in this research area are the techniques to be used for querying, indexing and distributing documents among peers in a network especially when document files are in XML format. In order to handle this problem we describe an XML P2P system that efficiently distributes a set of clustered XML documents in a P2P network in order to speed-up user queries. The novelty of the proposed system lies in the efficient distribution of the XML documents and the construction of an appropriate virtual index on top of the network peers.

Keywords: XML, P2P, XML clustering, XML queries, XML management

1 Introduction

1.1 Background and related work

Since the emergence of file sharing applications, the P2P model has been increasingly popular along with the deployment in distributed directory service, storage and grid computing [18], [22]. The model refers to communications between similar processes running in different computers, or communication between devices that are equivalent with regard to how they exchange information and control communications. Among the main qualities that distinguish P2P networks, we recall dynamicity of data sources, robustness, scalability, reliability, no central administration, and no control over data placement.

In the context of P2P computing many methods have been proposed for data management. As far as concerning the content-based full-text search which is a challenging problem in Peer-to-Peer (P2P) systems, Tang et al. [31] developed a P2P information retrieval system called pSearch, in which document semantics are computed by latent semantic indexing in a vector space. The pSearch system can achieve performance comparable to centralized information retrieval systems by searching only a small number of nodes. Aberer et al. [1] proposed PGrid that builds a trie and clusters semantically similar data, thereby providing in-network indexing. Their algorithm is an efficient, completely decentralized approach which supports the fast, parallel construction of structured overlay networks. BATON [16], a balanced tree overlay structure which supports both exact queries and range queries efficiently. Each node of the tree is stored on exactly one peer, and each node has links to its parent, children, adjacent nodes, and selected neighbors at the same level. P-Ring [9] proposes a P2P range index for efficiently supporting equality and range queries. Viglas in [32] addresses the issue of building scalable distributed structures over peer-to-peer overlay networks. In this concept he proposes schemes to maintain these structures such as B+-trees and heap files in a DHT. P2P indexes have been proposed for multi-dimensional data in [22], [12]. In these approaches, the entire multi-dimensional space is partitioned and merged as peers join and leave the P2P system. Sartiani et al. [27] proposed a p2p XML data management system called XPeer. XPeer is currently being implemented on top of an existing persistent XML query engine. In this system more powerful peers take up extended responsibilities for a group of peers. Peers export a summary of their XML data in the form of a tree-shaped DataGuide [14]. XP2P [4] is a P2P framework for answering XPath queries which also builds on a DHT framework and allows peers to store whole or fragments of XML documents locally, whose path expressions are encoded by Rabin's fingerprinting method [26] and stored in the DHT. Garces et al. [13] describe techniques for indexing data stored in peer-to-peer DHT networks. Their system built a hierarchy of indexes using a DHT containing query-to-query mappings, such that a user can look up more specific queries for a given broader query, thereby refining his or her interests. Skobeltysn et al. [29] proposed a solution for the efficient support of structured queries, more specifically, XPath queries, in large-scale structured P2P systems based on the approach of the P-Grid structured overlay network. Abiteboul et al. [2] present KADOP which is a distributed infrastructure for warehousing XML resources in a P2P framework. This system allows a user to publish XML resources, search for them and declaratively built thematic portals. Some challenges arise in specific tasks, such as the need of an efficient query language that can handle the nature of these XML documents which may be incomplete, of different schemes and being distributed in a network. XQuery [7] is designed to provide a flexible and standardized way of searching through (semi-structured) data that is either physically stored as XML or virtualized as XML. Also the XKeyword [15] provides efficient keyword proximity queries on large XML graph databases and XSearch [8] is a semantic search engine for XML based also on keyword search. Structure based queries work effectively when data have the same structure but require from the user to know each time the scheme of the data in order to perform right queries. When data have different schemes then keyword queries are more suitable.

Another challenge is techniques for indexing XML documents for this type of applications. There are three basic types of P2P indexes. First the no index type [19] just floods data in the network for routing information but this results to a network's overload. This means that the peer where the query is formed contact with neighbor peers until there is a result. The second type is a centralized index [23] where the information for the data that peers handle, is being stored in a single peer. For example, if a peer enters the network, it has to send its data information to the central peer of the network in order to be aware for future queries. This type has the drawback of central index server bottleneck and can only be solved in a degree with the creation of copies of this single peer. Finally a distributed index can provide better results and it depends on the nature of the network. In structured networks each peer stores index information with a hash function, about the data that handles. Recent research such as [28], [31] extend structured P2P systems by exploiting the content of documents for determining the keys. In [11] the authors propose a distributed catalog framework based on Chord [30]. XP2P [4] also extends Chord for XML data while RDFPeers [5] are based on Multi-Attribute Addressable Network [6] which extends again Chord to answer multi-attribute and range queries. A DHT-based approach is presented in [33] while a non-DHT P2P architecture is presented in [24]. In unstructured networks there are used routing indexes such as those presented in [10]. These routing indexes of a peer store information about the data that neighbor peers have. In [21] the authors present two architectures for routing XML documents, while in [20] Koloniari and Pitoura present content-based routing of path queries using Bloom filters for indexing. Finally, the authors at [25] present a new system, called psiX that runs on top of an existing distributed hashing framework. PsiX supports efficient location of relevant XML documents into the network according to user-submitter XPATH queries by creating algebraic signatures of both XML documents and user queries.

