An Internet-Scale Service for Publishing and Locating XML Documents

Praveen Rao [#1], Bongki Moon [*2]

[#] *University of Missouri-Kansas City, Kansas City, MO 64110.*
[*] *University of Arizona, Tucson, AZ 85721.*
[1] raopr@umkc.edu
[2] bkmoon@cs.arizona.edu

***Abstract*— In recent years, there has been a growing interest for peer-to-peer (P2P) based computing and applications. One of the most important challenges in P2P environments is to quickly locate relevant data across many participating peers. In this demonstration, we present *psi*X, which is an Internet-scale service for publishing and locating XML documents. This service runs on several PlanetLab nodes geographically spread across the globe. The *psi*X system adopts a suite of new techniques for XML indexing and pattern matching in a P2P network, namely, (a) representing XML documents and XPath queries compactly via algebraic signatures, (b) searching signatures of documents and value summaries indexed using distributed hierarchical indexes built over a Distributed Hash Table (DHT), and (c) gracefully adapting to failures while running on the Internet, where failures are a norm rather than an exception.**

I. Introduction

Recent years have witnessed a growing interest and popularity for peer-to-peer (P2P) based computing and applications. The popularity of file-sharing applications (*e.g.*, Napster [1], Kazaa [2]) among Internet users has been paralleled by tremendous amount of research to build scalable and robust P2P systems. One of the most important challenges in such systems is to quickly locate data items that a user is interested in. In a typical file-sharing application, a user poses keyword based queries. For instance, a user may request for "`Britney Spears`" videos on the Internet. The application finds the hosts that store a copy of "`Britney Spears`" videos and returns it to the user.

The extensible markup language XML has become the de facto standard for information representation and exchange on the Internet. Recently there has been a rising interest for P2P systems that adopt XML as their data model [3][4][5][6][7][8]. The ability to model the underlying heterogeneity of data sources and expressiveness of query languages such as XPath and XQuery make XML a suitable choice. Consider a P2P network where users publish different kinds of information such as bibliography, items-for-sale, movie reviews, and personal profiles in XML format. Suppose a user issues the following XQuery query to find the names of people who reside in "`Kansas City`."

```
for $d in collection("P2P")
let $e := $d/person
where $e/address[contains(.,"Kansas City")]
return <Name>{$e/name/text()}</Name>
```

This query can be processed in two phases. In the first phase, all qualifying peers are located who store an XML document that contains a match for the XPath expression `/person/address[contains(.,"Kansas City")]`. In the second phase, the query is executed on the matched XML documents either remotely by shipping it to those qualifying peers or locally by downloading them.

We have developed *psi*X (**p**olynomial **s**ignature **i**ndex for **X**ML) to address the first phase of query processing, that is, to quickly locate relevant XML documents and their publishers. Our system is currently built over a modified Chord DHT [9] and runs on several PlanetLab [10] nodes geographically spread across the globe. *Any user on the Internet can publish XML documents and can pose XPath queries to locate relevant XML documents published by other users.*

II. Our Motivations

In this section, we describe the limitations of prior research work that motivate the design of *psi*X. Most of the previous work use a form of path summarization (*e.g.*, DataGuide summarization [4], root-to-node paths [3], bloom-filters [11], path hashes [6]) to index XML documents and locate them in a P2P network. Therefore, a complex XPath expression is first decomposed into simple paths and each path is processed separately by issuing separate lookups into the P2P network. This approach can potentially increase the total number of node hops required to process a query. Some approaches either cannot process '//' axis or '*' wildcard in XPath queries [6] efficiently or rely on the existence of designated super-peers [4].

Load balancing is a challenge for inverted lists [3] where document summaries are stored on a per-tag basis. Some tags may appear frequently in the data and thus their lists can potentially grow very large. (A similar observation has been made in the work of KadoP [8].) A peer storing a long list would spend more resources and can be overloaded if the corresponding tag is frequently accessed.

