XPath Lookup Queries in P2P Networks *

Angela Bonifati
Icar CNR
Via Pietro Bucci, 41C
Rende, Italy
bonifati@icar.cnr.it

Ugo Matrangolo
DEIS, University of Calabria
Via Pietro Bucci, 41C
Rende, Italy
matrangolo@si.deis.unical.it

Alfredo Cuzzocrea
DEIS, University of Calabria
Via Pietro Bucci, 41C
Rende, Italy
cuzzocrea@si.deis.unical.it

Mayank Jain†
Indian Institute of Technology
IIT Delhi
New Delhi, India
mayankjain@cse.iitd.ernet.in

ABSTRACT

We address the problem of querying XML data over a P2P network. In P2P networks, the allowed kinds of queries are usually exact-match queries over file names. We discuss the extensions needed to deal with XML data and XPath queries. A single peer can hold a whole document or a partial/complete fragment of the latter. Each XML fragment/document is identified by a distinct path expression, which is encoded in a distributed hash table. Our framework differs from content-based routing mechanisms, biased towards finding the most relevant peers holding the data. We perform fragments placement and enable fragments lookup by solely exploiting few path expressions stored on each peer. By taking advantage of quasi-zero replication of global catalogs, our system supports fast full and partial XPath querying. To this purpose, we have extended the Chord simulator and performed an experimental evaluation of our approach.

Categories and Subject Descriptors: H.2.4 [Database Management]: Systems-distributed databases, query processing
General Terms: Algorithms, Design.
Keywords: XML Querying, XPath, P2P Networks, Distributed XML Indexes.

1. INTRODUCTION

Filesharing systems, such as Gnutella [11] and Kazaa [16], are identified as *unstructured* systems that rely on flooding to perform *keyword-based* searches. In these systems, there is no guarantee on the query answering whatsoever. Other kinds of systems addressing content-based routing are gaining momentum, namely the so called *structured* systems, exploiting indexes for biasing the search towards particular peers. Among these indexes, undoubtedly DHTs are the most promising ones due to their reduced search complexity and consequent high guarantees of query answering [26]. In light of this, DHTs are starting to be considered as the foundational indexes for data management applications on top of P2P. Among the existing proposals and research prototypes, it is worth to mention [21], [14], [8]. [21] presents an Internet-scale relational engine which builds on an inverted file for term searches. [14] considers each peer holding partitions on relational tables and extends the DHT for addressing approximate range queries. [8] augments the DHT with a B+-tree for answering exact range queries on tuples. In this paper, we focus on *the problem of answering XPath queries in a DHT-based network*. Indeed, path queries are quite different from keyword-based queries and from range queries. Thus, the extensions presented in [21], [14], [8] are not applicable to our case.

A key problem in handling XML data on top of P2P systems is to efficiently locate the data of interest. Recent works [19, 10] have devised mechanisms to quickly find with high probability the peers of interest to address a specific look-up of XML data. They build on the replication on each peer of more or less sophisticated data summaries/catalog information (path summary, histograms, bloom-filters [10] and multi-level bloom-filters [19]) to guide the lookup towards the most relevant peers. However, these solutions may not be scalable for a large number of peers and for arbitrary XML documents to be shared. Moreover, since these data summaries describe the local data as well as the data of the peer neighbors, it is difficult to maintain them up-to-date w.r.t. the changes in the network. There may be different solutions to this problem: *(i)* drastically limit the depth of the document to be distributed as done by [19]; *(ii)* limit the number of peers in the network, i.e. consider a community of independent data providers as in [10], who share the data and tend to remain in the network almost all their life; *(iii)* choose a quasi-zero replication approach, i.e. avoid repli-

†Work done as a summer intern at Icar CNR, Italy.

*This work is supported in part by the italian projects: MIUR Information Society, "SP1" and FIRB "GRID.IT" (RBNE01KNFP).

WIDM'04, November 12–13, 2004, Washington, DC, USA.

cating the *full* catalog information on each peer while still preserving the query capabilities. In this paper, we are interested to operate with assumption *(iii)* in large and dynamic P2P networks where peers can join and leave the network arbitrarily (as in [11, 16]). We have devised a system, called *XP2P* (*XP*ath for P*2P*), which employs XPath to handle XML fragments and to locate them afterwards by solely exploiting a small set of XPath expressions. We argue that our approach is more scalable for an arbitrary number of peers w.r.t. solutions proposed thus far. Indeed, it is worth to notice that, in order to ensure the maximum scalability, we do not perform replication of global catalog information, as opposed to [19, 10]. Indeed, for a peer that stores an XML fragment(s), it is sufficient to store a small set of *related* XPath path expressions, namely the path expression describing itself, the path expressions of its sub-fragments and the path expression of its super-fragment. Details on how to identify these related path expressions are described in Section 3. To this purpose, we employ an extremely lightweight indexing mechanism, which let us store on each peer extra information for only few bytes. To give an example, we have measured on a network of 10000 XML fragments (randomly generated from XMark documents [25]) whose size ranges the interval $3KB - 20MB$, that the related path expressions have a total size ranging the interval $100B - 40KB$.

Moreover, XP2P restricts to the cases in which there are only a *few replicas of the XML fragments*. The cases in which there is a large number of replicas can be easily handled by unstructured networks [11, 16], as they would lead to crawl all the network. An hybrid solution marrying Gnutella with PierDB [15] is presented in [21], which focuses on keyword-based queries. We think an extension of XP2P in this direction can rely on the results presented in [21] and is outside the scope of this paper.