1.2 Paper Motivation and Contribution

Most of the previous work on indexing and querying XML data over a P2P network is based on path decomposition. Complex user queries are decomposed in separate paths and those paths are looked-up over the network. The results of each look-up are then merged in order to identify the matching XML documents. However, this approach may lead in a vast increase of the number of hops required to identify relevant XML documents in the network. In addition, all of the previous works suppose random distribution of the XML documents among the network peers. Although this approach imposes no restrictions of what XML documents can be stored in every peer, it results on more complex indexing and querying algorithms.

Based on these notions, in our work we introduce an innovative scheme for storing and querying XML data over a P2P network, called *PeerXML*. The proposed scheme differentiates from the previous works in the sense that instead of assuming random distribution of XML documents along the network peers, it introduces specific rules of what XML documents can be stored in every network peer. More specifically, the proposed scheme is based on clustering of the stored XML documents and then efficiently distributing them along the network's peers. Each peer can store only XML documents belonging to the same cluster, thus ensuring a more homogeneous distribution of the XML documents along the P2P network. In addition, peers who are physically close to each other, store XML documents of the same cluster in order to reduce costly hops between distant peers.

Rather than storing a centralized index of the XML documents, *peerXML* builds a hierarchical index of the stored XML documents inspired by the VBI-tree which utilizes multi-level Bloom filters for reducing the size of the index. An indexed approach has been aloes used by psiX [25]. However, psiX requires an existing DHT network to work, while our index has no such restrictions. Finally, our querying algorithm allows a query to be processed holistically - even in the presence of '//', thus eliminating additional hops when searching for matching XML documents.

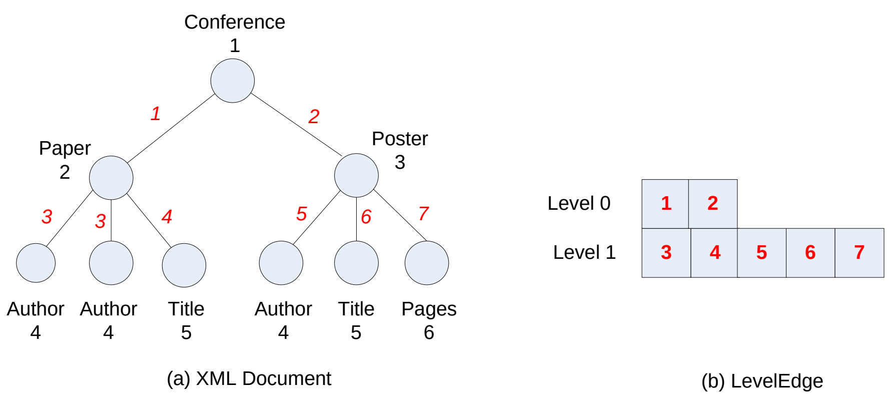

Fig. 1 Example of a LevelEdge representation

The contribution of this work can be summarized as follows:

- Clustering of the XML documents using the XEdge clustering algorithm.
- Distribution of the XML documents in the network's peers based on the belonging cluster.
- Multi-level indexing of the network's peers inspired by the VBI-tree [17].
- Utilization of multi-level Bloom filters for quickly testing if a query may match with a set of XML documents.
- Efficient routing and processing of the incoming user queries.
- Efficient handling of deletion/insertion/alteration of stored XML documents.

To our knowledge, this is the first work that utilizes XML clustering for efficiently distributing XML documents into the P2P network peers.

2 LevelEdge Representation and XEdge Clustering Algorithm

Our system utilizes the LevelEdge representation and XEdge clustering algorithm introduced in [3] in order to efficiently cluster the stored XML documents, before distributing them to the underlying P2P network. Below, we briefly describe LevelEdge and XEdge along with their use in our system.

The LevelEdge representation groups the distinct edges for each level in the XML document. It is organized as a vector of levels, where each level contains a list of distinct edges. Each distinct edge is uniquely defined by its two distinct point-nodes. The distinct edges are first encoded as integers and those integers are used in order to construct the LevelEdge representation of an XML document. Figure 1(b) presents the LevelEdge representation of the XML document in Figure 1(a). The integer numbers in the side of each edge in the XML document are the encodings of the corresponding edges. For example, all the Paper-Author edges are encoded as 3, while the Poster-Author edge is encoded as 5.

