SEARCHING AND RANKING XML DATA IN A DISTRIBUTED ENVIRONMENT

by

WEIMIN HE

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2008

To my wife Lotus and my daughter Jocelyn who constantly supported me for my doctoral studies.

ACKNOWLEDGEMENTS

Upon the successful completion of my dissertation, the first person I would like to thank is my supervising professor Dr. Leonidas Feagras. I would like to thank Dr. Fegaras from the bottom of my heart, for his full support, constant motivation and invaluable advice during the course of my doctoral studies. His serious academic attitude set an example for me in my research career. I would also like to extend my appreciation to Dr. Gautam Das, for sharing his excellent and unique research experience with me, which cast light on my research. I wish to thank Mr. Levine, for his seminal contributions that formed some of the foundations of my dissertation. I would also like to thank my academic advisors Dr. Ramez Elmasri and Dr. Chengkai Li for their interest in my research and for taking time to serve in my dissertation committee.

I wish to thank Dr. Filla Makedon and Dr. Bob Weems for providing constant financial support from the department for my doctoral studies. I am grateful to all the teachers who taught me during the years I spent in school, in China, and in the Unites States. Especially, I wish to thank my MS advisor, professor Tong Li, and professor Weiyi Liu, who brought me to the right track of research. I also want to express my appreciation to my colleagues at our lab, Cahty Wang, Ranjan Dash, Anthony Okorodudu, Jack Fu, Feng Ji, Kyungseo Park, Jae Sung Choi, Byoungyong Lee, and Kamal Taha, for their good advice and comments on my research.

Finally, I would like to express my deep gratitude to my wife, Lotus, for her persistent and self-giving contributions to my family, and also for her relentless encouragement and inspiration during the course of my doctoral studies. I am also extremely grateful to my mother, father, brother and sister for their sacrifice, encouragement and patience.

November 24, 2008

ABSTRACT

SEARCHING AND RANKING XML DATA IN A DISTRIBUTED ENVIRONMENT

Weimin He, Ph.D.

The University of Texas at Arlington, 2008

Supervising Professor: Leonidas Fegaras

Due to the increasing number of independent data providers on the web, there is a growing number of web applications that require searching and querying data sources distributed at different locations over the internet. Since XML is rapidly gaining in popularity as a universal data format for data exchange and integration, locating and ranking distributed XML data on the web are gaining importance in the database community. Most of existing XML indexing techniques combine structure indexes and inverted lists extracted from XML documents to fully evaluate a full-text query against these indexes and return the actual XML fragments of the query answer. In general, these approaches are well-suited for a centralized date repository since they perform costly containment joins over long inverted lists in order to evaluate full-text XML queries, which does not scale very well to large distributed systems.

In this thesis work, we present a novel framework for indexing, locating and ranking schema-less XML documents based on concise summaries of their structural and textual content. Instead of indexing each single element or term in a document, we extract a structural summary and a small number of data synopses from the document, which are indexed in a way suitable for query evaluation. The search query language used in our framework is XPath extended with full-text search. We introduce a novel data synopsis structure to summarize the textual content of an XML document that correlates textual with positional information in a way that improves query precision. In addition, we present a two-phase containment filtering algorithm based on these synopses that speeds up the searching process. To return a ranked list of answers, we integrate an effective aggregated document ranking scheme into the query evaluation, inspired by *TF*IDF* ranking and term proximity, to score documents and return a ranked list of document locations to the client. Finally, we extend our framework to apply to structured peer-to-peer systems, routing a full-text XML query from peer to peer, collecting relevant documents along the way, and returning list of document locations to the user. We conduct many experiments over XML benchmark data to demonstrate the advantages of our indexing scheme, the query precision improvement of our data synopses, the efficiency of the optimization algorithm, the effectiveness of our ranking scheme and the scalability of our framework.

We expect that the framework developed in this thesis will serve as an infrastructure for collaborative work environments within public web communities that share data and resources. The best candidates to benefit from our framework are collaborative applications that host on-line repositories of data and operate on a very large scale. Furthermore, good candidates are those applications that seek high system and data availability and scalability to the network growth. Finally, our framework can also benefit to those applications that require complex/hierarchical data, such as scientific data, schema flexibility, and complex querying capabilities, including full-text search and approximate matching.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In recent years, the popularity of the internet and the growing need to share the vast amount of data on the web, has fueled the need for a new web data format that will make data sharing and integration feasible. Information in traditional web pages encoded in HTML (Hyper Text Markup Language) format is usually hard to interoperate and exchange because the tags in HTML documents describe the presentation of data instead of the semantics of data. As such, a new standard was required to encode web data in a simple and usable format so that information providers can interoperate easily over heterogeneous data distributed on the web.

In order to facilitate the sharing of structured data across different information systems over the internet, a new open standard for web data representation and exchange, XML (eXtensible Markup Language), has been recommended and adopted as the new generation web data format. XML started strong and has grown quite rapidly. It has proven itself a very valuable technology, which "turns the web into a database" and allows data integration on the web. In fact, most data exchanged among a variety of web applications are already in XML format, such as web services that use XML-based descriptions in WSDL and exchange XML messages based on the SOAP protocol, e-commerce and e-business, collaborative authoring of large electronic documents and management of large-scale network directories. As a flexible and self-describing semi-structured data format, XML holds the promise to yield (1) a more precise search by providing additional information in the elements, (2) a better integrated search of documents from heterogeneous sources, (3) a powerful search paradigm using structural as well as content specifications, and (4) data and information exchange to share resources and to support cooperative search [87].

To query the vast amount of data on the web, the most common approach is to exploit some well-known search engines, such as Google, and issue a keyword query to retrieve a ranked list of document links that are relevant to the keyword query. In fact, keyword search querying has emerged as one of the most effective paradigms for information discovery, especially over HTML documents in the World Wide Web. One of the key advantages of keyword search querying is its simplicity. Users do not have to learn a complex query language and can issue queries without any prior knowledge about the structure of the underlying data. However, since the keyword search query interface is very flexible, queries may not be precise and can potentially return a large number of query results, especially in large document collections [17]. Despite the success of HTML-based keyword search engines such as Google, certain limitations of the HTML data model make such systems ineffective in some domains. These limitations stem from the fact that HTML is a presentation language and hence cannot capture much semantics. The XML data model addresses this limitation by allowing for extensible element tags, which can be arbitrarily nested to capture additional semantics [17].

Given the rich, self-describing structural content of XML, it is natural to exploit this information for more precise XML data retrieval. One approach is to employ sophisticated query languages, such as XQuery [3], to query XML documents. While this approach can achieve higher precision than keyword search querying, it requires the user to learn a complex query language and to know the schema of underlying XML data. An alternative approach is to retain the simple keyword search query interface, but exploit the XML's tagged and nested structure during query processing [17]. Although this approach can provide more precise query answers at the granularity of element instead of document to the user, it may still return a large number of false positives in the query answers because simple keyword queries can not enforce the containment relationships between text nodes and elements in XML documents.

In addition, due to the increasing number of independent data providers on the web, there is a growing number of web applications that require locating data sources distributed over the internet. These new web applications, such as file sharing, instant messaging and collaborative computing, have brought up the popularity of a new computing model called the peer-to-peer (P2P) model. In a P2P network, a large number of nodes (computers connected to the internet) operate together to share data and resources with each other on an equal basis. An important feature of P2P networks is that all nodes provide resources, including bandwidth, storage space, and computing power. Thus, as nodes arrive and demand on the system increases, the total capacity of the system also increases. In a distributed environment, especially P2P systems, different users and applications may employ various formats and schemas to describe their data. Moreover, some application domains, such as health-related applications, use sensitive data that are required not to be exposed to all users for privacy reasons. Therefore, there is a need for a query language that can work with incomplete or no-schema knowledge but also capture whatever semantic knowledge is available. The flexibility of XML in representing heterogeneous data makes it suitable for distributed applications and P2P networks, where data are either native XML documents or XML mappings of data or services that are represented in various format in the underlying sources [1]. As such, querying and ranking distributed XML data has attracted much interests in the literature. Searching and ranking of distributed XML data, especially in P2P environments, pose several challenges. First, efficient indexing of XML data becomes more difficult in a distributed environment. Full indexing of web-accessible XML data in a distributed environment is neither feasible or desirable because the distributed query evaluation may cause the transmission of very long partial intermediate results between different peers, which can significantly degrade the query response time and increase the network traffic. As such, the data indexing scheme must be designed very carefully to be efficient in terms of both storage space and query response time. More importantly, XML data must be indexed in a desirable way to minimize the cost of index updates. Second, query processing and ranking over hierarchical and nested XML data become more complex in a distributed environment. Efficient distributed query plans must be developed to collect relevant XML data on different nodes and to route the final query answers to the query client. In addition, developing an effective ranking scheme in a distributed environment for XML data, especially for XML meta-data, is also a challenging work.

In this thesis work, we develop a framework for efficient indexing, querying and ranking schema-less XML data in a distributed environment. Our data indexing scheme is based on very compact meta-data extracted from original XML documents. Our query processing algorithms are novel in the sense that we introduce a novel operation called containment filtering during the query evaluation, which can enforce the structural constraints in the query, thus significantly improving the query precision. Our ranking scheme extends traditional TF*IDF scoring used in IR to rank XML documents based on data synopses, which can effectively rank the most relevant documents. Finally, we extend our framework to structured P2P networks and develop effective data indexing and query processing algorithms in DHT (Distributed Hash Table) networks. We conduct extensive experiments to validate the effectiveness, feasibility, and scalability of our framework.