Statement of the problem and underlying motivations. Given an XPath expression p_x describing an XML document or a fragment of it and a set of peers N, our goals are: to uniquely assign p_x to a peer in N, which holds the fragment corresponding to p_x; to also store within the peer the related path expressions of that fragment, i.e. the path expressions identifying the external fragments linked to the original fragment; to guarantee a look-up of a path expression in the network, by solely using the DHT and the related path expressions. Our design builds on current P2P technology. The contributions of the paper are the following:

- A framework for handling XML data content in a structured P2P network, and efficiently storing and querying the data by means of few related path expressions stored on each peer.
- A lightweight DHT for XML fragments that let us locate the peers of interest and hence evaluate XPath queries.
- Two algorithms addressing (the first for queries with *child*-axis, the second for *descendant*-axis) the evaluation of XPath lookups. Fragment lookup is thus done without the need of storing global catalog information.
- An extensive experimental evaluation of XP2P scalability and query routing performances.

There are several motivations behind this work. File-sharing systems, customarily known as such, are being considered the new architectures for data integration [17] but are not limited to play that role [13]. We believe that XP2P is widely applicable in all those cases when there is no mediation between the peer schemas and a common knowledge of the global schema is unforeseeable. An example would be that of organizations that, for *privacy* reasons, tend to keep, within each peer, only partial information of external peers XML data and still wants to query that data when a complete schema knowledge is lacking. As highlighted by recent important research [20], such a complete schema knowledge may be impossible to achieve even in a centralized scenario, thus leading to schema-free XML querying.

The rest of the paper is organized as follows: Section 2 discusses in detail the existing literature on this topic; Section 3 describes how we model XML data in XP2P; Section 4 illustrates the principles behind XP2P DHT; Section 5 shows the algorithms for XPath expression lookups; finally, Section 6 discusses our prototype and the experimental results.

2. RELATED WORK

Enhancing P2P networks with query processors is indeed a timely topic, which has received the attention of the database community these last years [21, 14, 8]. In [14], the shared data are horizontal partitions of relational tables and the queries answered are approximate range queries. A locality preserving hash function is used to map query ranges on the network: this improves significantly query response time because similar ranges are very likely hashed to the same peer. Relational range queries in P2P networks are also addressed in [8] by Gehrke et al. using a distributed version of a B^+-Tree called P-Tree such that every peer contains a subset of the tree and some routing information. However, Ptrees are not highly scalable since every peer can hold at most one tuple. In XP2P, we focus on the general case when a peer can hold more than one XML fragment at a time. Moreover, the replicated index consists of few path expressions and is often smaller than the entire P-tree. Distribution of XML has recently attracted the attention of the community. Distributing XML in a small community of providers or in a distributed repository is, however, remarkably different w.r.t. distributing (and replicating) it in large and dynamic P2P networks. Gertz et al. in [4] proposed a fragmentation method and an allocation model for a virtual XML repository. While this approach is well suited for a static situation where there are a set of peers that never leave the network, it lacks support for highly dynamic contexts such as P2Ps. Distribution and replication for documents enriched with service calls is addressed in [1], where the distribution is guided by the services execution and not applicable here. XML distributed query evaluation is addressed in [27] but the results are valid for hundreds of sites, not for thousands of them. Thus, the problem is still unsolved for large dynamic P2P networks, except for recent papers [19, 10]. [19] considers hierarchical P2P network topologies where every peer contains the XML content and statistical information about the neighborhood. [10] extends Chord for XML data by using tag names as hash keys. However, this approach introduces overhead when a large number of sites has the same tag. The authors say that their framework is extensible for supporting paths instead of tag names but they do not discuss the extensions to handle linked XML fragments. Both approaches [19, 10] lack scalability for a large number of peers and in case of frequent updates of the network. Similarly to [10], XPeer [24] is a

system for sharing XML data, which uses full tree-guides to perform query evaluation. Differently from [10], however, it is not DHT-based and assumes the presence of super-peers.

Data integration and exchange for P2P networks is addressed in [17],[28]. In [17] mapping tables on peers represent the aliases of the same item in the network. Then, a lookup search on a peer is conducted by looking at its mapping table to find the name aliases on other peers. Halevy et al. in [28] proposed a framework for data integration based on a global mediated schema: every query is first translated w.r.t. this global schema and then forwarded to the peers whose local schema can be mapped to the global schema. In XP2P, we do not assume mapping tables or a global schema, but the extension of P2P for data integration is an interesting future direction.

Exact-match queries are not sufficient for searching a large set of XML documents shared on a network. The class of XPath queries proposed here includes partial-match XPath queries, which is reminiscent of tree pattern relaxations presented in [3]. However, the latter only focuses on XPath full-text queries in a centralized scenario.

3. DATA MODEL IN XP2P

We now study how to model XML fragments in XP2P. An XML document can be seen as an unranked, labelled tree t having a distinguished root node r. Given an XML tree t, we first define $\mathcal{X}P$ as the set of distinct linear *root-to-node* path expressions, each starting at the root of t, namely r, and leading to some node in t. A *linear* path expression $/s_1[i]/s_2[j]/../s_n[k]$ is a path expression which only uses the *child* axis and optional positional filters $[i], [j], \ldots, [k]$ on steps $s_1, s_2 \ldots s_n$, respectively [1]. Positional filters $[i], [j], \ldots, [k]$ indicate the positions of elements/attributes in their corresponding *document order*.

Given an XML tree t and the set $\mathcal{X}P$, an XML fragment f is a subtree of t rooted in some node of t, n_f, and identified by the *distinct* linear absolute path expression in $\mathcal{X}P$ starting from r and leading to n_f. We call this distinct path expression, the *identifier* of the fragment.