Although the LevelEdge cannot be used for fully reconstructing the original XML document, it is compact enough and it can be used for quickly answering if a query doesn't exist in a given XML document. If the query's structure doesn't match with the LevelEdge summarized information, it is certain that the query is not contained in the underlying XML document. However, depending on the structure of the query and the underlying document, the LevelEdge may provide a false positive answer. Although this case may lead to an overhead in our system, the system's accuracy is not affected because every positive answer leads to a full query checking against the corresponding XML document, thus a false positive answer will be rejected after fully checking the query against the underlying XML document.

Our system utilizes the structure of the LevelEdge representation to construct multi-level Bloom filters for quickly checking if a query is contained in the stored XML documents of a peer, as we describe in a later section.

The XEdge clustering algorithm is a modified version of k-Means where each XML document is represented by its LevelEdge and which utilizes the previously described distance metric in order to calculate the distance between two LevelEdge representations. In addition, for every cluster we define its cluster representative. A cluster representative is a LevelEdge representation that summarizes all the LevelEdge representations of the XML documents belonging to the corresponding cluster. More precisely, each level of the cluster representative contains all the distinct edges in that level of all the cluster's LevelEdge representations.

XEdge consists of the initialization phase and the main phase. In the initialization phase, k clusters are formed and the initial centroid for each cluster is calculated. During the main phase, every LevelEdge representation is checked again each cluster and is assigned to the closest cluster. The distance between a LevelEdge representation   and a cluster   is defined as the distance between   and the cluster's representative. After assigning all the LevelEdge representations, the cluster representatives are recalculated. The main phase is repeated until no cluster representative is changed.

Our system utilizes XEdge for clustering the XML documents before distributing them to the underlying P2P network and the cluster representatives for quickly checking if a user query may have an answer in the XML documents belonging to a specific cluster. Thus, we check the query against only documents that belong to matched clusters, as described later.

3 Multi-Level Bloom Filters

In order for quickly checking if a twig query is contained in a set of XML documents, we propose an extension of Bloom filters based on the LevelEdge representation. The LevelEdge Bloom filter (LBF) for a LevelEdge representation L with n levels is a set of n Bloom filters {LBF_0, LBF_1, … LBF_{n-1}}. In each LBF_i Bloom filter we insert all the edges appearing in the *i*-th level of L. If the original LevelEdge representation summarizes the structure of only one XML document, then the corresponding LBF

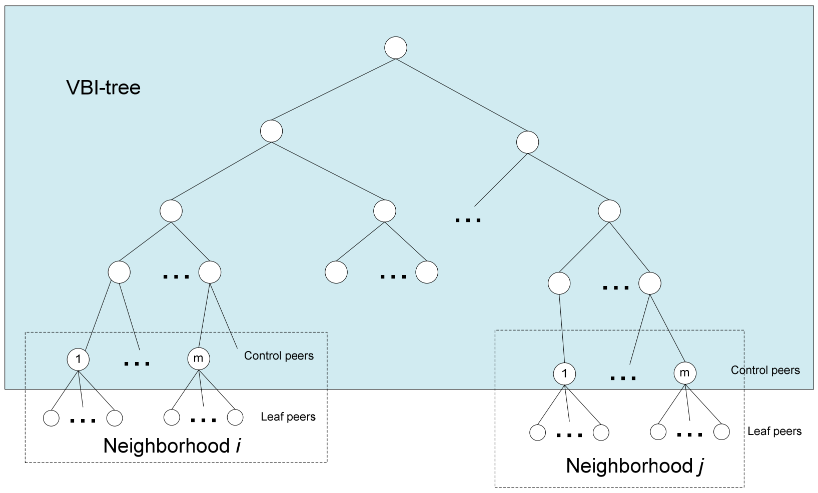

Fig. 2 Utilized indexing scheme

can be utilized for checking if a twig query exists in the underlying XML document. On the other hand, if the original LevelEdge representation summarizes a set of XML documents, then the LBF can be utilized for checking the existence of a twig query in any document of the underlying set.

The filtering of a twig query q through an LBF LF is the process of checking whether there is possibility that q is contained in any of the documents that are summarized by LF. If the filtering process returns a positive result, then we fully process the corresponding query against every document summarized by the underlying LevelEdge representation of LF. Our model is orthogonal to any method of twig query processing against a set of XML documents.