We believe a distributed balanced hierarchical index on the document summaries can overcome the aforementioned load imbalance problem. KadoP [8] also builds a hierarchical index, but it has the following shortcomings. The number of DHT lookups depends on the number of nodes in the query pattern. Although parallel lookups can reduce the response time, they

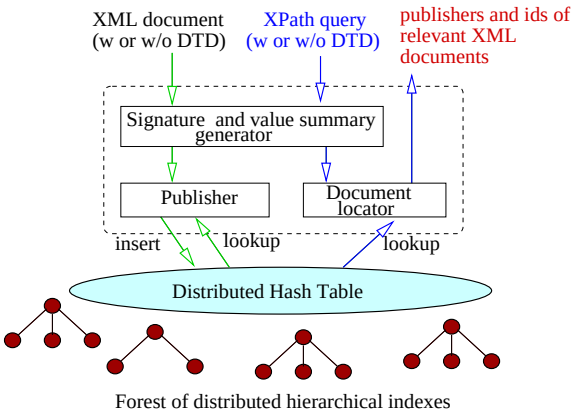
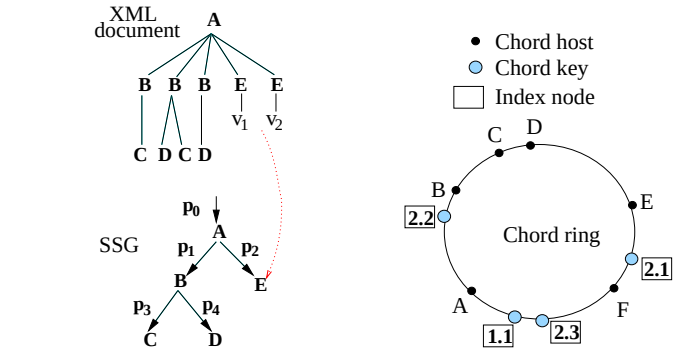


Fig. 1. Architectural overview of *psi*X



(a) XML document tree and SSG (b) Chord ring

Fig. 2. Signature construction, and maintenance of the index

can substantially increase the network traffic. Further wildcard '*' is ignored during query processing.

III. Design of *psi*X

The *psi*X system adopts a suite of new techniques for XML indexing and pattern matching in P2P enviroments. Due to limited space, we describe only the salient features of our system. Further details can be found in the technical reports [12][13]. The architectural overview of *psi*X is shown in Figure 1.

A. Signature & Value Summary Generator

A key component of *psi*X is the *signature and value summary generator*. An XML document is summarized and represented as an algebraic signature that captures its structural characteristics. This signature is essentially a product of irreducible polynomials in Galois Field GF(2) and can be represented compactly as a bit string. Further it can be efficiently operated upon using operations such as division, greatest common divisor (GCD), and least common multiple (LCM). (Irreducible polynomials behave like prime numbers and their benefits over primes are described in [12].)

Consider an XML document tree and its structural summary graph (SSG) as shown in Figure 2(a). An SSG essentially captures the parent-child relationships between the tag names in an XML document [12]. An SSG can be constructed from a DTD or a schema or by scanning the document once. (A dummy incoming edge is added to the root to handle documents with only one element.) The edges of the SSG are labeled with distinct irreducible polynomials. The signature of the document is constructed by traversing the XML document tree in preorder, and matching each document edge with the corresponding edge in the SSG. The product of those polynomials assigned to the matching edges in the SSG denote the signature of the document. For example, the signature of the document in Figure 2(a) is $p_0p_1p_2p_3p_4$. (Documents with recursion of elements are also supported.) Values in the documents are summarized via histograms for numeric data and polynomial signatures for textual data [13]. For example, the values v_1 and v_2 in Figure 2(a) are summarized together (say as V) since their element E map to the same node in the SSG (dotted arrow). The document signature and value summary (p_2, V) are considered for indexing.

An XPath query or a twig pattern is also mapped into its polynomial signature by using the SSG. Both '//' and '*' wildcard can be present in the query. *The existence of a query pattern in a document is determined by dividing the document signature with the query signature.* (The division operation is a polynomial division.) Thus a query pattern can be processed *holistically* without decomposing it into simple paths. The divisibility test is only a *necessary condition* for a pattern match and hence false positives can occur. We observed that signatures can yield above 95% precision in most cases [12]. False positives are discarded when the query is executed over the matched documents.