More precisely, we model an XML fragment as a labeled tree, whose nodes may have arbitrary labels and the special label `sub`, which is used to include the root-to-node path expression of a child fragment. Node labeled with `sub` can only contain as values path expressions in $\mathcal{X}P$ and these path expressions must have as their prefix the *identifier* of that fragment. This is reminiscent of vertical partitioning in database [22] (vertically partitioned attributes belonging to the same table), but different though due to the hierarchical nature of XML data.

Given an XML fragment, we can define the set of *related* path expressions of that fragment as follows:

- *super fragment* path expression p_s: it is the path expression of the fragment which is the ancestor of the current fragment; an ancestor fragment is not necessarily the *parent* axis of the path expression of the current fragment but it can be any of its *ancestor*-axes;
- *child fragment* path expressions p_c: they are the path expressions stored as PCDATA within `sub` tags in the fragment.

[1]Observe that, being the root unique, the only positional filter $[i]$ admitted on the first step s_1 is [1].

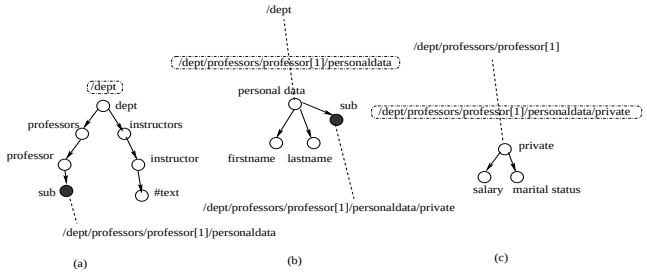


Figure 1: An example of linked fragments with their related path expressions.

Fragments which refer each other by means of related path expressions are called *linked* fragments. Obviously, while a fragment can have several p_c, it can only have one p_s. Note that all the related sub-fragments p_c of a given fragment appear in document order in the latter.

An example of linked XML fragments are illustrated in Figure 1, where fragment (a) is the root fragment (its identifying path expression, enclosed in the dashed round-corners box, is exactly the root) and has one p_c path expression to fragment (b), whose p_s is the path expression of fragment (a). Note that p_s for fragment (b) is not necessarily its parent step, but an ancestor step (in such a case, the root itself). Finally, fragment (b) points to fragment (c), which has no child path expression. Note that fragments (a), (b) and (c) taken separately (i.e. without looking at the related path expressions) are found to be affiliated by simply matching their identifiers prefixes.

In XP2P, any XML fragment comes equipped with its identifier, plus (possibly) a set of p_c path expressions and a p_s path expression. These fragments are stored in the peer with their actual content and with its related path expressions p_s and p_c. This is realized in XP2P by extending the Chord nodes, as explained in next section.

Remark We do not assume that each fragment has to be mandatorily linked to other fragments for being considered. Indeed, it may happen that the fragment does not own any related path expression and/or its identifier does not refer to a super-fragment identifier. In such a case, the fragment can still exist in the network and being queried, by means of its identifier, i.e. simply the path expression leading to its root. Note that if the same path expression comes from two distinct documents, it simply suffices to prefix it with the name of the originating document.

4. PUTTING PATHS IN A DHT

An interesting issue is how to access the fragments once these have been stored within the peers.

Previous work [4] dealt with the problems of building XML distributed repositories, where each site holds local indexes and global information (up to 47% of the original document overall). This is not feasible in a P2P, because, being the latter very dynamic, the approach adopted in [4] would lead to replicate global information on every peer and to maintain this information up-to-date each time there is a variation in the network. In an highly dynamic P2P context, one is mainly interested to *lookups*, i.e. to know that a peer holds a fragment, to retrieve it and to possibly cope with network changes without global disruption. Past works [19, 10] utilized catalogs and data summaries (i.e. multi-level

bloom filters [19], distributed path summaries, B+trees and histograms [10]) that partially or entirely describe the local/global data. These approaches are probabilistic, and are proved to work in small communities of data providers better than in large and dynamic P2P.

Our target are conceivably P2P networks with an high turnover of nodes. Such networks maintain as less as possible global information at every peer to avoid global refreshments of all the network. A graceful solution is that of using a light *distributed* index structure that let the peer know about a few (but sufficient in practice) other peers. In particular, this structure must be highly adaptive to the changes of the network, i.e. it must ensure that the entire network does not disrupt when a peer is unreachable. This is realized by Chord protocol by means of a distributed hash table. A Chord node is aware of $O(logN)$ other nodes for efficient routing, as opposed to other protocols which used consistent hashed fingers on all the other nodes. Being the routing table distributed, Chord turns to be highly scalable to a large number of nodes.

Each node (e.g. N_8 in Figure 2) in original Chord stores both the hashed keys of the successors at logarithmic distance (the so called finger table) and of the node predecessor. *XP2P* keeps these hash keys to ensure logarithmic lookup and further extends the Chord ring (see Figure 2) by storing on a node the following information: the local content of a fragment(s), and its related path expressions, i.e. the list of their child fragments and their super fragment path expressions. These are to be hashed directly in the Chord hash space and may exploit the Chord finger table and the node predecessor.

Suppose we are looking for the fragment `/dept/professors/professor[1]/personaldata/` in the Chord ring in Figure 2 (where `(.....)` stands for `/dept/professors` omitted for space reasons), i.e. fragment (b) of Figure 1. We can hash it directly in the Chord hash space. The same happens with child fragments `/dept/professors/professor[1]/personaldata/hobbies` and `/dept/professors/professor[1]/personaldata/private` and with the super fragment `/dept`. They can be accessed by means of Chord lookup mechanism, which uses the finger table and the node predecessor. However, in order to embed path expressions in Chord, we need to further modify the ring, i.e. to *equip each node with the additional capability of "hashing" path expressions*. Moreover, for better manageability, we have chosen to store the path expressions in a different format, other than hash keys. We describe our approach in paragraph 4.1.