In order to filter a twig query against a LBF we first query the edges at the first level of the query with every level of the LBF. If we find some level(s) of the LBF that contain all the edges of the first level of the query we continue the filtering process, otherwise we abort it. For every such matched level of LBF we apply the following steps: We proceed to the next level of the query and check the current edges with the next level of the LBF. If the edges are contained in the LBF we repeat until we reach the end of the query or until the edges are not contained in the LBF. For edges with the ancestor/descendant axis '//', the query is split at the //, and the sub-queries are processed at all the appropriate levels. All matches are stored and compared to determine whether there is a match for the whole query. If the query was matched against the LBF, then there is big possibility that the query is contained in some of the underlying XML documents; otherwise we are certain that the query is not contained in any of the underlying XML documents.

4 Document Distribution and Index Construction

The clustering of the initial set of XML documents is performed once in a central server, before initializing our system. The utilized clustering algorithm is the XEdge algorithm described in Section 2. The number of formed clusters may vary and depends mainly on how many different "topics" of XML documents are included in our collection. From now on we make the assumption that the clustering algorithm has formed k distinct clusters.

For every formed cluster, the XEdge algorithm creates a corresponding LevelEdge representation, called cluster representative as described in Section 2. Our system creates for every cluster representative an LBF as described in Section 3. Those LBFs are used for quickly testing if a query is contained in the XML documents of a specific cluster, thus avoiding routing the query in peers that we are certain they don't contain XML documents matching with the query. This is a main advantage of utilizing the formed clusters for minimizing the hops during query routing and processing. From now on, we will refer to the LBFs for the cluster representatives as $CLBF_1$ … $CLBF_k$, where $CLBF_i$ is the LBF for the *i*-th cluster representative. As we describe later, every peer of the network has a copy of all the CLBFs in order to speed up the query routing and processing phase and minimize the total hops.

Our indexing scheme is inspired by the VBI-tree framework [17] which introduces a P2P framework for multidimensional indexing schemes. Below we describe in details the proposed scheme:

The underlying P2P network is divided into *k neighborhoods*, with each neighborhood storing and managing the XML documents of a single cluster. Each neighborhood consists of physical neighbor peers, in order to minimize the total number of required hops during query routing and processing. Thus, neighbor peers store similar XML documents, while distant peers store different XML documents. Every neighborhood is organized in a two-level hierarchy to help optimize the query routing as well as the insertion/deletion/update of the stored XML documents. The low-level peers of a neighborhood N_i are called *leaf peers* and are used for actually storing the XML documents of the *i*-th cluster. Each leaf peer stores d_i / n_i documents, where d_i is the total number of XML documents belonging to the *i*-th cluster and n_i is the total number of leaf peers in N_i. The top-level peers of N_i are called *control peers* and are used for query routing through the current neighborhood as well as through different neighborhoods in the network. Every control peer is responsible for a subset of the leaf peers in N_i, called its leaf *subset peers*. The total number of control peers is much smaller than the total number of leaf peers, but not too small otherwise they will become the bottleneck of the query processing procedure described later. The control peers of each neighborhood know all their leaf subset peers and can redirect any query to all of them. On the other hand, the leaf peers know only their control peer as well all their sibling leaf peers, which are the peers belonging to the same control peer.

All the control peers of our network are organized in a multi-level indexing scheme inspired by VBI-tree. Thus, every control peer is assigned a pair of VBI-Tree nodes: a routing node and a data node, in which the data node is the left adjacent node of the routing node (in the in-order traversal of the tree). We utilize this scheme for efficient and balanced query routing through the different control peers in our network. Each routing node of the indexing scheme maintains links to its parent, its children, its adjacent nodes and its sideways routing tables as in VBI-tree. In addition every routing node stores a LevelEdge structure that summarizes the XML documents of all its children data nodes (e.g. control peers) and a corresponding LBF. This LBF is used for checking the cover area of each routing node in the VBI-tree. Thus the root node stores an LBF that covers all the XML documents stored in the network.

A query is said to match with the cover area of a routing node if and only if it matches with the corresponding node's LBF. The routing algorithm described in the original VBI-tree utilizes this cover-area check in order to identify in which nodes the query should be forwarded.

In order for the query processing to be efficient, every control and leaf peer initializes and utilizes some extra structures, described below:

Each leaf peer creates an LBF for all its XML documents, called *Local LBF*, which is used for quickly determining if a query is likely to match with any stored XML document in that peer. On the other hand, each control point stores the cluster representative LBF for the neighborhood's cluster along with an LBF for all the XML documents that are stored in its leaf subset peers.

5 Query Routing and Processing

The query processing algorithm of our system utilizes the indexing structure to efficiently forward the user queries to the appropriate control peers of the neighborhoods that are possible to match the query. The set of XML documents is considered as the total search space and each data node (control peer) of the indexing scheme contains a region of the search space that corresponds to the XML documents belonging to its leaf subset peers. A query intersects with a region of the search space if and only if it matches with any of the XML documents belonging to that region. The indexing scheme in order to apply the VBI-tree range query algorithm for forwarding the query to the appropriate control peers should also be able to check if two regions (sets of XML documents) intersect with each other. This can be easily done by checking the LevelEdge structures of the corresponding routing nodes. If the two LevelEdges contain at least one common edge in any of their levels, then the corresponding regions (sets of XML documents) intersect with each other.