Sometimes the SSG may not be know to a user issuing the query. For such a case, a publisher can ignore the SSG, and summarize the document into a set of distinct tags (*i.e.*, elements, attributes). Each distinct tag in this set is assigned a distinct irreducible polynomial and their product represents the document signature. The query signature is computed similarly. Note that the structural relationships are ignored and this can lead to lower precision while matching. The value summaries now use the polynomials assigned to their corresponding tag name during signature construction for indexing purposes.

B. Forest of Distributed Hierarchical Indexes

The *psi*X system builds a forest/collection of *distributed* and *balanced hierarchical* indexes for document signatures and value summaries, whose root ids are generated using element names and irreducible polynomials assigned to value summaries, respectively. These indexes are collectively maintained by participating peers. An *H-index* indexes document signatures (index keys) in order to quickly identify those signatures that are divisible by a query signature. For each distinct XML element, a separate *H-index* is built. The index node entries in an *H-index* satisfy the following *containment property* – the signature of an entry in a node is the LCM of the signatures stored in its child node. Thus document signatures that are divisible by a query signature are found by testing for divisibility of a node entry's signature starting from the root of the index. Signatures of structurally similar documents

are grouped together in the index nodes by using a similarity measure called $PSim$ that depends on the number of common irreducible polynomials between two signatures [12].

Textual and numeric value summaries are indexed using hierarchical distributed indexes called *tH-index* and *nH-index*, respectively. A separate *tH-index* or *nH-index* is created based on the irreducible polynomial associated with the value summary. This is because tag names are already used to identify *H-index*es. For example, the value summary (p_2, V) for the XML document in Figure 2(a) is indexed by a *nH-index* maintained for polynomial p_2. While *H-index* and *tH-index* differ only in the way the index keys (*i.e.*, signatures) are constructed, *nH-index* uses one-dimensional intervals as index keys and its containment property is like that of an R-tree [14]. Section III-C describes how these different indexes are utilized during document publishing and query processing.

Each hierarchical index is stored using the key-value pair abstraction provided by the DHT. For example, in Figure 2(b), the root `'1.1'` and 3 child nodes `'2.1'`, `'2.2'`, and `'2.3'` of the index are stored collectively by peers A, B, and F according to Chord's protocol [9]. Thus the signatures are partitioned across participating peers, thereby avoiding the load imbalance problem arising in inverted lists. Since *psi*X is built using the abstractions provided by the DHT, it inherits the DHT's scalability, robustness, and load balancing properties.

C. Publisher and Document Locator

To publish a document, the Publisher inserts the document signature into each *H-index* maintained for every distinct XML element in the document. The value summaries are inserted into appropriate *tH-index*es and *nH-index*es. (If an index does not exist already, it is created by the Publisher.) An index is traversed by performing DHT $lookup$ operations and signatures are added to the index using DHT $insert$ operations. We have extended the basic $insert$ operation in Chord to support the updating of signatures in the index nodes and enable concurrent operations such as node splitting in a decentralized fashion. Each index node has a header that determines the id of the new node that is created when it splits, and enables implicit synchronization among peers during index node splitting.

The Document Locator issues DHT $lookup$ operations to fetch the index nodes. For *H-index* and *tH-index*, the signatures in the node entries of a fetch node are tested for divisibility with the query signature to determine further traversal of the index. For *nH-index*, the query interval is tested with the intervals in the node entries according to the operator in the value predicate. Given an XPath query, if value predicates are absent, then the *H-index* corresponding to the *target node* in the query is searched. Otherwise, an appropriate *tH-index* or *nH-index* is selected for searching. Picking the best index is a future research challenge.