Reliable replication of XML fragments According to the data model presented in Section 3, the fragments may be linked one to another and their identifiers, i.e. the linear path expressions identifying them, are distinct one w.r.t. the other. However, in order to guarantee reliability, XP2P admits fragments replica in a controlled manner. Each fragment having an identifier path expression is stored on the peer uniquely accessible through that identifier and *may be replicated* on its successor nodes.

Thus, for instance in Figure 2, the fragment identified by `/dept/professors/professor[1]/personaldata` belong to the peer $N8$ and can be redundantly replicated on each successor peer in the finger table. This is done in the spirit of DHASH [7], which first added a storage layer to Chord. The level of redundancy can be varied and can successfully

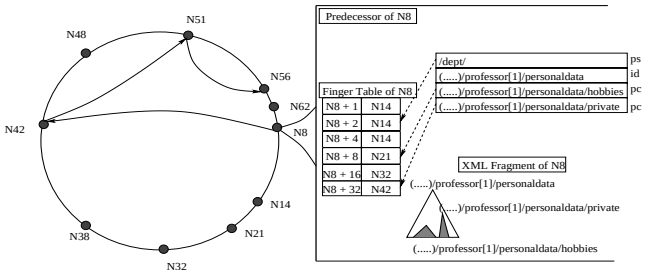


Figure 2: XP2P extension of a Chord ring.

cope with the situations in which node failures occur. The problem of replication to guarantee a reliable query response is orthogonal to our problem and left as future work. Here, we simply adopt a well-known solution.

4.1 Fingerprinting path expressions

Path expressions ranging the set $\mathcal{X}P$ are distinct path expressions often sharing a prefix between each others. A straightforward extension of Chord would lead to directly hash these path expressions, using SHA-1. However, we have devised a more suitable solution. *Instead of hashing* them, XP2P reduces them to shorter *fingerprints* [23, 5, 6], i.e. bit tokens exhibiting the same length overall (at most 8-9 bytes against 20 bytes obtained with SHA-1 used in Chord). Path fingerprinting in XP2P is done in the spirit of URL fingerprinting [6], since the space of paths resembles the space of URLs [2]. Besides reducing the occupancy of the original strings for URL caching as done in [6], fingerprinting in our case brings two main advantages. First, it has a nice *concatenation* property, which leads to quickly compute path expressions for fragments lookups. Such a property, discussed in detail below, only pertains to fingerprints and not to hash keys, thus motivating the encoding of path expressions as fingerprints rather than as hash keys. Secondly, we foresee the use of fingeprints to address authenticity problems in P2P networks, as discussed next.

The fingerprinting function is similar to hash functions as it can be seamlessly applied when the items to be fingerprinted differ in at least one bit [5, 23], which is true for distinct path expressions in $\mathcal{X}P$. Thus, even collections of homogeneous documents can be reduced to distinct fingerprints by differentiating them with namespaces. In the remainder, for the sake of clarity, we will ignore this technicality.

Overview of the fingerprinting scheme due to Michael Rabin [23] Let $A = (a_1, a_2, \ldots, a_m)$ be a binary string. We assume that $a_1 = 1$, otherwise a prefix bit can be used. We associate to the string A a polynomial $A(t)$ of degree $m-1$ with coefficients in the algebraic field $\mathcal{Z}_2$, $A(t) = a_1 * t^{m-1} + a_2 * t^{m-2} + \cdots + a_m$. Let $P(t)$ be an irreducible polynomial of degree k, over $\mathcal{Z}_2$. Given $P(t)$, the fingerprint of A is the following: $f(A) = A(t) mod P(t)$. The irreducible polynomial can be easily found following the method in [23] and an interesting example is in [18] (p.542). The method let pick uniformly at random a polynomial of degree k and compute the probability that $A(t)$ divides it. The latter being the probability that two strings have the same fingerprint, must be kept sufficiently small.

[2]Linked URLs are reminiscent of path expressions sharing the same prefix.

Computation of path expressions fingerprints. The path expressions used to identify fragments are drawn from the set $\mathcal{X}P$. Indeed, the computation of each short token associated to each path expression can proceed incrementally, as stated by the following important *concatenation* property of the fingerprinting function [23]: the fingerprinting of the concatenation of two strings can be computed via the equality: $f(concat(A, B)) = f(concat(f(A), B))$. Consider for instance the following path expression: `/dept/professors/professor/private`, and suppose we are given the fingerprint of `/dept/professors/professor`, i.e. $f($`/dept/professors/professor`, then $f(concat($`/dept/professors/professor/private`$)) = f(concat(f($`/dept/professors/professor`$), $`/private`$))$. The same holds for steps and filters of a path expression, which are concatenated one after the other (e.g. `/professor[1]` as concatenation of `/professor` and `[1]`). The above property is extremely useful when updates are performed locally on the fragments roots. Indeed, instead of recomputing the fingerprint of the modified fragment from scratch, the fingerprint of the parent is concatenated to the new root tag. In order for a peer to compute the related path expression fingerprints, it only suffices to store the irreducible polynomial on that peer. The latter has a fixed degree equal to $N_f + 2 * D_{max} + Q$, where 2^{N_f} is the number of fragments in the network, $2^{D_{max}}$ being the length of the longest path expression in the network and 2^Q being a threshold due to the probability of collision between two arbitrary distinct tokens [5]. In our setting, we use a degree of 64, which leads to an acceptable probability of 2^{-10}, and let us exploit a maximum length for path expressions of 50 steps (averaged on a length of 10 characters per step) and a maximum number of fragments equal to 2^{30}, which is quite huge. Observe that this polynomial is a quite small structure to be *replicated on each participating peer* if compared to replicated structures used in other approaches (e.g. multi-level Bloom filters, restricted to documents with 50 distinct elements and 3-steps paths, in [19] and a combination of P-Indexes, A-Indexes and T-Indexes, measuring from 22% up to 47% of the whole document, in [4]). Moreover, such a polynomial can accommodate changes of the network till the maximum number of fragments, 2^{30} and till the maximum depth of a document (i.e. 50). [3]