When a query is submitted to a peer p_j of the network, the peer p_j is automatically responsible for processing and answering the submitted query. p_j checks at first its LBF to see if the query is likely to be contained in its XML documents. If so, it performs a full query processing against all its stored XML documents using an XML search algorithm and stores the results in its cache. Then, it forwards the query to its parent control peer for further routing.

Algorithm 1: QueryProcessing(node n, query q)

```
if (n is leaf_peer) then
    if (q matches LBF of n)
        perform full match against the documents of n
    end if
    QueryProcessing(parent(n), q) /*Forward the query to the parent of n*/
else  /*n is a control peer*/
    if (q matches LBF of n)
        /*Forward the query to the leaf subset peers of the control peer*/
        for (each leaf $l_n$ in leaf subset peers of n) do
            QueryProcessing($l_n$, q)
        end
    end if

    if (q matches CLBF of n)
        /*Forward the query to the all control peers of the neighborhood*/
        for (each control peer $c_p$ of the neighborhood(n)) do
            QueryProcessing($c_p$, q)
        end
    end if

    /*Now use the VBI-tree routing algorithm to forward the query into*/
    /*appropriate peers*/
    VBI_query_process(n, q)
end if
```

When a query reaches a control peer c_j, it first checks its LBF to see if the query is likely to be contained in the XML documents stored in its leaf subset peers. If so, it forwards it to all its leaf subset peers. Next, it checks the cluster representative LBF to see if the query is likely to be contained in the XML documents of the neighborhood's cluster. If so, it forwards the query to the rest control peers of the neighborhood. Those peers will check their LBFs and if the query matches, they will forward the query to their leaf subset peers. Finally, the control peer uses the VBI-tree to forward the query to any other control peer which its LBF contain the query. The query routing is done as proposed in the original VBI-tree framework [17], with the difference that the routing nodes check their LBFs to decide where to forward the query.

Every peer that matches any of its XML documents with the query propagates the results back to the original peer, because this peer is responsible for gathering the total results. The peer can utilize any ranking or top-k algorithm before displaying the results to the end-user. From the previously described query routing process, it is clear that the query is propagated only to the appropriate cluster neighborhood leaf peers, thus reducing the total hops and checks.

6 Updates Handling

The proposed P2P scheme has the advantage of efficiently handling insertion/deletion/alteration of the underlying XML documents. Any of those updates is handled locally in a single neighborhood and includes updates only in the control peers of the corresponding neighborhood and in a single leaf peer. Thus, the overhead is always constant and relatively small as it only affects a very small number of peers in the total P2P network. Below we describe the handling of each supported operation:

Insertion/Alteration. When a new XML document is inserted or altered in a leaf peer, the peer itself updates its LBF in order to include the newly added/altered XML document. In addition, it forwards the XML document to its parent control peer, which in turn updates its LBF as well as the CLBF of the corresponding cluster. At next, it forwards the altered CLBF to the rest of control peers in the current neighborhood. Finally, the VBI-tree nodes are updated accordingly in order to reflect the changes in the data nodes (control peers). It is important to note that every altered VBI-tree node also updates its stored LBF and LevelStructure in order to reflect the changes. Thus, the change is propagated only on nodes belonging to the path from the root to the affected leaf peer.

Deletion. When an XML document is deleted from a leaf peer, the peer itself updates its LBF in order to exclude the removed XML document. In addition, it forwards the XML document to its parent control peer, which in turn updates its LBF as well as the CLBF of the corresponding cluster. At next, it forwards the updated CLBF to the rest of control peers in the current neighborhood. Finally, the VBI-tree nodes are updated accordingly in order to reflect the changes in the data nodes (control peers). Again the change is propagated only on nodes belonging to the path from the root to the affected leaf peer.

The insertion/removal of a node from the network is handled accordingly to the original VBI tree and is beyond the scope of our work.

7 Experimental Study

We have built a prototype P2P emulator to evaluate the performance of our proposed indexing system over large-scale networks. The prototype was implemented in Java 6 and the experiments were performed in a machine with 2.2 Core2Duo processor and 2 GB of RAM.

In order to evaluate the performance of the proposed system, we performed two different experiments and we counted the average number of hops required for each query in order to reach the appropriate peers in the network. In each experiment we utilized the Niagara XML dataset along with additionally created synthetic XML documents to form a dataset of about 2000 XML documents. Those documents were first clustered and then distributed in the P2P network as described. In addition we created a varied set of user queries, with each query matching either with some XML documents of a specific cluster or with none cluster.