D. Coping with Failures

On the Internet, *failures are rather a norm than an exception.* Chord stores a key-value pair on k closest successors to the key in the identifier ring [9]. When a peer leaves or fails, the key that it is responsible for is available from its successor. Chord uses UDP instead of TCP connections for RPC calls performed during key lookup and insert. This is because TCP is expensive for the nature of communication patterns in the DHT. As a result, RPC errors can occur. In Chord, if a minimum number of replicas are stored successfully, then the $insert$ operation succeeds. On a lookup, the closest replica is fetched. (Chord uses Vivaldi [15] for network coordinates to determine the closest peer.) However, this scheme is troublesome for *psi*X because the index node replicas become inconsistent due to RPC errors and each peer may use a different replica during publishing.

To suit our requirements, we modified Chord's insert scheme as follows. First, the insert operation is issued on the index node stored at the immediate successor for the node id. (We will call this the *primary copy*.) If it is successful, then the other replicas (or *secondary copies*) are updated, for which failures may occur. If a failure occurs while updating the primary copy, then the insert operation is forced to fail. The Publisher retries after waiting for some time interval. The primary copy of an index node is used to traverse the index (via lookup operations) during publishing. Lookup failures can occur during publishing and while locating documents. These are handled via retries.

IV. Demonstration Scenario

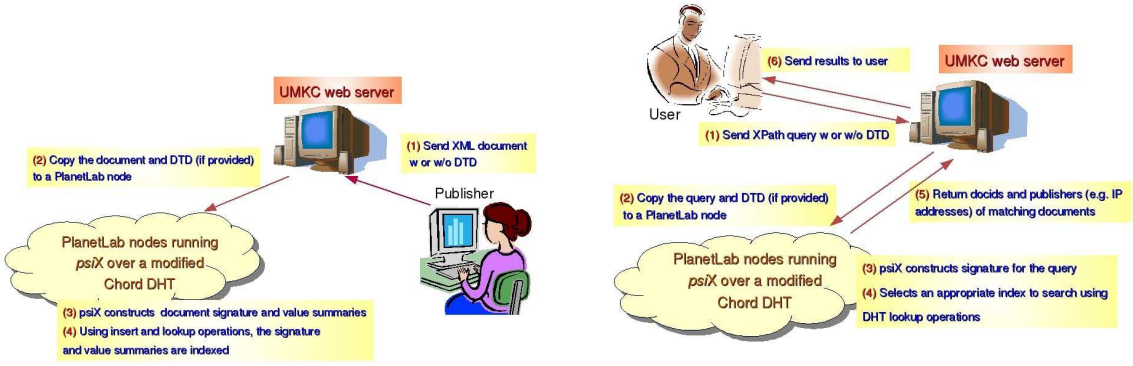
We have setup a web interface [16] through which anybody can access the services of *psi*X. Since direct access to PlanetLab nodes requires prior registration and authorization, a broker (*i.e.*, web server) is introduced between a user and PlanetLab nodes. (Several brokers can be setup to avoid overloading and failures.) These nodes are remotely controlled using PlMan [17]. New nodes can be added or existing nodes can be removed at any time.

A. Publishing XML Documents

Figure 3(a) shows the flow of control and data when a user publishes an XML document. A user submits a document with or without DTD through an HTML form shown in Figure 3(c). (If DTD is not provided, the signature is constructed without resorting to the SSG, although the SSG can be constructed by scanning the document once.) The user also specifies a global document id, which can be any unique string to identify the publisher (*e.g.*, URL, IP address) and a local docid. The web server copies the inputs to a randomly selected PlanetLab node. The input is processed by *psi*X and the user is returned a log file that shows the status of the publishing process.

B. Locating Relevant XML Documents using XPath

Figure 3(b) shows the flow of control and data when a user poses an XPath query to locate relevant XML documents. A user inputs a query (and a DTD if available) through an HTML form shown in Figure 3(d). (If the DTD is not provided, then the signature is constructed by ignoring the SSG.) The user can indicate whether the *nearest replica* or the *primary copy* of



(a) Flow of control and data during publishing

(b) Flow of control and data while locating documents

(c) HTML form for publishing an XML document

(d) HTML form for specifying an XPath query

Fig. 3. Setup of *psi*X for demonstration purposes

TABLE I

QUERY PERFORMANCE

XPath query	# of docs found	Total # of hops	Time taken
//Actor/Name	65	6	0.395 secs
//Movies[Title][Year]	122	5	0.30 secs
/Frame/Colors[@outline_colors =“defaultCDATA1”]	54	6	0.251 secs
/dblp/article[volume>“1”]	3	4	0.166 secs
/table/*/L_COMMENT	101	6	0.247 secs

the index nodes should be used for searching. The web server copies the user inputs to a randomly selected PlanetLab node running *psi*X. The user input is processed, and an appropriate hierarchical index is chosen. The global docids of the matched documents are returned to the web server and passed back to the user. The user can then request the documents from the publishers directly.