As an extension, we devise the use of fingerprinting to verify the authenticity of a fragment. Indeed, within the XML content of a fragment, one can think of storing the fingerprint of its entire XML content as an additional component. Using fingerprints to do this would imply to only need the stored polynomial to check the authenticity of that fragment. Conversely, with hash keys one would have to store the entire set of hash keys (of XML content) present in the network.

Fragment and path allocation. In summary, within Chord a node's identifier is chosen by hashing the IP address by means of SHA-1 [9]. We have made an extension of Chord using fingerprints instead of hash functions. Besides the successor nodes and the predecessor node, which are natural to the Chord protocol, each peer in XP2P *stores* minimal *access information*, properly *fingerprinted*: *(i)* the fingerprint of its own identifier; *(ii)* the fingerprint of the super-fragment p_s; *(iii)* a list of fingerprints of path expressions of the external sub-fragments, p_c.

[3]These are quite acceptable bounds since, only if these are abundantly overcome (of *at least* one order of magnitude, e.g. from 2^{30} to 2^{31}), the degree of the polynomial needs to be adjusted.

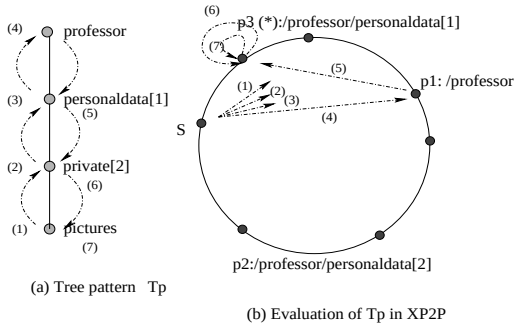
5. PARTIAL AND FULL FRAGMENT LOOKUP IN XP2P

Once the path expressions have been fingerprinted, it is interesting to understand how to use them for fragments lookups. We allow two kinds of lookups, namely partial and full. By partial lookup, we shall mean the lookup of a fragment which only partially matches the original request. For instance, consider the case in which the fragment of Figure 2 located on peer N_8 is looked up. N_8 is reached through the DHT, and its fragment returned. This answer is partial as the two sub-fragments rooted in `sub` tags are not retrieved. Conversely, by full lookup we shall mean the retrieval of the fragment completed with its sub-fragments, until each `sub` element is unfolded.

The rationale behind these two kinds of lookups derives from many popular P2P networks [11, 16], where usually the required items are searched until a time-out expires or until a certain number of findings is reached. Moreover, since a P2P network is highly dynamic, it is reasonable to have a partial query answer. Indeed, a node can always leave the network due to failures. Already in such a case, a complete query answer which retrieves that node cannot be guaranteed. Moreover, the interest for partial query evaluation is increasing even in the centralized cases, as recent important work shows [3].

The XPath fragment we consider for full or partial lookups is $\{/, //, []\}$, where filters are restricted to positional filters as they are used to identify placed fragments. Observe that we are only able to evaluate unrestricted value predicates on local fragments, since these are not directly encoded in the DHT. We distinguish between path queries containing only the *child* axis and those containing *descendant* axis. We focus here on the first kind and give an algorithm for their evaluation. The others are deferred to Section 5.1.

To represent the query pattern, we use the formalism introduced in [2]. Answering a tree pattern t_p of length n in $\{/, []\}$ is done by fingerprinting it directly in the Chord ring. However, this search will be successful only when there is an *exact-match* between the tree pattern and an existing fragment. We call a *miss* the lack of a fragment on the network, after an exact-match fails. A miss can be due to the absence of a fragment on the network for two reasons: *(i)* the fragment was never placed on the peers; *(ii)* the fragment temporarily left. When a miss happens for a tree pattern, the search is biased towards finding at least a *partial-match*. Thus, the tree pattern is *pruned* of a step at a time (by step we mean a compound step, composed of a step name plus a positional filter) and then possibly resumed afterwards (see Figure 3). More precisely, we start looking exhaustively at the $n-1$ prefix path expressions corresponding to t_p in a bottom-up order. As soon as one of these prefixes is hashed on a peer p_i, we can stop the bottom-up evaluation at p_i. Henceforth, we start analyzing the local content of p_i and its sub-fragment list in a top-down fashion and further on, until a result is found or the time-out expires. The evaluation of a tree pattern varying in $\{/, []\}$ is thus done in a composite bottom-up/top-down fashion. This *hybrid* evaluation allows to bidirectionally navigate the network since it may happen



(a) Tree pattern Tp

(b) Evaluation of Tp in XP2P

Figure 3: An example of lookup for a pe in $\{/, []\}$.

that a miss happens while doing bottom-up evaluation (for example, for a temporary node failure) and that the miss can be recovered when doing top-down (due to new joins in the network).