Table 1. Results of first experiment

#Peers	#Total hops	#Neighbor hops	#Routing hops	Percentage (#Total hops / #peers)
100	11	5	6	11%
200	19	15	4	9.5%
500	57	51	6	11.4%
1000	108	97	11	10.8%

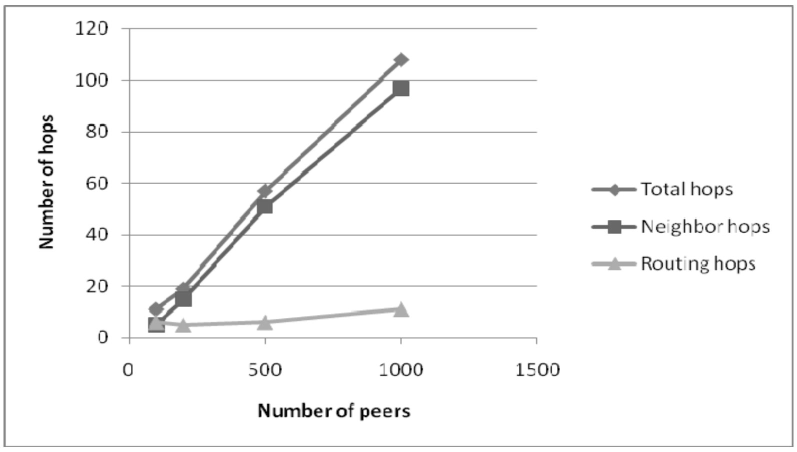

Fig. 3 Number of hops in relation to the number of peers

For each query, which was propagated in a random leaf peer, we counted the number of hops required in order to reach the appropriate leaf peers which contain the XML documents that match with it.

7.1 Varying number of peers

In this experiment, we wanted to study the relationship between the number of peers in the network and the number of hops required for each query to be processed. Thus, we created 8 clusters of totally 2000 XML documents which were distributed in 100, 200, 500 and 1000 number of peers in the network. For each case we counted the average number of hops for each query in the query set. The experimental results are shown in Table 1 and Figure 3.

As we can observe, the average number of hops required for each query to be processed is increasingly analogously to the number of peers in the P2P network. This was an expected result because as the number of peers increases, the number of peers that contain XML documents which match with the query increases, so the query needs to be propagated to more nodes. In addition, due to the fact that the XML documents belonging to each utilized cluster were very similar to each other, each query related to a specific cluster was propagated to all the leaf peers of the corresponding network neighborhood. This leads to the identified relationship between the number of hops and the number of peers in the P2P network.

Table 2. Results of second experiment

#Clusters	#Total hops	#Neighbor hops	#Routing hops
4	145	141	4
8	108	97	11
16	107	91	16

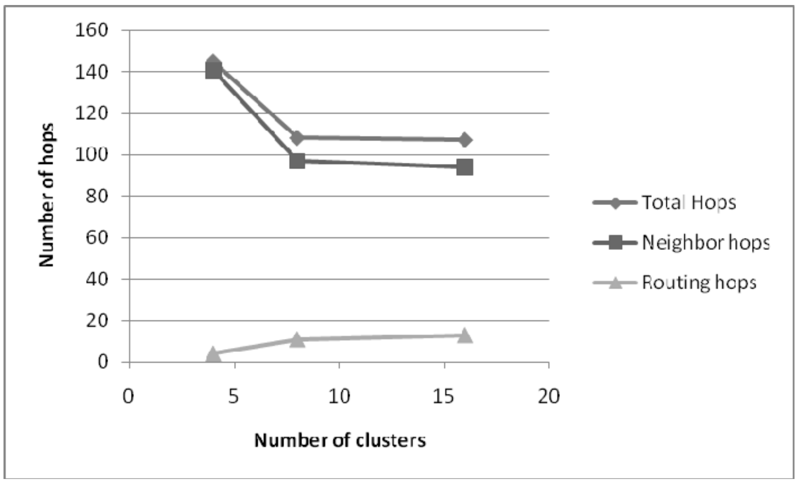

Fig. 4 Number of hops in relation to the number of clusters

However, from Table 1 it is clear that the average number of peers engaged in the processing of a single query is about 10% of the total peers in all cases. This means that no matter how large is the P2P network, each single query will reach only 10% of the peers, thus the traffic is reasonably small in all cases. However, the most important result of that experiment is that most of the required hops per query (about 90% in most cases) are between leaf peers in the same neighborhood of the P2P network and only about 10% are hops between routing peers. For example, in the case of 1000 peers, only 11 hops are between routing nodes, while the rest 97 are hops between leaf peers in the same neighborhood. This means that each query requires very little hops in order to reach the matched appropriate network neighborhood which is related to it. After reaching it, it is being propagated to all the neighborhood's peers as described in Section 5.