In this section, we first give a brief background introduction to the XML data format and P2P networks. Then, we present the motivation behind this thesis work and overview our key approaches. We also show a potential application example based on our framework and summarize the broad impact of our framework.

1.1 Introduction to XML

XML is a general-purpose specification for creating custom markup languages. It started as a simplified subset of SGML (Standard Generalized Markup Language), and is designed to be relatively human-legible. Different from HTML, in which tags associated with data express the presentation style of data, XML allows its users to define their own tags to identify the meaning of the data. The relationships among data elements are provided via simple nesting and references. XML encoding provides the information in a far more convenient and usable format from the data management perspective. Due to its inherent data self-describing capability and flexibility of organizing data, XML has evolved as the defacto standard for information exchange among various applications on the internet.

1.2 XML Query Languages

The role of user-defined nested tags in XML is somewhat similar to that of schemas in relational databases. The self-describing nature of XML makes it easy to query without the need of a schema. Therefore, many query languages have been proposed and developed to query XML data. Two mainstream XML query languages are XPath [2] and XQuery [3], developed and recommended by W3C.

XPath is a path language for selecting nodes from an XML document such that the path from the root to each selected node satisfies the pattern specified in the query. XPath is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria. An

XPath expression consists of a sequence of location steps. Each location step has three components: an axis, a node test, and a predicate. An XPath expression is evaluated with respect to a context node. An axis specifier, such as “child” or “descendant”, specifies the direction to navigate from the context node. The node test and the predicate are used to filter the nodes specified by the axis specifier. A predicate is used to filter the selected nodes based on certain properties, which are specified by XPath expressions.

XQuery is a powerful XML query language and is more complex and expressive than XPath. An XQuery expression can contain multiple FLWOR (For-Let-Where-Order-Return) clauses and each clause in itself can include sub-XQuery queries. The For and Let clauses bind nodes selected by XPah expressions to user-defined node variables. The Where clauses specify selection or join conditions on node variables. The Return clauses operate on node variables to format query results in XML format. Although the nested and compositional syntax of XQuery makes it much more expressive than XPath, its rich semantics also significantly increases its optimization and evaluation complexity. Although our work considers XPath only, it can be extended to cove XQuery.

1.3 Peer-to-Peer Computing

In the past few years, the peer-to-peer (P2P) model has emerged as a new and popular computing model for many web applications, such as file sharing [4, 5, 6, 7], collaborative computing and instant messaging. A P2P network consists of a large number of nodes, called peers, that share data and resources with each other on an equal basis. Peers are connected through a logical network topology implemented on top of an existing physical network, which may dynamically adapt to cope with peers joining and departing. A node in a P2P network can act as both a service provider and a client. Compared to traditional client-server systems, P2P systems are more scalable [8], flexible, fault-tolerant, and easy to deploy. More importantly, network resources can be fully utilized and shared, and the server workload can be distributed among all the peers in the system. By leveraging vast amounts of computing power, storage, and connectivity from personal computers distributed around the world, P2P systems provide a substrate for a variety of applications, such as network monitoring, web search, and large scale event/notification systems.

In general, P2P systems can be classified into two categories: unstructured P2P and structured P2P [9]. In an unstructured P2P system, each peer publishes its data locally in a specific sharing directory. A peer can query data by sending a query message that can be routed to the data owner peer through intermediate peers. In such a system, there is no global protocol for the data placement and searching, and the network topology is not tightly controlled. In a structured P2P system [10, 11, 12, 13], the location of data is determined by some global scheme, such as a global hash function that stores a data object to a node based on a search key. In such a system, peers form a virtual distributed hash table (DHT) and a query can be routed to the destination peer more efficiently. Unstructured P2P systems have many advantages, such as easy-deployment, more flexibility, and low costs of maintenance. However, due to their inherent unstructured nature, unstructured P2P systems usually have poor search performance and are not scalable because the query load of each node grows linearly with the total number of queries, which in turn grows with the number of nodes in the system. In contrast, structured P2P systems have high search efficiency and are more scalable. The routing time for a query message is only log(N), where N is the number of peers in the system. Moreover, their DHT-based data placement strategy naturally leads to load balancing in the system. However, structured P2P systems can only support exact-match queries because of their use of hashing mechanisms for data location.

1.4 Motivation of this Thesis Work

As XML has become the *de facto* form for representing and exchanging data on the web, there is an increasing interest in indexing, querying, and ranking XML documents. Although approximate matching with relevance ranking for plain documents has long been the focus of Information Retrieval (IR), the hierarchical nature of XML data has brought new challenges to both the IR and database communities. Until recently, XML query languages, such as XPath [2] and XQuery [3], were very powerful in expressing exact queries over XML data, but they did not meet the needs of the IR community since they were lacking of full-text capabilities. This has changed recently and there is now an emerging standard for full-text search on XML [14], which extends the syntax and semantics of XQuery and XPath with full-text search capabilities. With these syntactic extensions, one can specify queries to search for XML documents based on partial information on both the structure and the content of the documents. These queries are potentially more precise than simple IR-style keyword-based queries, not only because each search keyword can be associated with a structural context, which is typically the path to reach the keyword in a document, but structural constraints can also be used to specify the structural relationships among multiple search keywords.

Consider, for example, the full-text search query

```
//biblio/publisher[name = "Wiley"]
        //book[author/lastname = "Smith"]
               [title contains "XML" and "SAX"]/price
```

against a pool of indexed XML documents. It searches for all books in biblio documents published by Wiley and authored by Smith that contain the words “XML” and “SAX” in their titles. When searching for documents that satisfy this query, we do not want to waste any time by considering those that do not match the structural constraints of the query or those that do not contain the search keywords at relative positions as specified by the structural relationships in the query. For example, we do not want to consider a document that, although has books authored by Smith, none of these books has both “XML” and “SAX” in their titles, even though there may be other books not authored by Smith with both these keywords in their titles.

Based on the above observations, it is clear that, when indexing an XML document to make it available for searching and querying, each keyword in the document must be indexed and encoded in a substantial amount of detail to help us decide whether a set of keywords in a document satisfy the structural/containment constraints in a query. Current XML indexing techniques [15] combine structure indexes and inverted lists extracted from XML documents to fully evaluate a full-text query against these indexes and return the actual XML fragments of the query answer. This is typically accomplished by performing containment joins over the sorted inverted lists derived from the element and keyword indexes. Since all elements and keywords have to be indexed, such indexing schemes may consume a considerable amount of disk space and may be time-consuming to build. More importantly, the query evaluation based on these indexes may involve many joins against very long inverted lists that may consider many irrelevant documents at the early stages. Although many sophisticated techniques have been proposed to improve these joins by skipping the irrelevant parts of these lists, it is still an open research problem to make them effective for a large document pool, especially for large scale distributed systems, such as peer-to-peer systems.

1.5 Our Approach

In this thesis, we present a novel framework for efficient indexing, locating and ranking schema-less XML documents based on condensed summaries extracted from the structural and textual content of the documents. Instead of indexing each single element or term in a document, we extract a structural summary and a small number of data synopses from the document, which are indexed in a way suitable for query evaluation. The result of the query evaluation is a ranked list of document descriptions that best match the query, based on an aggregated score derived from both content similarity and term proximity. A document description consists of meta information about the document, such as the document URL, structural summary, and description. Based on the retrieved meta information, the client can choose some of the returned document locations and request a full evaluation of the query over the chosen documents using any existing XML query engine and return the XML fragments of the query answers.

To find all indexed documents that match the structural relationships in a query, the *query footprint* is extracted from the query and is converted into a pipelined plan to be evaluated against the indexed structural summaries. The resulting documents that match the query footprint are further filtered out using the data synopses associated with the search predicates in the query and the qualified document locations are returned to the client. Instead of just returning the intersection of all documents that satisfy each search predicate separately, we take into account the containment relationships among the search predicates incorporated into the positional dimension of our data synopses, resulting in a more accurate evaluation of textual and containment constraints in a query when compared to regular one-dimensional Bloom filters [16]. In order to avoid the long join lists during the query evaluation, we also use a two-phase containment filtering algorithm to prune the unqualified document locations before the actual join operations and thus reduce the query response time. In addition, we integrate into the query evaluation an effective aggregated ranking scheme that combines *TF*IDF* costing and term proximity to score the documents and return a ranked list of document locations to the user. Finally, we extend our framework to structured peer-to-peer systems by indexing data synopses in a distributed hash table (DHT), evaluating the user query in a distributed fashion, and collecting the answers along the way from peer to peer.

1.6 Our Contributions

In summary, we make the following contributions in this thesis:

- We present a novel framework for indexing, querying, and ranking XML documents based on content and structure synopses, that is suitable for full-text XPath evaluation.
- We present a two-phase containment filtering algorithm based on our data synopses that improves the searching process.
- We introduce an effective aggregated ranking scheme to score an XML document based on our data synopses.
- We extend our framework to structured P2P networks and develop distributed data indexing and query processing algorithms.
- We experimentally validate the advantages of our indexing scheme, the query precision improvement of our data synopses, the efficiency of the optimization algorithm, the effectiveness of our ranking scheme, and the scalability of our framework.

1.7 Broad Impact

We expect that the framework presented in this thesis will serve as an infrastructure for collaborative work environments within public web communities that share data and resources. The best candidates to benefit from our framework are collaborative applications that host on-line repositories of data and operate on a very large scale (thousands of nodes, massive data, very frequent queries, and moderately frequent updates). Furthermore, good candidates are those applications that seek high system and data availability and scalability to the network growth. Finally, our framework can also benefit to those applications that require complex/hierarchical data, such as scientific data, schema flexibility (semi-structured data), and complex querying capabilities, including full-text search and approximate matching.