As an example of evaluation on the data of Figure 1, consider Figure 3, which shows a tree pattern issued on a network constituted of peers p_1, p_2 and p_3. The peer S starts the evaluation at step *(1)*, when a lookup is performed on path expression `/professor/personaldata[1]/private[2]/pictures`. The fragment does not belong to the network, thus its prefix `/professor/personaldata[1]/private[2]` is looked up *(2)* but not found, and so on, `/professor/personaldata[1]` *(3)* is searched, but not found since the peer p_3 is temporarily down, till `/professor` *(4)*, which is instead correctly found on peer p_1. Thus, we can start the top-down evaluation from `/professor` *(4)*, look in its sub-fragments and check again `/professor/personaldata[1]` *(5)*, which is now up and can correctly answer our query (*(6,7)* are evaluated locally on peer p_3). Results are shipped back on peer S and reconstruction of results is done on peer S, which indeed originated the query.

As a final remark, note that the complexity of partial-match *child* axis queries is $O(N_s \times log(N))$, where N_s is the number of steps of the path expression and N the number of peers in the network.

5.1 Towards XPath queries: An algorithm for the *descendant*-axis

Evaluation of the descendant-axis is noticeable as it cannot be done by fingerprinting the tree pattern (or its prefixes) and look for its fingerprint directly. In addition, no global schema is available in XP2P to guide the evaluation of the descendant axis towards some particular peers. One naive solution is an exhaustive search over the entire ring that starts at an arbitrary peer. However, this solution, although complete, would lead to many unuseful accesses. We have devised a better solution, which searches the space of the peers (and thus of the fragments located on those) by selecting *peer by peer* the most promising direction. The algorithm relies on two key observations: *(i)* the evaluation of a tree pattern containing a '//' may take place bottom-up or top-down w.r.t. the parse tree, but top-down is preferred [12]; *(ii)* a fragment can always exploit information about the sub-fragments, since it stores their fingerprints.

We first discuss observation *(i)*. To enable the evaluation of the descendant axis in XP2P, we follow the result of [12], which proved that top-down and bottom-up have both polynomial time complexity in a centralized setting,

	Algorithm **Optimistic step-wise query decomposition of descendant axis** Input: a tree pattern t_p containing '//', a list of peers L_P, each peer p_i with a list of paths L_{p_i} Output: query decomposition over L_P
1	start at peer p_i containing the root fragment
2	or, alternatively, at a random peer p_i
3	while L_P is not empty
4	within L_{p_i}, seek "promising" path expressions w.r.t. t_p
5	let these "promising" path expressions be LP_{p_i}
6	let the remainings be OT_{p_i}
7	while LP_{p_i} is not empty
8	find the peer owning a path p_j in LP_{p_i}
9	evaluate the path p_j in LP_{p_i}, $L_P = L_P + p_j$
10	while OT_{p_i} is not empty
11	find the peer owning a path p_k in OT_{p_i}
12	evaluate the path p_k in OT_{p_i}, $L_P = L_P + p_k$
13	$L_P = L_P - p_i$

Figure 4: Optimistic Step-Wise Decomposition Algorithm for descendant-axis.

but top-down yields less intermediate results than bottom-up. In XP2P, evaluation of a step with '//' proceeds as much as possible *top-down*. We optimistically start evaluation of '//' from the peer containing the "context" fragment. If the latter is down, we can always start at any other peer, as much as possible closer to the context. The context fragment and the fragments closer to the context are known to any arbitrary peer, since they prefix the related path expressions of that peer.

Observation *(ii)* leads to devise an evaluation strategy for the *descendant*-axis. For simplicity, we describe here the evaluation of a tree pattern of the kind $/s_1/..../s_i//s_j/.../_{j+k}$, as the extension to multiple descendant-axis steps is a composition of the former. On an arbitrary peer, we can explore the local information and the related path expressions to find the answer to the above tree pattern. At an arbitrary peer p_j, we can have four cases, i.e. either find *(1)* local s_j elements, i.e. contained in the fragment; *(2)* intermediate steps s_j of the related path expressions; *(3)* last steps s_j of the related path expressions or *(4)* not find them at all as steps in the related path expressions. In case *(1)* the elements are local and can be retrieved. Case *(2)* is not meaningful when evaluation is proceeding top-down, since it means that s_j elements which appear as intermediate steps were encountered in the past and are already included in the query result. Cases *(3)* and *(4)* are the most significant as they provide new directions to explore: let us call the first path expressions (*(3)*) *promising* path expressions LP_{p_i}, and the remaining ones (*(4)*) OT_{p_i}.

These considerations lead to the *optimistic* step-wise decomposition algorithm in Figure 4 for a tree pattern containing a descendant-axis. In its first step, the algorithm performs a local search on the current peer, and proceeds by following the promising directions indicated by sub-fragments whose root is the searched element. All the other remaining sub-fragments are probed afterwards. Observe that the complexity of the *descendant* axis query evaluation is bounded by $O(N_f \times log(N))$, being N_f the total number of fragments and N the number of peers in the network.

An arbitrary path expression in $\{/, //, []\}$ is then solved by doing a composition of exact-match and partial-match lookups for linear path expressions and of the optimistic step-wise algorithm for the descendant-axis. For the sake of conciseness, details are omitted.

Finally, observe that once the context node $/s_1/..../s_i$ (i.e. the node on which evaluation of a path expression starts) has been found, retrieval of s_j can proceed in parallel on all the paths in LP_{p_i} (OT_{p_i}, resp.). This leads to a quite flexible distributed query processing, and has been validated by our experiments. Observe that, it is also in order to maximally exploit parallelism that assembling of results is done directly within the peer which originated the query.