To provide insight into the performance of *psi*X, we published 2,455 XML documents into a P2P network setup using 29 PlanetLab nodes each running a Chord peer with 4 virtual peers. The cost of locating relevant documents in terms of the total time taken and the number of hops required are reported in Table I.

V. CONCLUSIONS

In this demonstration, we present *psi*X, an Internet-scale service for publishing and locating XML documents. The *psi*X system adopts a novel signature scheme to perform holistic XML pattern matching to avoid excess network traffic in a P2P network. Distributed and balanced hierarchical indexes are built for document signatures and value summaries using a DHT. The indexes allow concurrent write operations from peers in a decentralized fashion. Failures are gracefully handled by *psi*X while operating in an Internet environment.

REFERENCES

[1] “Napster,” http://www.napster.com.
[2] “Kazaa,” http://www.kazaa.com.
[3] L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt, “Locating Data Sources in Large Distributed Systems,” in *Proc. of the 29th VLDB Conference*, Berlin, 2003.
[4] C. Sartiani, P. Manghi, G. Ghelli, and G. Conforti, “XPeer: A Self-Organizing XML P2P Database System,” in *Intl. Workshop on Peer-to-Peer Computing and Databases*, Greece, 2004.
[5] G. Koloniari and E. Pitoura, “Peer-to-Peer Management of XML Data: Issues and Research Challenges,” *SIGMOD Record*, vol. 34, no. 2, pp. 6–17, June 2005.
[6] G. Skobeltsyn, M. Hauswirth, and K. Aberer, “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.
[7] Y. Zhang and P. A. Boncz, “Integrating XQuery and P2P in MonetDB/XQuery,” in *Proceedings of the 1st Workshop on Emerging Research Opportunities for Web Data Management*, Barcelona, Jan. 2007.
[8] S. Abiteboul, I. Manolescu, N. Polyzotis, N. Preda, and C. Sun, “XML Processing in DHT Networks,” in *Proc. of the 24th IEEE Intl. Conference on Data Engineering*, Cancun, Mexico, Apr. 2008.
[9] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications,” in *Proc. of the 2001 ACM-SIGCOMM Conference*, San Diego, CA, Aug. 2001, pp. 149–160.
[10] “PlanetLab,” http://www.planet-lab.org.
[11] Georgia Koloniari and Evaggelia Pitoura, “Content-based routing of path queries in peer-to-peer systems,” in *Proc. of the 9th Intl. Conference on Extending Database Technology*, Crete, Greece, 2004, pp. 29–47.
[12] P. Rao and B. Moon, “Locating XML Documents in Peer-to-Peer Networks using DHTs,” UMKC, Tech. Rep. TR-DB-2008-01, Mar. 2008, http://r.faculty.umkc.edu/raopr/TR-DB-2008-01.pdf.
[13] P. Rao, “Building an Internet-Scale Service for Publishing and Locating XML Documents on PlanetLab,” UMKC, Tech. Rep. TR-DB-2008-02, June 2008, http://r.faculty.umkc.edu/raopr/TR-DB-2008-02.pdf.
[14] A. Guttman, “R-trees: A Dynamic Index Structure for Spatial Searching,” in *Proc. of the 1984 ACM-SIGMOD Conference*, Boston, MA, June 1984, pp. 47–57.
[15] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris, “Practical, distributed network coordinates,” *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 113–118, 2004.
[16] “*psi*X,” http://vortex.sce.umkc.edu/psix.
[17] “PlanetLab Experiment Manager,” http://www.cs.washington.edu/research/networking/cplane/.