As an application example of our framework, consider a web search engine for biological data repositories. Nowadays, biologists would like to share biological data with their colleagues in different institutions. Most biological repositories provide only a simple keyword search mechanism over huge biological data sets to return all the documents that satisfy the keyword queries. If the query contains some popular keywords, the user may have to tolerate a long response time or may just run out of memory because of the large number of document answers returned. Using our XML search framework, we can extract biological meta-data from different data sources and publish them on a centralized server or distributed on a P2P network. The actual XML data remain on the remote data sources. Our system will provide a navigational tool, which shows the schema summary of data and helps a user pose a query, which is translated into a full-text XPath query. The query results are a ranked list of document locations that satisfy the query. A biologist can choose top document hits and click on the link to the document owner to get the actual XML fragments as query answers.

1.8 Layout of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, a compilation of the related work in the area of XML data management is provided. More specifically, keyword search over XML, semantic search over XML, peer-to-peer data management for XML, and XML summarization techniques are presented. In Chapter 3, we describe our query language and overview our system architecture. In Chapter 4, we introduce our meta-data indexing scheme. In Chapter 5, the query processing in our framework is presented. In particular, a novel key operation called *Containing Filtering* is introduced to filter out unqualified document locations. In Chapter 6, we present the two-phase containment filtering algorithm to further reduce query processing overhead. In Chapter 7, we discuss the relevance ranking scheme in our system. Finally,we summarize our thesis and envision the future work in Chapter 8.

CHAPTER 2

RELATED WORK

With the popularity of XML as the universal data format for a wide range of web data repositories, extensive research work has been done on designing powerful query languages, developing efficient indexing and query evaluation algorithms, and proposing effective ranking schemes over XML data. In addition, due to the emergence of a variety of P2P applications, locating and querying distributed XML data sources in unstructured and structured P2P networks has also attracted much attention in the literature. In this chapter, a comprehensive overview of the research work related to this dissertation is presented.

2.1 Keyword Search over XML

One method for searching XML data on the web, is keyword search, which borrows ideas from the traditional IR community [17, 18, 19]. A user query is typically a set of keywords and the query answer is a ranked list of relevant XML fragments, each of which contains all the keywords in the query. The advantages of this paradigm are the following. First, the query mechanism is relatively simple and there is no need for the user to learn the complex syntax of XML query languages. Second, the user does not have to know the schema of the data before he/she can issue a keyword query. In fact, keyword search provides a simple and user-friendly query interface to retrieve XML data in a variety of web and scientific applications, where users may not know XPath/XQuery, or the schema of data is unavailable or change frequently. The major drawback of keyword search is that it may not be as precise as if the query had also specified the position of the keywords in the nested XML structure. Keyword search over XML introduces many challenges. First, the result of the keyword search query can be a deeply nested XML element instead of the entire document. Second, ranking has to be done at the granularity of XML elements instead of entire XML documents, which is more complicated due to the nested nature of XML data. Therefore, keyword search for XML has recently attracted much interests.

In [17], the authors propose the XRANK system that can efficiently produce ranked results for keyword search queries over hierarchical and hyperlinked XML documents. XRank is the first system that takes into account both the hierarchical an hyperlinked structure of XML documents, and a two-dimensional notion of keyword proximity, when considering the ranking for XML keyword search queries. The authors first adapt the algorithm for computing PageRanks of HTML documents for use with XML documents by mapping each element to a document, and by mapping all edges (IDREF, XLink and containment edges) to hyperlink edges. Based on the algorithm, they define ElemRank, which is a measure of the objective importance of an XML element. Then, in order to reduce the space overhead and spurious query results caused by the naive inverted-list-based indexing, the authors employ Dewey IDs to encode XML elements, which jointly captures ancestor and descendant information in an XML document. An interesting feature of Dewey IDs is that the ID of an ancestor is a prefix of the ID of a descendant. Therefore, ancestor-descendant relationships are implicitly captured in the Dewey ID. The inverted list for a keyword k contains the Dewey IDs of all the XML elements that directly contain the keyword k. To handle multiple documents, the first component of each Dewey ID is the document ID. Associated with each Dewey ID entry is the ElemRank of the corresponding XML element, and the list of positions where the keyword k appears in that element [17]. Equipped with Dewey ID-based inverted list index structures, The authors develop an efficient algorithm for scoring XML elements that takes into account both hyperlink and containment edges.

The XKSearch [18] system takes a list of keywords and returns the set of Smallest Lowest Common Ancestor (SLCA) nodes, i.e. the set of smallest trees containing all keywords, to the user. Compared with XRank, XKSearch improves the precision of query results by considering only the “smallest” XML subtree as a query answer if it contains no tree that also contains all keywords. According to the SLCA semantics, the result of a keyword query is the set of nodes that must satisfy the following two conditions: (i) each node must contain all the keywords in the query either directly or indirectly(in descendant nodes); (ii) each node has no descendant node that also contains all keywords. For each keyword the system maintains a list of nodes that contain the keyword, in the form of a tree sorted by the id’s of the nodes. The key property of SLCA search is that, given two keywords k_1 and k_2 and a node v that contains keyword k_1, one need not inspect the whole node list of keyword k_2 in order to discover potential solutions [18]. Instead, one only needs to find the left and right match of v in the list of $k2$, where the left (right) match is the node with the greatest (least) id that is smaller (greater) than or equal to the id of v. Based on the above key property, the authors propose two efficient algorithms, termed Indexed Lookup Eager Algorithm and Scan Eager Algorithm. Both algorithms produce part of the answers very quickly so that users do not have to wait long to see the first few answers. Their core contribution, the Indexed Lookup Eager algorithm, exploits key properties of smallest trees in order to outperform prior algorithms by orders of magnitude when the query contains keywords with significantly different frequencies. The Scan Eager algorithm is turned for the case where the keywords have similar frequencies [18].

2.2 Semantic Search over XML

In general, keyword search approaches suffer from two drawbacks: (i) they do not distinguish tag names from textual content; (ii) they can not express complex query semantics. Keyword-based XML search queries do not explicitly refer to the tags in an XML document, and thus they can not incorporate semantic knowledge in a precise way. In other words, they do not enforce the containment relationships between the keywords and tags in the query. To address this limitation, an alternative paradigm for XML search on the web, termed semantic search, has been proposed. A search query for a semantic search can be a set of simple tag-term pairs, such as *(author:Ullman, title:database)*, which enforces the containment relationship between a term and a tag. The semantics of this query is that the term "Ullman" must be in a tag *author*, and the term "database" must be in a tag *title*. Although the semantic search based on tag-term pairs is more precise than the simple keyword search, it is still not precise enough to capture the complex containment relationships among different tags in the query. A semantic search query can also be a complex XML query expressed in full-fledged XML query languages extended with full-text search functionalities, such as XQuery. The major advantage of this paradigm is that the query answers are potentially more precise than simple keyword search. Its main drawback is that it requires the user to partially know the schema in order to issue an effective query.

In [20], the authors propose XSEarch, which is a semantic search engine for XML. XSEarch can return semantically related XML document fragments that satisfy the query to the user. It allows the user to specify labels and keyword-label combinations that must or may appear in a satisfying document. A search term has the form $l : k$, $l :$ or $: k$ where l is a label and k is a keyword. A search term may have a plus sign prepended, in which case it is a required term. Otherwise, it is an optional term. In order to satisfy a query Q, each of the required terms in Q must be satisfied. In addition, the elements satisfying Q must be meaningfully related. XSEarch assumes that there is a given relationship R that determines when two nodes are meaningfully related. The authors define a natural relationship, called interconnection relationship, and use it in their working system. In order to rank query answers, the authors also extend the traditional $tf * idf$ scoring of IR to rank the list of returned XML fragments. They compute the weight of a keyword (also called a term) k in a given leaf node n_l using a variation of the standard $tf * idf$ formula. Each label l is associated with a weight $w(l)$ that determines its importance. The label weights can be either user defined or system generated. Based on the weights of keywords, the authors employ vector space model to determine how well an answer satisfies the user query. Although XSEarch can enforce single containment relationships between keywords and tags in the query, it can not enforce the complex containment constraints among different search predicates in the query, and thus may degrade the query precision.

In [21], the authors propose a bulk-algegra, TIX, that permits the integration of IR style query processing into a traditional pipelined query evaluator for an XML database. The major advances in TIX include (i) the ability to manage relevance scores, including score generation, manipulation, and use; and (ii) facilities for management of result granularity. The authors introduce two new operators, **Threshold** and **Pick** for the relevance ranking during the query evaluation. The **Threshold** operator is very similar to the selection operator and it can simplify the expression of irrelevance filtering, which is necessary for an algebra targeting IR. The **Pick** operator is the key operator that removes the redundancy in the returned results for an IR-style query. The **Pick** operator is quite different from projection in that projection only needs information local to the node being projected (e.g., the tag name), while

Pick needs information that may reside elsewhere in the data tree (e.g., the ancestor nodes). The authors also develop new evaluation strategies for efficiently scoring composite elements [21]. The proposed two algorithms, **TermJoin** and **PhraseFinder**, can effectively implement the score generation by using a stack-based approach.