6. EXPERIMENTAL STUDY

The implementation bases on the Chord simulator, whose `find` operation has been extended to accept a linear path expression instead of a document ID. Since the Chord simulator does not come with a storage layer, we built our own, by using native BerkeleyDB [29]. Indeed, the fragments content and the related path expressions properly fingerprinted are physically stored in BDB B+-trees. We have pursued several experiments on the XP2P prototype. All the experiments have been conducted by using different instances of the XMark data set [25], more precisely we have considered three sets of XMark documents fragments, ranging the sizes shown in the following table:

Name	*Min Frag. Size*	*Max Frag. Size*	*Avg Frag. Size*
Small	*1KB*	*1.3MB*	*3KB*
Medium	*3KB*	*18MB*	*16KB*
Big	*8KB*	*29MB*	*41KB*

Mainly, the experiments were targeted to ascertain the validity of our approach, in particular to assess the load distribution achieved by fingerprinting w.r.t. classical hashing used in Chord; to measure the impact of storing the related path expressions on each peer; to test the query performances of Algorithms of Sect. 5 and the system scalability while varying the number of peers in the network.

Fingerprinting load distribution. The first set of experiments were aimed at showing that replacing the SHA-1 hash keys used in Chord with fingerprints, besides obvious advantages in terms of occupancy, computation and authenticity, is fairly equivalent for load balancing. To this purpose, we have measured the number of fragments assigned to each peer (*peer#* indicates the participating peer), when 10000 fragments are handled by the DHT on 512 nodes.

The results of Figure 5 shows that, even if there are a few dominant peaks for fingerprinting, each peer is holding a comparable number of fragments. However, fingerprints are more attractive than simple hashing for path expressions for what we discussed in Section 4.1. Other experiments (not shown for space reasons) executed on larger numbers of nodes gave a similar load distribution.

Occupancy of related path expressions. In order to show the occupancy of the related path expressions, we have performed an experiment to measure the average/maximum size of those for each participating peer, while varying the number of fragments. The experiment has been executed on the *Big* set of XMark fragments. As it can be noted in Figure 5 (bottom), in a network of 1000 peers, the maximum size is nearly $14KB$ when 20000 fragments are handled in the network.

Lookups performances Moreover, we have explored the performance of two kinds of path queries: *(1)* path queries containing only `/` when used to do full or partial searches

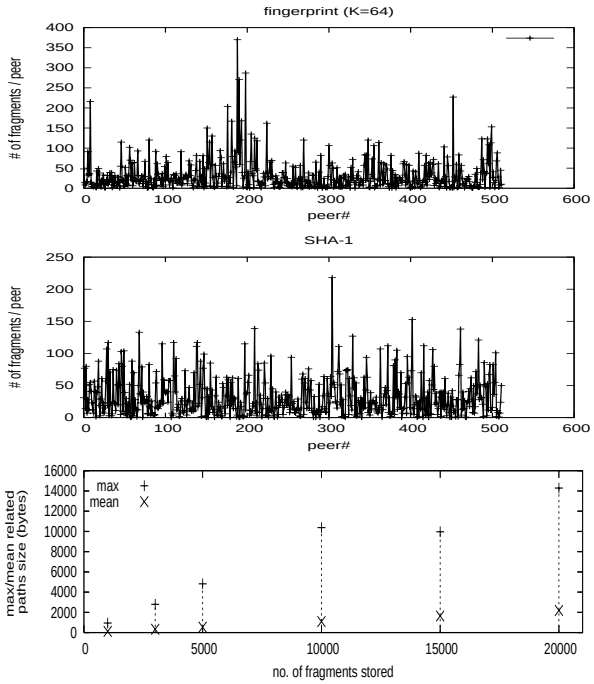


Figure 5: Load distribution with a 512-nodes system and 10000 fragments with Fingerprinting (top) and SHA-1 (medium). Occupancy of the related path expressions (max, mean) for a 1000 **nodes network (bottom).**

(using the Algorithms of Section 5); *(2)* queries of the kinds $/s_1/..../s_i//s_j/.../s_{j+k}$, employing the Algorithm 4.

We have conducted experiment *(1)*, first by considering only exact-match XPath queries and secondly by considering partial-match XPath queries (under the meaning defined in Section 5). In both experiments, we have considered a network with approximately 10000 fragments. The query set for the experiment on exact-match XPath queries has been chosen by randomly picking path expressions of fragments present in the DHT. Conversely, the query set for the experiment on partial-match XPath queries has been chosen by constructing synthetic path expressions that *(i)* do not belong to the DHT; *(ii)* may have in common a prefix with a path expression present in the DHT. For example, in Figure 1, a synthetic path expression is `/dept/instructors`.

Figure 6 (top, medium, resp.) plots the number of hops that have been made to complete the query under different configurations of the network (ranging $500-5000$ peers). In these experiments, we have considered a time-out of 1000 ms and counted the nr. of hops in all cases, included those in which there are misses. It is worth to observe that in both cases of exact-match queries and partial-match queries, the trends are pretty much linear for the three data sets. This let us conclude that the behavior is independent of the fragment sizes present in the network, but mainly depends on the way these fragments are balanced in the DHT. In other words, this is a confirmation that XML fragments (of any size) can be handled in a P2P network by means of a DHT without causing bottlenecks on the peers holding the largest fragments.