Based on the assumption that peers belonging to the same network's neighborhood are physically close to each other, the cost of the neighbor hops is much less than the cost of the routing hops. Thus the proposed scheme achieves to process user queries with a very small number of routing hops (about 1% of the total number of peers in the P2P network).

7.2 Varying number of clusters

In this experiment, we wanted to study the relationship between the number of formed clusters (neighborhoods) in the network and the number of hops required for each query to be processed. Thus, we emulated a P2P network of 1000 peers and formed 4, 8 and 16 clusters of totally 2000 XML documents. For each case we counted the average number of hops for each query in the query set. The experimental results are shown in Table 2 and Figure 4.

As we can observe, the average number of hops required for each query to be processed is at first decreasing as the number of clusters increases and then converges to a constant value. This was an expected result because as the number of clusters increases, the number of control peers and the size of the VBI tree index increases but the size of each neighborhood is decreasing, so the query requires more routing hops but much less neighbor hops. Thus, the total number of hops is decreasing between 4 and 8 clusters but remains about the same between 8 and 16 clusters. The later result comes from the fact that in the case of 16 clusters some clusters were very similar to each other, so some queries matched with more than one cluster. This prevented the number of hops of decreasing as expected. If the clusters were totally distinct to each other, then the number of hops would decrease again between 8 and 16 clusters. In addition, this experiment confirms the observation that the number of routing hops is about 10% of the total number of hops required per query. The rest of them are hops between leaf peers of the same network's neighborhood.

Both experiments showed that each query requires a very small number of hops (~1% of peers) between routing nodes of different neighborhoods. The rest of the required hops are between nodes in the same neighborhood and are necessary, because a query that matches with a cluster's signature (LBF) may match any document belonging to that cluster. Thus, the query should be forwarded to every node in the corresponding network neighborhood in order to acquire all the possible matches. However, between nodes in the same neighborhood that are physically located close to each other, the cost of those hops is small relatively with the cost of hops between nodes in different neighborhoods. Thus, the main contribution of the proposed scheme is that it eliminates the costly hops between different neighborhoods, thus reducing the total processing time of each query.

8 Conclusions and Future Work

In this work we have presented a novel scheme for storing and querying XML data over a P2P network. The proposed scheme is based on clustering of the stored XML documents for efficiently distributing them along the network's peers. The proposed scheme utilizes the XEdge clustering algorithm for clustering the XML documents and then distributes the documents of the same cluster into peers belonging to the same network neighborhood. This distribution is based on the assumption that peers belonging to the same network neighborhood are physically close to each other, thus we are able to eliminate messages between peers belonging to different network neighborhoods. This is achieved by implementing an efficient index structure on top of the network's neighborhoods, inspired by the VBI-tree index and by efficiently propagated only to the neighborhoods that match with. The experimental results showed that the total number of hops required per query is about 10% of the total number of peers, but only 10% of them are hops between peers in different neighborhoods.

As future work, we intend to improve our P2P network simulator in order to perform more detailed experiments and integrate into it other approaches too; moreover we aim to extend our routing process in order to efficiently take into consideration not only the query structure but the query value predicates as well.

9 Acknowledgements

Panagiotis Antonellis' work was supported in part by the Hellenic State Scholarships Foundation (IKY).

References

1. Aberer, K. Datta, A., Hauswirth, M. and Schmidt, R. "Indexing Dataoriented Overlay Networks," in Proc. of the 31st VLDB Conference, Trondheim, Norway, 2005, pp. 685–696.
2. Abiteboul, S., Manolescu, I., Preda, N. (2005) Constructing and Querying Peer-to-Peer Warehouses of XML Resources. ICDE: 1122-1123
3. Antonellis, P., Makris, C. and Tsirakis, N. (2008) XEdge: Clustering Homogeneous and Heterogeneous XML Documents Using Edge Summaries. 23rd Annual ACM Symposium on Applied Computing, Fortalezza, Brazil
4. Bonifati, A., Matrangolo, U., Cuzzocrea, A. and Jain, M. "XPath Lookup Queries in P2P Networks," in the 6th annual ACM Intl. Workshop on Web Information and Data Management (WIDM'04), Washington, DC, Nov. 2004, pp. 48–55.
5. Cai, M. and Frank, M. (2004) RDFPeers: A Scalable Distributed Repository based on a Structured Peer-to-Peer Network. In WWW.
6. Cai, M., Frank, M., Chen, J., Szekely, P. (2004) MAAN: A Multi-Attribute Addressable Network for Grid Information Services. J. Grid Comput. 2(1): 3-14.
7. Chamberlin, D. (2003) XQuery: An XML query language. IBM System Journal, 41
8. Cohen, S., Mamou, J., Kanza, Y. and Sagiv, Y. (2003) XSEarch: A semantic Search Engine for XML. In VLDB.
9. Crainiceanu, A., Linga, P., Machanavajjhala, A., Gehrke, J. and Shanmugasundaram, J. "P-Ring: An Efficient and Robust P2P Range Index Structure," in Proc. of the 2007 ACM-SIGMOD Conference, Beijing, China, 2007, pp. 223–234
10. Crespo A., and Garcia-Molina, H. (2002) Routing Indices for Peer-to-Peer Systems. In ICDCS.
11. Galanis, L., Wang, Y., Jeffrey, S. and DeWitt, D. (2003) Locating Data Sources in Large Distributed Systems. In VLDB.
12. Ganesan, P., Yang, B. and Garcia-Molina, H. "One Torus to Rule them All: Multi-dimensional Queries in P2P Systems," in Seventh Intl. Workshop on the Web and Databases, Paris, France, Jun. 2004.