In [22], the authors propose a novel concept termed query relaxation, to address the mismatch between the approximate matching based keyword search queries and the exact match based XPath queries. They consider queries on structure as a template, and they look for answers that best match this template and the full-text search. To achieve this, they provide an elegant definition of relaxation on structure and define primitive operators to span the space of relaxations. Their query answering is based on ranking potential answers on structural and full-text search conditions. The authors set out certain desirable principles for ranking schemes and propose natural ranking schemes that adhere to these principles. They also develop efficient algorithms for answering top-K queries and discuss results from a comprehensive set of experiments that demonstrate the utility and scalability of the proposed framework and algorithms [22].

2.3 XML Summarization Techniques

Since our querying and ranking schemes are based on XML meta-data, our work is also related to XML summarization techniques. Polyzotis *et al* [23] propose an XSKETCH synopsis model that exploits localized stability and value-distribution summaries to accurately capture the complex correlation patterns that exist between and across path structure and element values in the XML data graph. In [24], the authors further propose a novel class of XML synopses, termed XCLUSTERs, that addresses the key problem of XML summarization in the context of heterogeneous value content. Similarly, [25] presents a novel data structure, the *bloom histogram*, to approximate XML path frequency distribution within a small space budget and to accurately estimate the path selectivity. A *dynamic summary* layer is used to keep exact or more detailed XML path information to accommodate frequently changed data. However, all these proposed XML synopses and summaries are mainly used for selectivity estimation, rather than for locating and ranking XML documents, as is in our framework.

Another work by Cho *et al* [26] addresses the meta-data indexing problem of efficiently identifying XML elements along each location step in an XPath query that satisfy range constraints on the ordered meta-data. The authors develop a meta-data index structure named *full meta-data index* (FMI) by applying the R-tree index for XPath location steps. The FMI can quickly identify XML elements that are reachable from a given element using a specified XPath axis and satisfy the meta-data range constraints. To reduce the meta-data update overhead, another R-tree-based index structure, named *inheritance meta-data index* (IMI), is also proposed to enable efficient lookup in the index structure, while keeping the update cost manageable.

2.4 Peer-to-Peer Data Management over XML

P2P provides a good platform for exchanging a large amount of data on the web. Over the past few years, the popularity of P2P applications and XML has brought many researchers on investigating the problem of indexing and querying distributed XML data sources in P2P networks [27, 28, 29, 30, 31, 32, 33].

2.4.1 XML Data Management in Unstructured P2P Systems

Some researchers are interested in querying XML data in unstructured P2P networks. Research efforts focus primarily on building space efficient routing indexes for XML documents. Most approaches build path indexes with the use of aggregation and suitable encoding schemes for the paths.

DBGlobe [34] is a project that aims at building a service-oriented P2P system for global computing. Service-oriented approach can resolve heterogeneity and semantic mismatch problems caused by heterogeneous peers in the system. Each peer publishes its data through services and accesses data on other peers by invoking a service. Direct querying of data is also supported by defining services that employ an XML-based query language[34]. DBGlobe differentiates roles of peers in the system by defining a small number of Cell Administration Servers(CASs) and a large number of Primary Mobile Objects(PMOs). To efficiently route path queries for XML data, multi-level bloom filters are employed to route a query to neighbors that potentially store relevant data. DBGlobe also explores querying XML data by embedding some service calls in XML documents. These service calls can be activated at certain times and the results can be returned to the users.

In [31], the authors proposed two multi-level Bloom filters, termed Breath Bloom Filter and Depth Bloom Filter, to summarize the structure of an XML document for efficient path query routing in unstructured P2P networks. In their framework, each peer maintains a local index to summarize its local XML data and one or more merged indexes to summarize the XML data of its neighbors. They also advocate building a hierarchical organization of nodes by clustering together nodes with similar content. The peers form hierarchies in which each peer stores summarized data for the peers belonging to its subtree. Each peer that receives a query first checks its local index for any matches. Then, if it is an internal peer, it checks its merged index and if there is a match it forwards the query to its subtree [31]. Although their approach is effective for simple linear XPath queries, it can not handle the descendant axis in the query effectively and precisely.

Kd-synopsis [32] is a graph-structured routing synopsis based on length-constrained FBsimulation relationship, which allows the balancing of the precision and size of the synopsis according to different space constraints on peers with heterogeneous capacity. Here the k and d are length constraints to be imposed over the backward and forward-simulation relationships. which are theoretical foundations of their routing synopsis. The authors also address the aggregation and update maintenance issues for routing tables consisting of kd-synopses. Although a kd-synopsis is more precise than the approach in [31], it can deal with only simple Branching Path Queries(BPQ) [35] without full-text search predicates.

2.4.2 XML Data Management in Structured P2P Systems

Due to the inherent better scalability of structured P2P systems, much work has also been done on query processing of XML data in structured P2P systems. In structured P2P systems, data items are placed on specific nodes. Most of structured P2P systems are based on publishing and indexing data in distributed hash tables(DHT), that follows a strict topology in which each peer has a specific number of neighbors.

In [36], Papadimos et al. present a framework for querying distributed XML data in a P2P environment. Their system can be viewed as a loosely structured P2P system because it provides distributed catalogs for efficient query routing. Their approach differs from traditional distributed query processing since no coordinator reformulates a client query into several sub queries, coordinates query executions and combines results from different data sources. Also, mutant query processing is different from wrapper-mediator-based approach in data integration because there is no global schema, no translation from the common query language into local query language (vice versa), and no mediation between heterogeneous data sources. The essence of mutant query processing is to evaluate a query plan partially on one peer, combine the partial results and the rest of query plan into a mutant query plan (MQP), then send it to some other peer for further evaluation [36]. The final result is sent back to the original peer as and when the query plan is fully evaluated. A MQP is very similar to a traditional query plan except that in MQP, URNs are used to refer to abstract resources that may be stored on remote peers and verbatim XML fragments represent partial results from partial evaluation of the plan. MQPs are encoded in XML and transmitted among peers. In mutant query processing, an important issue is how to resolve URNs and route the mutated query plan to an appropriate peer. To address this issue, distributed catalogs are employed to efficiently route queries to peers with relevant data. To construct distributed catalogs, multi-hierarchic namespaces(MHNs) are used to categorize data. Each hierarchy in a MHN is called a dimension. The cross product of each category from each dimension forms an interest cell. several interest cells form an interest area. A data provider can use interest areas to describe the data they provide and data consumers can also use interest areas to form queries.

Galanis et al. [27] propose a meta-data indexing scheme and query evaluation algorithm over distributed XML data in structured P2P networks. In their framework, a distributed catalog service is distributed across the data sources themselves and is responsible for routing user queries to the relevant peers. They also propose a structure-based key splitting scheme for load balance. Unfortunately, the authors do not address the indexing cost, since their design is based on the assumption that querying is far more frequent than data placement. Their framework is more suitable to data-centric XML data rather than to document-centric ones, since the latter may include large text portions inside specific tagnames (such as, the tagname paragraph), which results in the routing of large parts of a document to the same nodes. In their system, XPath queries are routed to peers based on the last tagname in the query, which serves as the DHT lookup key. For example, if the user issues the following full-text search query Q:

```
//biblio/publisher[name = "Wiley"]
        //book[author/lastname = "Smith"]
               [title contains "XML" and "SAX"]/price
```

Q will be routed to a node based on the DHT key price and the list of all nodes who own documents whose structures match Q are collected and routed to a second node based on the key lastname. The latter node uses the data summary for lastname to keep those nodes from the incoming list who have documents with lastname "Smith". Then, the resulting list of nodes is routed to a third node based on the DHT key text, which is again shortened using the data summary for text. Their evaluation of Q will return even those nodes who own documents that have one book written by Smith and another with XML and SAX in their titles. That is, their method does not address containment relationships between predicates. Furthermore, if the last tagname of a query is very common, such as price, their method may involve routing a large list of nodes (all nodes who own documents with tagname price). The proposed patching of the problem, called structured-based key-splitting, requires a major change in the design, in which all possible pairs tag/price that match the endpoint of a query are extracted from its structural summaries and used as routing keys. Nevertheless, evaluation becomes impossible if all endpoints of a query take the form //tag and all these tags have been split. In addition, their framework only supports general XPath queries without full-text search predicates.

In [28], the authors propose XP2P, which indexes XML data fragments in a P2P system based on their concrete paths that unambiguously identify the fragments in the document (by using positional filters in the paths). The search key used for fragment indexing is the hash value of its path. Thus, XP2P can answer simple, but complete XPath queries (without predicates or descendant-of steps) very quickly, in one peer hop, using the actual query as the search key. The main drawback of this method is that retrieving a complete data fragment with all its descendants would require additional hops among peers by extending the query with the child tagnames of each retrieved fragment recursively, until all descendants are fetched. The descendant-of step requires even more searching by considering the parts of the query that do not have descendant-of steps and appending to them the child tagnames of the retrieved fragments (which makes it impossible to answer queries that start with a descendant-of step). Although XP2P can answer simple linear XPath queries efficiently, it does not support complex XPath queries with conditions or search predicates.

In [29], the authors propose a system termed KadoP, which indexes XML data in the form of postings in DHT networks, where each posting encodes information on an element or a term. Given a query, the KadoP system combines the postings stored in the index to locate the peers that can contribute to the query, and forwards the query to these peers where the final results are computed. In a more recent work [30], in order to reduce the data indexing and query processing overhead, the authors employ a horizontal partitioning scheme, in which a large set of postings is distributed among peers based on range conditions [30]. This scheme enables a highly parallel twig join algorithm that can reduce the total processing time. To limit data transfers, they also introduce Structural Bloom Filters for distributed structural XML joins. Although the authors propose some optimization techniques to reduce the negative impacts caused by full data indexing, it is still an open problem to make the system scale to a large and dynamic DHT network and XML data repository, where data volumes are huge and updates are frequent.