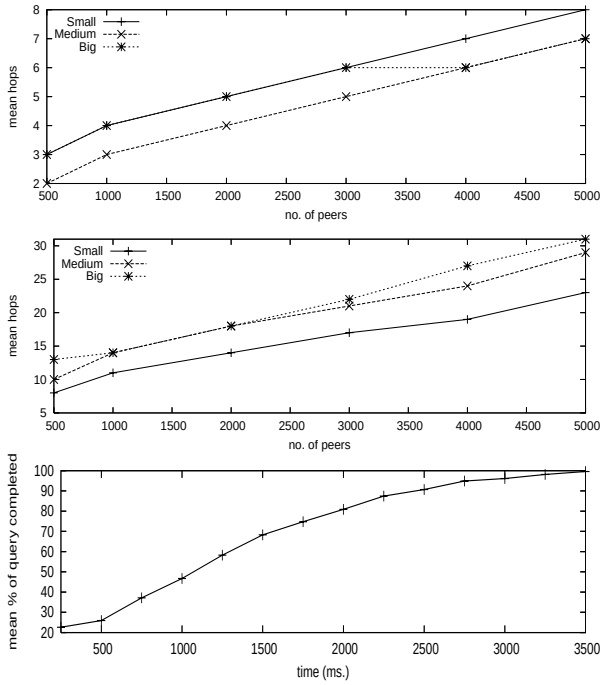


Figure 6: Nr. of hops measured for the exact-match queries (top) and partial-match queries (medium). Query results w.r.t. time for the step-wise Algorithm 4 (bottom).

Finally, we have conducted experiment *(2)* for '//' by considering a 1000 peers network with 10000 fragments ranging the *Medium* dataset sizes. Figure 6 (bottom) depicts the average percentage of the query completed w.r.t. time. We conclude that the implicit parallelization of the query evaluation process is fully exploited and allows to complete the query answering in a relatively small amount of time.

7. CONCLUSIONS AND FUTURE WORK

We have presented XP2P, a P2P framework for answering XPath queries. XP2P uses XPath to identify XML fragments in a P2P network and only needs to store few related path expression and a polynomial on each participating peer. Our system has been implemented on top of Chord [26]. The results gathered from the experimental study are very encouraging in that the lightweight indexing mechanism let us answering queries in still reasonable times and offers scalability w.r.t. the number of peers and fragments in the network. Our prototype is thus far enabled for lookups of simple path expressions. Future work will be devoted to investigate ad-hoc evaluation mechanisms of more expressive XPath queries and further optimizations of XPath queries containing the descendant axis.

Acknowledgments

The authors would like to express their gratitude to Prof. Ion Stoica and the Chord team for making the software publicly available.

8. REFERENCES

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

[2] S. Amer-Yahia, S. Cho, L. Lakshmanan, and D. Srivastava. Minimization of Tree Pattern Queries. In *Proc. of SIGMOD*, 2001.

[3] S. Amer-Yahia, L. V. Laksmanan, and S. Pandit. FleXPath: Flexible Structure and Full-Text Querying for XML. In *Proc. of SIGMOD*, 2004.

[4] J.-M. Bremer and M. Gertz. On Distributing XML Repositories. In *Proc. of WebDB*, 2003.

[5] A. Broder. *Some Applications of Rabin's Fingerprinting Method*. Springer-Verlag, 1993.

[6] A. Broder, M. Najork, and J. Wiener. Efficient URL Caching for World Wide Web Crawling. In *Proc. of WWW*, 2003.

[7] E. Brunskill. Building peer-to-peer systems with chord, a distributed lookup service. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, page 81. IEEE Computer Society, 2001.

[8] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. Querying Peer-to-Peer Networks Using P-Trees. In *Proc. of WebDB*, 2004.

[9] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. of SOSP*, 2001.

[10] L. Galanis, Y. Wang, S. Jeffery, and D. DeWitt. Locating Data Sources in Large Distributed Systems. In *Proc. of VLDB*, 2003.

[11] Gnutella homepage. `http://www.gnutella.com/`.

[12] G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. In *Proc. of VLDB*, pages 95–106, 2002.

[13] S. D. Gribble, A. Y. Halevy, Z. G. Ives, M. Rodrig, and D. Suciu. What Can Database Do for Peer-to-Peer? In *Proc. of WebDB*, 2001.

[14] A. Gupta, D. Agrawal, and A. E. Abbadi. Approximate Range Selection Queries in Peer-to-Peer Systems. In *Proc. of CIDR*, 2003.

[15] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proc. of VLDB*, 2003.

[16] The Kazaa Homepage. `http://www.kazaa.com`.

[17] A. Kementsietsidis, M. Arenas, and R. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *Proc. of SIGMOD*, 2003.

[18] D. Knuth. The Art of Computer Programming III: Sorting and Searching, 2nd Edition. In *Addison-Wesley*, 1973.

[19] G. Koloniari and E. Pitoura. Content-Based Routing of Path Queries in Peer-to-Peer Systems. In *Proc. of EDBT*, 2004.

[20] Y. Li and C. Y. andH. V. Jagadish. Schema-Free XQuery. In *Proc. of VLDB*, 2004.

[21] B. T. Loo, R. Huebsch, J. M. Hellerstein, I. Stoica, and S. Shenker. Enhancing P2P File-Sharing with an Internet-Scale Query Processor. In *Proc. of VLDB (To appear)*, 2004.

[22] M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, 1999.

[23] M. Rabin. Fingerprinting by Random Polynomials. In *CRCT TR-15-81, Harvard University*, 1981.

[24] C. Sartiani, P. Manghi, G. Ghelli, and G. Conforti. XPeer: A self-organizing XML P2P database system. In *Proc. of P2PDB Workshop, co-held with EDBT*, 2004.

[25] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse. XMark: A benchmark for XML data management. In *Proc. of VLDB*, 2002.

[26] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proc. of ACM SIGCOMM*, 2001.

[27] D. Suciu. Distributed Query Evaluation on Semistructured Data. In *TODS*, 2004.

[28] I. Tatarinov and A. Halevy. Efficient Query Reformulation in Peer-Data Management Systems. In *Proc. of SIGMOD*, 2004.

[29] Website. Berkeley DB Data Store, 2003. http://www.sleepycat.com/pro-ducts/data.shtml.