13. Garces-Erice, L., Felber, P. A., Biersack, E. W., Urvoy-Keller, G. and Ross, K. W. “Data Indexing in Peer-to-peer DHT Networks,” in Proc. of the 24th IEEE Intl. Conference on Distributed Computing Systems, Tokyo, Mar. 2004, pp. 200–208
14. Goldman, R. and Widom, J. “DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases,” in Proc. of the 23rd VLDB Conference, Athens, Greece, Aug. 1997, pp. 436–445.
15. Hristidis, V., Papakonstantinou, Y., and Balmin, A. (2003) Keyword Proximity Search on XML Graphs. In ICDE
16. H. Jagadish, B. C. Ooi, and Q. H. Vu, “BATON: A Balanced Tree Structure for Peer-to-Peer Networks,” in Proc. of the 31st VLDB Conference, Trondheim, Norway, 2005
17. Jagadish, H. V, Ooi, B.C, Vu, Q.H, Zhang, R. and Zhou, A. (2006) VBI-Tree: a Peer-to-Peer Framework for Supporting Multi-Dimensional Indexing Schemes. ICDE
18. Jiang, H. and Jin, S. (2005) Exploiting Dynamic Querying like Flooding Techniques for Unstructured Peer-to-peer Networks. In Proceedings of IEEE ICNP.
19. Knowbuddy's Gnutella faq. 2009. [Online] Available at: http://www.rixsoft.com/Knowbuddy/gnutellafaq.html [Accessed 10 January 2009].
20. Koloniari, G. and Pitoura, E. (2004) Content-Based Routing of Path Queries in Peer-to-Peer Systems. In EDBT.
21. Koudas, N., Rabinovich, M., Srivastava, D. and Yu, T. (2004) Routing XML Queries. In ICDE.
22. Liu, B., Lee, W.C. and Lee, D. L. “Supporting Complex Multi-Dimensional Queries in P2P Systems,” in Proc. of the 25th IEEE Intl. Conference on Distributed Computing Systems, Columbus, OH, Jun. 2005, pp. 155–164
23. Napster. 2009. [Online] Available at: http://www.napster.com [Accessed 10 January 2009].
24. Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I. and Loser, A. (2003) Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-to-Peer Networks. In WWW.
25. Praveen R. Rao, Bongki Moon (2009) Locating XML Documents in a Peer-to-Peer Network using Distributed Hash Tables. IEEE Transactions on Knowledge and Data Engineering, 08 Jan. 2009.
26. Rabin, M. O. “Fingerprinting by Random Polynomials,” Harvard University, Cambridge, MA 02138, Tech. Rep. TR 15-81, 1981
27. Sartiani, C., Manghi, P., Ghelli, G. and Conforti, G. “XPeer: A Self-Organizing XML P2P Database System,” in Intl. Workshop on Peer-to-Peer Computing and Databases, Greece, 2004
28. Schmidt, C. and Parashar, M. (2003) Flexible Information Discovery in Decentralized Distributed Systems. In HPDC.
29. Skobeltsyn, G., Hauswirth, M. and Aberer, K. “Efficient Processing of XPath Queries with Structured Overlay Networks,” in The 4th Intl. Conference on Ontologies, DataBases, and Applications of Semantics, Aiga Napa, Cyprus, Oct. 2005
30. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrish-nan, H. (2001) Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In SIGCOMM.
31. Tang, C., Xu, Z. and Dwarkadas, S. “Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks,” in Proc. of the 2003 ACM-SIGCOMM Conference, Germany, Aug. 2003, pp. 175–186.
32. Viglas, S. “Distributed File Structures in a Peer-to-Peer Environment,” in Proc. of the 23th IEEE Intl. Conference on Data Engineering, Cancun, Mexico, 2007, pp. 406–415
33. Wang, Q. and Oszu, M. (2004) 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.