2.5 Peer-to-Peer Data Management over Relational Data

Our framework is also related to work on relational query processing systems on P2P architectures, such as PIER [37, 38, 39], and on data integration based on P2P systems, such as Piazza [40]. PIER adapts the existing distributed relational database technology to a DHT-based P2P architecture. One adaptation is the core join algorithm, which is a DHT-based symmetric hash join. Another novel idea introduced by PIER is the soft-state timeline, where each object is stored at a node for a relative short time period and discarded afterwards. To extend their lifespan, the publisher must periodically extend the objects' timeline by sending renewal signals to probe the storage nodes. If the probing fails, the objects are published again. This periodic probing guarantees a high availability of data, even after node failures, but it comes with a high cost. In the Piazza data integration system, each peer exports its own relational schema while a user poses queries over a mediated schema, which defines the integration mappings between the peer schemas. This framework has been extended to handle XML data, where each peer exports an XML Schema. Even though our framework can be used for data integration too, this task is accomplished in a more dynamic environment, with no need for a mediated schema.

2.6 Contributions of this Thesis Work Compared to Related Work

In this thesis work, we propose a novel and effective framework for indexing and searching distributed XML data based on concise data synopses extracted from original XML documents. Several significant contributions are made in this thesis work that distinguish our work from related work in the literature. First, unlike most of work in keyword search or semantic search paradigms, which fully index XML data at the granularity of elements or terms, we extract a small number of compact data synopses from original XML documents and publish these meta-data in a distributed environment. The searching process is based on these concise data synopses. This way, the overhead for data indexing, data publishing, and query response time can be effectively reduced. Second, our framework supports general XPath queries with multiple full-text search predicates instead of keyword queries, which may degrade the query precision, or simple semantic search, which does not encode the complex structural constraints in the query and thus is still not precise enough. Our query evaluation not only considers the structural constraints among context elements and terms, but also considers the complex containment relationships among different search predicates in the query, and thus can effectively improve the query precision. Finally, compared to the related work on XML data management in P2P environments, our framework can support more general and complex queries, require less data publishing and indexing overhead, achieve higher query precision, and scale more gracefully to large online XML data repositories.

CHAPTER 3

SYSTEM OVERVIEW

In our framework, as and when an XML document is indexed, only a structural summary and a small number of compact data synopses are extracted from the document and indexed on the server. As the query client, a user can pose a simple full-text XPath query and locate a ranked list of XML documents that satisfy both the structural constraints and the full-text search predicates in the query. Based on the returned ranked document list, the user can choose some interesting document hits and send the original query to the document owner, which evaluates the query over the actual XML documents and the actual XML fragments will be returned as the query answers.

An XML document in our framework is indexed on both its textual content and its structural makeup, called the structural summary, which is a concise summary of all valid paths to data in the document. Even though a formal schema, such as an XML Schema or a DTD, would have been useful information for indexing and accessing data, our framework does not require it. The textual content of a document is summarized into data synopses, which capture both content and positional information in the form of bit matrices across the dimensions of indexed terms and their positions in the document. These matrices are small enough to accommodate frequent document publishing but precise enough to reduce the number of false positives in locating documents that satisfy the structural and content constraints of a query.

Based on our framework, we first build a centralized indexing and searching system, in which the server is responsible for indexing the meta-data extracted from distributed XML data sources and answering queries from any client in the system. Then, we extend our framework and apply it to a DHT-based structured P2P network, in which both data indexing and query answering are in a pure distributed fashion. Both structural summaries and data synopses are indexed in the distributed hash table using appropriate DHT keys. As a client submits a query, a distributed query plan is generated and evaluated in a totally distributed fashion in the P2P network, collecting qualified document hits along the way from peer to peer and finally returning a list of document locations to the client.

In this chapter, we first describe the syntax of the search queries used in our framework and then overview the functionality of the key components in our system.

3.1 Query Specification

Our query language is XPath [2] extended with simple full-text search. We extend the XPath syntax with a full-text search predicate $e \sim S$, where e is an arbitrary XPath expression. This predicate returns true if at least one element from the sequence returned by e matches the *search specification*, S. A search specification is a simple IR-style boolean keyword search that takes the form

$$\text{“term”} \ \mid \ S_1 \ \underline{\text{and}} \ S_2 \ \mid \ S_1 \ \underline{\text{or}} \ S_2 \ \mid \ (\ S \ )$$

where S, S_1, and S_2 are search specifications. A term is an indexed term that must be present in the text of an element returned by the expression e. As a running example used throughout the paper, the following query, Q:

```
//auction//item[location ~ "Dallas"]
   [description ~ "mountain" and "bicycle"]/price
```

searches for the prices of all auction items located in Dallas that contain the words "mountain" and "bicycle" in their description.

3.2 System Architecture

Our system architecture is shown in Figure 3.1. The *Meta-data Indexer* is responsible for indexing XML meta-data on the server. *Meta-data Indexer* extracts the meta-data from *XML Document Repository* and constructs efficient indexes for these meta-data. We assume XML documents published by any client are cached on the server and the *Meta-data Indexer* directly extracts the meta-data from these documents. Any client can submit a full-text XPath query to the system. When a client submits a query to the server, the *Query Footprint Extractor* extracts the query footprint from the query, which is sent to the *Structural Summary Matcher* that matches the query footprint against the *structural summaries* derived from all indexed documents. The query footprint captures structural components and entry points associated with search predicates in the query. The resulting full label paths that match the entry points in the query footprint are sent to the *Query Optimizer*. The *Query Optimizer* retrieves the corresponding *content synopses*, *positional filters* and *document synopses*, and employs a two-phase containment filtering algorithm described in Section 5 to evaluate the IR search predicates and locate all the qualified document locations. In addition, by accessing the *TF*IDF Synopses*, a scoring scheme is incorporated into the query evaluation to rank the document locations. Finally, a ranked list of document locations are returned to the client.

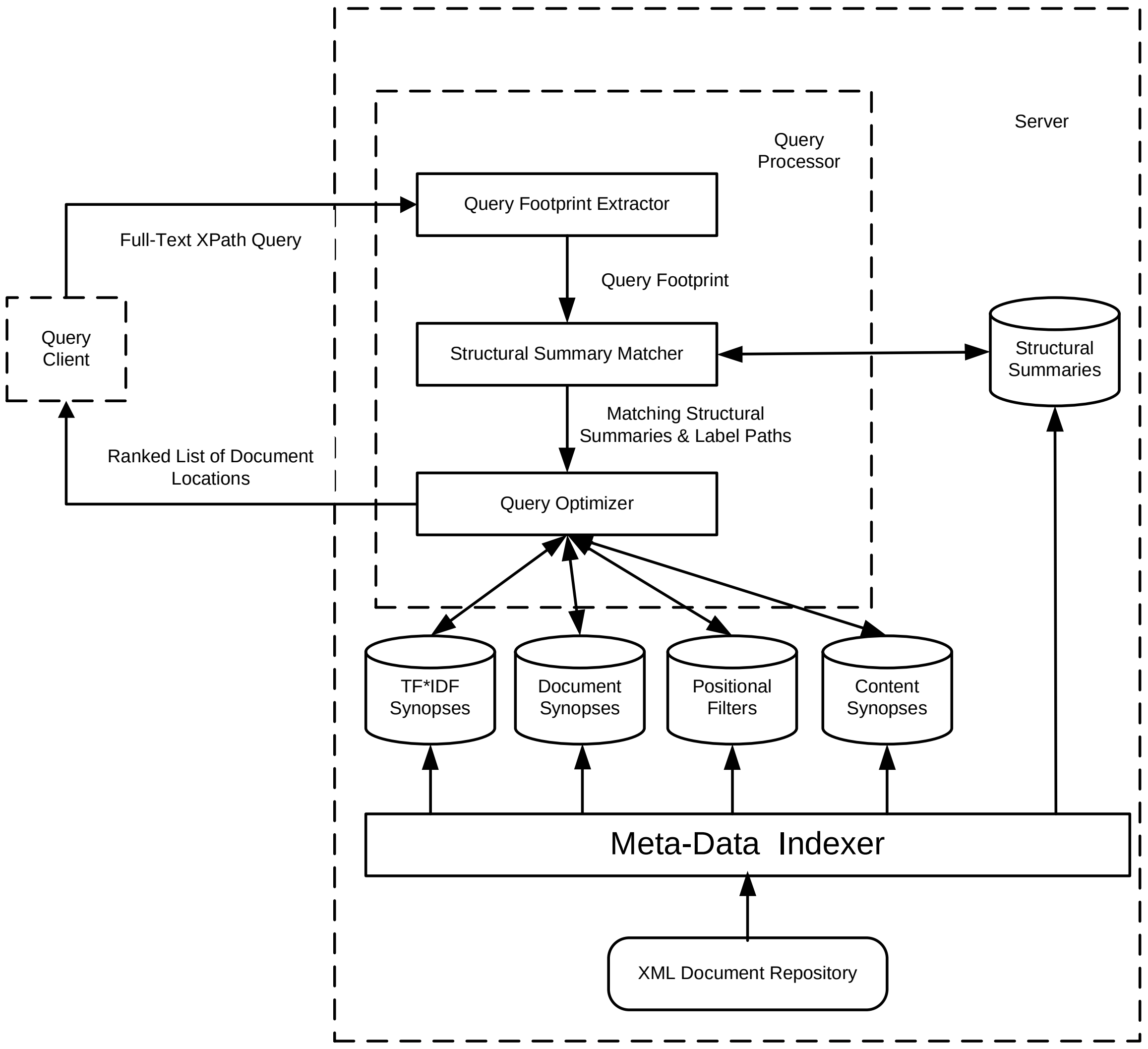


Figure 3.1. System Architecture.

CHAPTER 4

DATA INDEXING

Similar to [41], an XML document in our framework is modeled as a labeled, directed tree. An example XML document is shown in Figure 4.1. The document represents the auction information from an auction web site. For brevity, we omitted the attribute nodes in the document. When an XML document is indexed in our system, instead of indexing each single element or term in the document, only a structural summary and a small number of concise data synopses are extracted from the document and indexed on the server. The meta-data indexed in our framework are: *Structural Summaries* (SS), *Content Synopses* (CS), *Positional Filters* (PF), *Documents Synopses* (DS), *TF Synopses*, and *IDF Synopses*. We postpone the description of *Documents Synopses* to Chapter 5 after the introduction to query processing because it is used to improve the query response time. Similarly, since *TF Synopses*, and *IDF Synopses* are closely related to the ranking scheme in our framework, we also postpone their descriptions to Chapter 6 when we introduce our ranking function.

4.1 Motivation

The problem that we examine first is, given an XML document and an XPath query that contains search specifications, is the document likely to match the query? Since we are using synopses, our goal is to find an approximation method that reduces the likelihood of false positives, does not miss a solution, and does not require high placement and maintenance overheads. Since we are interested in evaluating IR-style search specifications, the document content is broken into simple terms (keywords),

```
<auction>
  <sponsor>
    <name> ebay </name>
    <address> 1040 W. Abram Dr. San Jose, CA </address>
  </sponsor>
  <item>
    <name> bicycle </name>
    <description> a mountain bicycle used for 2 years </description>
    <payment> Credit Card, Cash, Money Order </payment>
    <location> Dallas, TX </location>
    <price> 30 </price>
  </item>
  <item>
    <name> car </name>
    <description> 1999 Toyota Camery LE, mileage 80K </description>
    <payment> Credit Card, Check </payment>
    <location> Arlington, TX </location>
    <price> 5000 </price>
  </item>
  <item>
    <name> house </name>
    <description> a brand new house build in 2007 </description>
    <payment> Credit Card, Money Order </payment>
    <location> Arlington, VA </location>
    <price> 150,000 </price>
  </item>
</auction>
```

Figure 4.1. An Example XML Document.

followed by stemming and stop word elimination. The resulting terms, called the indexed terms, are summarized in content synopses. More specifically, for each unique label path in a document that directly reaches text, we create one content synopsis that summarizes the indexed terms in this text along with their positions. The positional information is derived from the document order of the begin/end tags of the XML elements in the document. That is, the position of an element tag is the number of the begin and end tags that precede this tag in the document. The positional range of an XML element, on the other hand, consists of the positions of the begin/end tags of the element, while the positional range of an indexed term is the positional range of the enclosing element. That is, terms that belong to the same XML element have the same positional range. All positions in a document are scaled and mapped into a bit vector of size L, called a positional bit vector, so that the last position in the document is mapped to that last bit in the vector.

The positional dimension of the synopses is necessary due to the containment restrictions inherent in the search specifications of a query. For example, the search specification $e \sim t1$ and $t2$ for two terms t1 and t2 becomes true if and only if there is at least one document node returned by e that contains both terms. Using one-dimensional term bitmaps alone, such as Bloom Filters, and checking whether both the $t1$ and $t2$ bits are on, will give us a prohibitive number of false positives (as is shown in our experimental evaluation). For instance, using Bloom Filters, the running query Q might have returned all documents that have one item whose location is “Dallas”, a second item whose description contains “mountain”, and a third item whose description contains “bicycle”. Therefore, term position is crucial in increasing search precision and reducing false positives.

Informally, given a document, the content synopsis Hp associated with a label path p is a mapping from an indexed term t to a positional bit vector. In our implementation, a content synopsis is summarized into a bit matrix of size $L \times W$, where W is the number of term buckets and L is the size of bit vectors. Then, $Hp[t]$ is the matrix column associated with the hash code of term t. If there are instances of two terms $t1$ and $t2$ in the document that have the same positional ranges (ie, they are contained in the same element), then the $Hp[t1]$ and $Hp[t2]$ bit vectors should intersect (ie, their bitwise anding should not be all-zeros). For example, we can test if a document matches the search specification **description** $\sim$ **"mountain" and "bicycle"** by bitwise anding the vectors $H3[mountain]$ and $H3[bicycle]$, which correspond to the node 3 (the node description) in Figure 4.2. If the result of the bitwise anding is all zeros, then the document does not satisfy the query (the opposite of course is not always true).

But given the bit vectors $H3[mountain]$, $H3[bicycle]$, and $H4[Dallas]$, how can we enforce the containment constraint in query Q that the item whose location is "Dallas" must be the same item whose description contains "mountain" and "bicycle"? We cannot just use bitwise *anding* or *oring*. We address this problem by using M bit vectors F_p so that the positional range of the *ith* item goes to the i mod M bit vector. That way, two consecutive items in the document, the i and $i+1$ items, are placed to different bit vectors, thus reducing the likelihood of overlapping, which may in turn result in false positives. M should be at least 2, since it is very common to have a situation similar to that of consecutive items. Thus, our positional filters F_p are matrices of size $L \times M$.

4.2 Structural Summary

To match the structural components in the query during the query evaluation, we construct a type of data synopses, called **Structural Summary**(SS) [41], which is a structural markup that captures all the unique paths in the document. In our

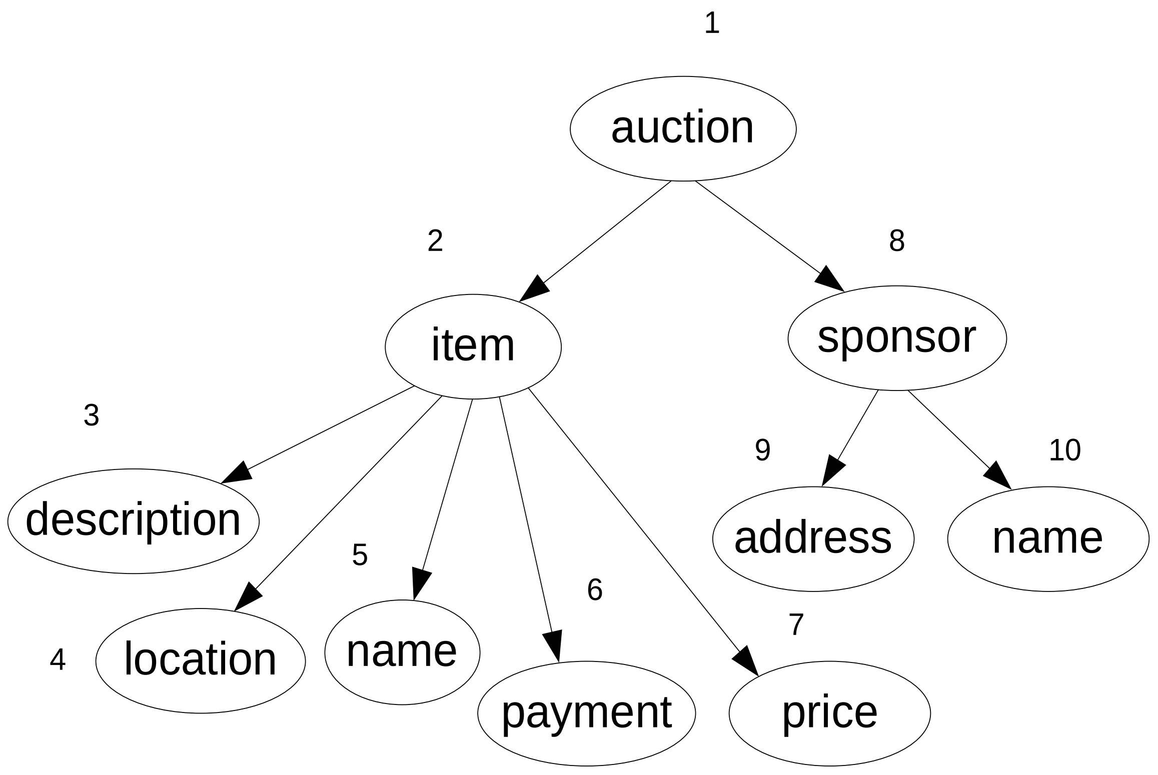


Figure 4.2. Structural Summary Example.

framework, a structural summary of an XML document takes the form of a labeled, directed tree. The idea is to preserve all the paths in the document in the summary tree, while having far fewer nodes and edges. The structural summary of the example XML document is shown in Figure 4.2. Each node in a structural summary has a tagname and a unique id. As we can see, one SS node may be associated with more than one elements in the document. For example, in Figure 4.2, the node **item** corresponds to three **item** elements in the example XML document. Note that a structural summary node is not associated to an **extent** in our framework.

4.3 Content Synopses

To capture the textual content of a document, for each text node k in the structural summary S of the document D (an SS node that corresponds to at least one document element that contains text), we construct a *content synopsis* (CS) H_p^D to summarize the textual data associated with k, where the path p is the unique simple path from the root of S to the node k in S. H_p^D is a bit matrix of size $L \times W$, where W is the number of term buckets and L is the number of positional ranges in the document. The positional information is represented by the document order of the begin/end tags of the elements. More specifically, for each term t directly inside an element associated with the node k whose begin/end position is b/e, we set all matrix values $H_p^D[i, \text{hash}(t) \bmod W]$ to one, for all $\lfloor b{\times}L/|D|\rfloor \leq i \leq \lfloor e{\times}L/|D|\rfloor$, where 'hash' is a string hashing function and $|D|$ is the document size. That is, the $[0, |D|]$ range of tag positions in the document is compressed into the range $[0, L]$. H_p^D is implemented as a B^+-tree index with index key p, because, during the query processing, we need to retrieve the content synopses of all documents for a given path p. For example, the content synopsis for the SS node **description** ($k = 3$) is illustrated on the right of Figure 4.3. Each dark cell represents a bit set to one. As we can see, after the term "bicycle" is hashed to the term bucket 11, we obtain a bit vector that has 4 one-bit ranges (displayed with black color). Each one-bit range represents a **description** element that directly contains "bicycle" in the document. The start/end of a range corresponds to the document order of begin/end tag of a **description** element. Since a node in a structural summary may correspond to many elements in a document, the positional dimension is very useful information when evaluating search predicates in a query. In our running query example, both "mountain" and "bicycle" have to be in the same description element in a document to satisfy the query Q. If we had used one-dimensional Bloom filters [16], to check whether the bits for both terms are both

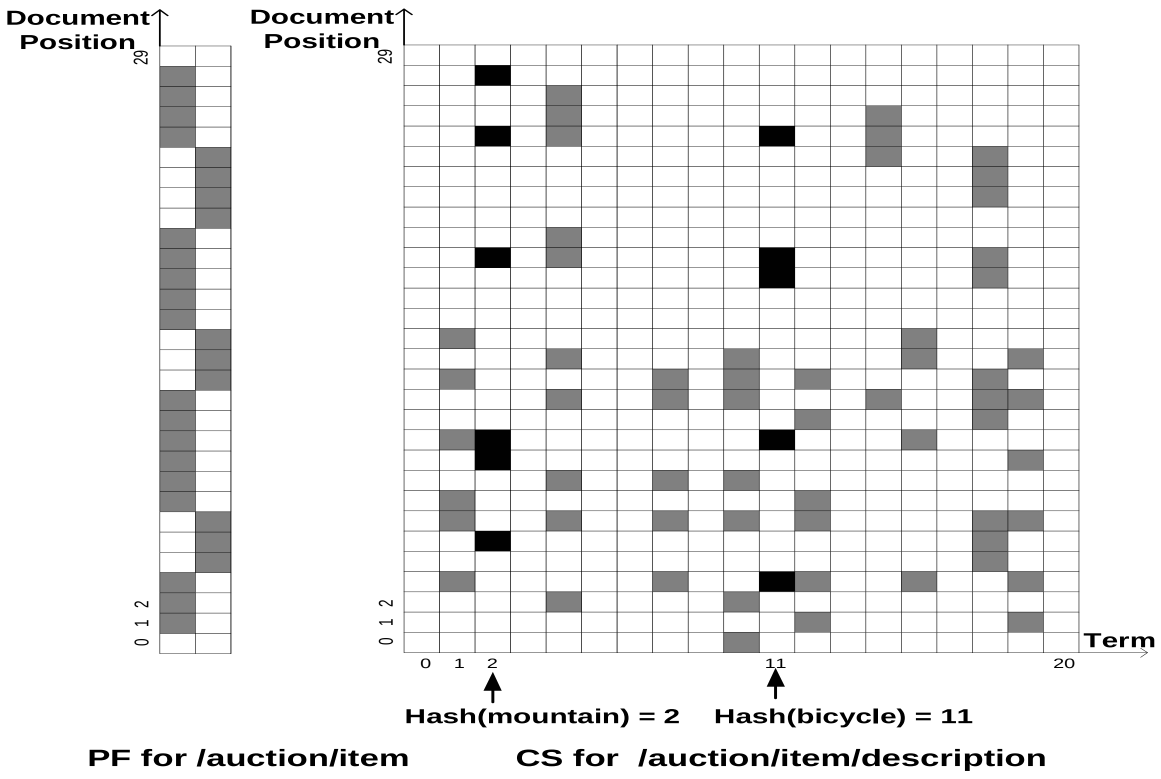


Figure 4.3. Data Synopses Example.

one, we may have gotten a prohibitive number of false positives. For instance, Q may have returned an unqualified document that has an item whose description contains "mountain", and another item whose description contains "bicycle". As such, term positional information is crucial in increasing the search precision. With our content synopses, we can evaluate the search predicate description $\sim$ "mountain" and "bicycle" by bitwise *anding* the vectors H_3["mountain"] and H_3["bicycle"], which are the two darker bit columns extracted from the content synopsis in Figure 4.3. If all bits in the resulting bit vector are zeros, the corresponding document does not have both terms in the same **description** element and thus does not satisfy the search predicate.

4.4 Positional Filters

Although the positional information in CS enforces the constraint that the terms in the same search predicate must be in the same element associated with the predicate, it can not ensure that different elements associated with different search predicates are contained in the same element in a document. For example, given the relevant bit vectors H_3[“mountain”], H_3[“bicycle”], and H_4[“Dallas”] only, we can not enforce the containment constraint in Q that the item whose location contains “Dallas” must be the same item whose description contains “mountain” and “bicycle”. To address this problem, for each non-text node n in the structural summary of a document, we construct another type of data synopsis, called *Positional Filter* (PF), denoted by F_p^D. As we did for H_p^D, F_p^D is also implemented as a B-tree with index key p. F_p^D is a bit matrix of size $L \times M$, where L is the document positional ranges of the elements associated with node n that is reachable by the label path p, and M is the number of bit vectors in F_p^D. The value of M should be no less than 2 because we want to map consecutive elements in a document to different bit vectors, thus reducing the bit overlaps of consecutive elements when their mapped begin/end ranges intersect. More specifically, the positional range of the ith element goes to the $i \bmod M$ bit vector, so that two consecutive elements in the document, the i and $i+1$ element, are placed to different bit vectors, thus reducing the likelihood of overlapping, which may result to false positives. The positional filter for SS node **item** is demonstrated on the left in Figure 4.3. The 7 one-bit ranges indicate there are 7 **item** elements in the document. We will show you how to utilize content synopses and positional filters to determine if a document satisfies all the search predicates in the query in the next section.

Table 4.1. Data Set Characteristics and Data Synopses Size

Data Set	Data Size (MB)	Files	Avg. File Size (KB)	Avg. SS Size (KB)	Avg. CS Size (KB)	Avg. PF Size (KB)
XMark1	5.63	1150	5	0.408	0.292	0.015
XMark2	55.8	11500	5	0.413	0.305	0.016
XBench1	95.2	266	358	0.419	0.950	0.095
XBench2	1050	2666	394	0.427	2.012	0.174

4.5 Experimental Evaluation

We have implemented our framework using Java (J2SE 6.0) and Berkeley DB Java Edition 3.2.13 [42] was employed as a lightweight storage manager. We first conducted extensive experiments to evaluate the scalability and efficiency of our indexing scheme. Our experiments were carried out on a WindowsXP machine with 2.8GHz CPU and 512M memory. We used two XML benchmark data sets XMark [43] and XBench [44] as our data sets. The main characteristics of our datasets and data synopses size are summarized in Table 4.1.

4.5.1 Scalability of Data Synopses

From Table 4.1, we can see that for each of the XMark data sets, the average size of a structural summary and content synopsis is about 10% and 5% of the document size, respectively. The average size of a positional filter is less than 1% of the document size. For data set XBench1, although the average document size is about 70 times larger than that of XMark data set, the size of a structural summary is almost the same as that of XMark data set. The average size of a content synopsis is about 0.3% of the document size and the average positional filter size is less than 0.03% of the document size. For data set XBench2, the average size of a data synopsis is a little

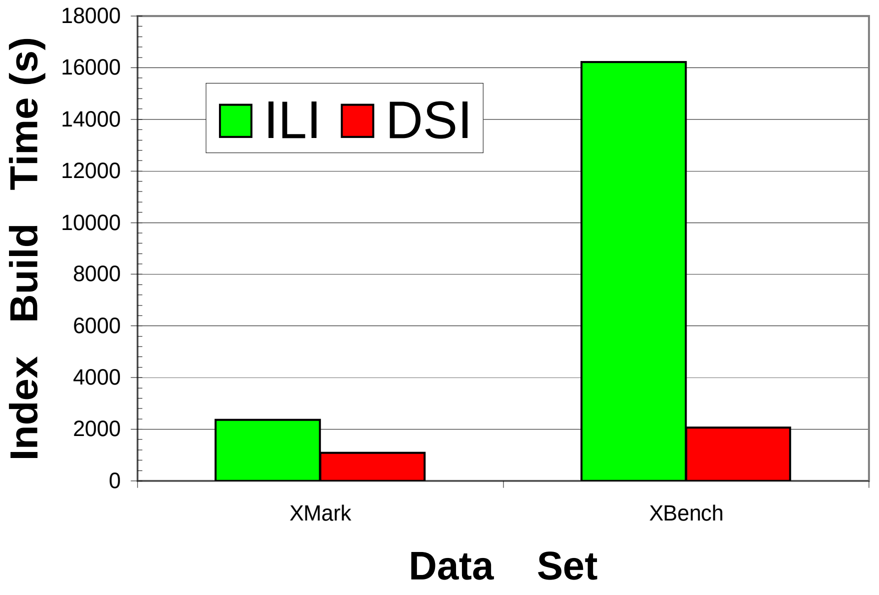


Figure 4.4. Index Build Time.

larger because each document contains about twice amount of elements. However, the average size of a content synopsis and positional filter is still only about 0.6% and 0.05% of the document size, respectively. The above results demonstrate that our data synopses are small enough to make our system scalable. Note that the size of a document synopsis is not listed in Table 4.1 because the size of a document synopsis does not affect the system scalability. A document synopsis is a global data structure for all documents and only one document synopsis is accessed for a path-term pair during the query evaluation.

4.5.2 Efficiency of Indexing Scheme

To demonstrate the efficiency of our Data Synopses Indexing (DSI) scheme, we implemented the traditional Inverted List Indexing (ILI) scheme in Berkeley DB and compared it with our indexing scheme. We chose XMark2 and XBench2 as data sets for this experiment because the data set size is the key factor for the indexing

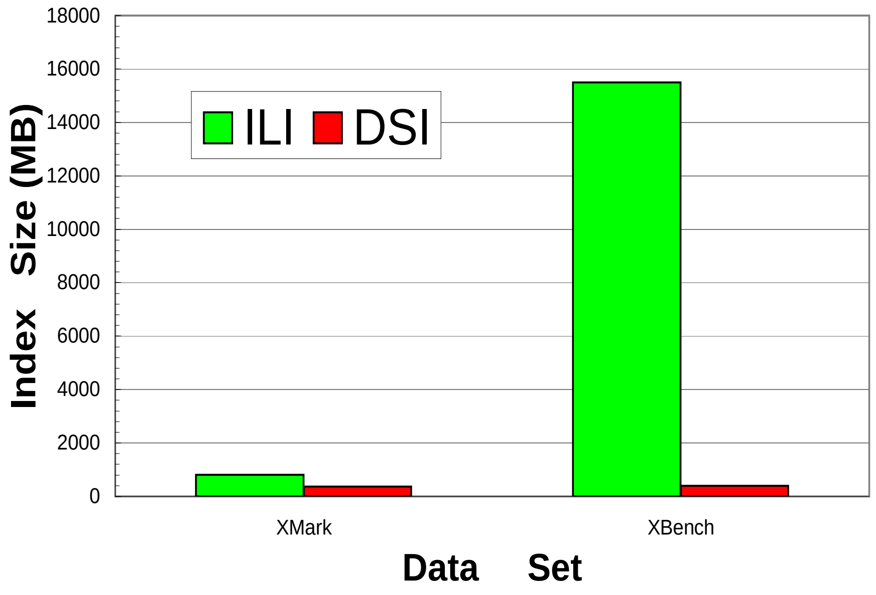


Figure 4.5. Index Size.

scheme comparison experiment. From Figure 4.4, we can see that for XMark data set, DSI consumes less than 50% indexing build time than ILI. For XBench data set, DSI consumes only around 10% indexing build time of ILI. From Figure 4.5, we can see that for XMark data set, the index size of DSI is about 50% of ILI, and for XBench data set, the index size of DSI is less than 5% of ILI. Figure 4.6 shows that for XMark data set, the improvement on query response time of DSI compared to ILI is not very significant. However, for XBench data set, DSI achieves orders of magnitude improvement on query response time than ILI. In summary, for XBench data set, DSI is much more efficient than ILI because XBench data is text-centric data generated from Text-Centric Multiple Document(TC/MD) class, which is more suitable for the index comparison experiments. In fact, DSI consumes less than 8% index build time than ILI and the index size of DSI is about 3% of that of ILI. The query response time of DSI is over 40 times faster than ILI.

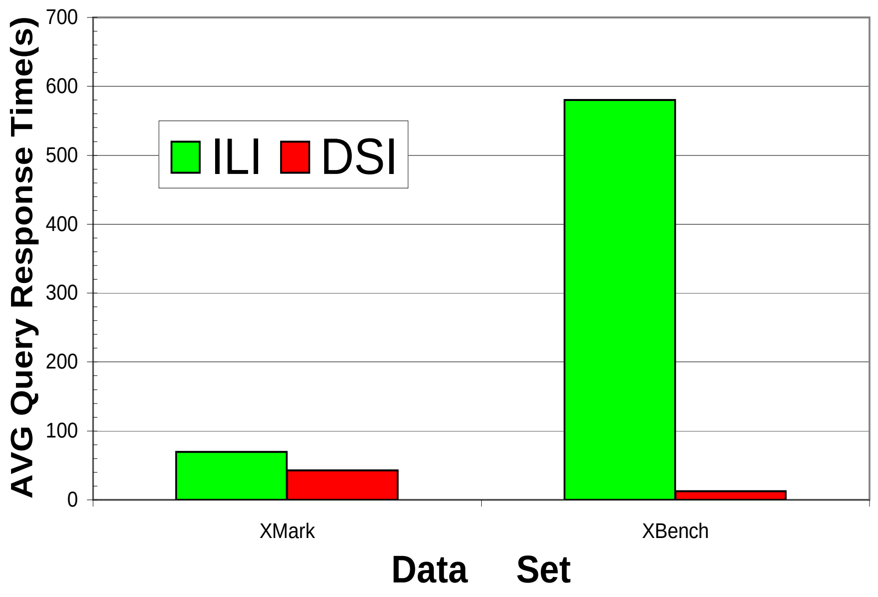


Figure 4.6. Query Response Time.

CHAPTER 5

QUERY PROCESSING

5.1 Overview

When the client submits a query, the server extracts the query footprint (QF) from the query and matches the footprint against the local indexes, through which all the matching structural summaries and distinct label path combinations corresponding to the query search predicates are retrieved. Based on the retrieved label path combinations, documents that have data synopses associated with those label paths are found and qualified documents are filtered out using containment filtering. The scores of documents are also calculated and aggregated during the query evaluation and the ranked document list is returned to the client.

5.2 Query Footprint Derivation

The first step in evaluating a full-text XPath query is deriving the query footprint from the query. A query footprint captures the essential structural components and all the *entry points* associated with the search predicates. For our running example Q:

```
//auction//item[location ~ "Dallas"]
   [description ~ "mountain" and "bicycle"]/price
```

the query footprint of Q is:

```
//auction//item:1[location:2][description:3]/price
```

The numbers 1, 2, and 3 are the numbers of the entry points in the query footprint that indicate the places where data synopses are needed for query evaluation (one positional filter for the label path associated with entry point 1 and two content synopses for the label paths associated with the entry points 2 and 3).

5.3 Structural Summary Matching

In our framework, achieving efficient query footprint matching against structural summaries is essential because a query footprint may match a large number of structural summaries in the system. The key point is to index structural summaries effectively to enable efficient query footprint matching. Borrowing ideas from structural joins in XML query processing [45], we employ a numbering scheme to encode each node k in a structural summary S by the triple (b, e, l), where b/e is the begin/end numbering of k and l is the level of k in S. In our framework, the server maintains the following mapping to store structural summaries: $\mathcal{M}_{ss} : tag \rightarrow \{(S, k, b, e, l)\}$, where tag is the tagname of the node k in S, and b, e, l are as defined above. Thus, the key operation in the structural summary matching is a structural join between two tuple streams corresponding to two consecutive location steps in the query footprint. Our index is designed in such a way that the tuples are delivered in major order S and minor order b, so that the structural join can be evaluated in a merge-join fashion. To accelerate query processing and avoid materializing intermediate tuples, we leverage the iterator model [46] in relational databases to form a pipeline of iterators derived from the query footprint (one iterator for each XPath location step). That way, the pipeline processes the structural summary nodes derived from the indexes one-tuple-at-a-time and the intermediate results are never materialized.

Let QF be the structural footprint of a query. Then the structural summary matching is accomplished by the function $\mathcal{SP}[\![QF]\!]$ that returns a set of tuples (ρ, S, k, b, e, l), where (S, k, b, e, l) is similar to that of $\mathcal{M}_{ss}$ and ρ is a vector of node numbers, such that $\rho[i]$ gives the node number in S corresponding to the ith entry point in QF. The function $\mathcal{SP}[\![QF]\!]$ is defined recursively based on the syntax of QF, generating structural joins for each XPath step. Here, we give two rules as an illustration:

$$\begin{aligned}
&\mathcal{SP}[\![QF/tag]\!] \\
&\quad = \; \{\, (\rho, S, k_2, b_2, e_2, l_2) \mid (\rho, S, k_1, b_1, e_1, l_1) \in \mathcal{SP}[\![QF]\!] \\
&\qquad\qquad\qquad\qquad\qquad\qquad \wedge\, (S, k_2, b_2, e_2, l_2) \in \mathcal{M}_{ss}(tag) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \wedge\, b_2 > b_1 \;\wedge\; e_2 < e_1 \\
&\qquad\qquad\qquad\qquad\qquad\qquad \wedge\, l_2 = l_1 + 1 \,\} \\
&\mathcal{SP}[\![QF : i]\!] \\
&\quad = \; \{\, (\rho[i] := k, S, k, b, e, l) \mid (\rho, S, k, b, e, l) \in \mathcal{SP}[\![QF]\!] \,\}
\end{aligned}$$

The first rule shows how to evaluate a *Child* location step using the index $\mathcal{M}_{ss}$ to retrieve all structural summaries S that contain tag. The second rule assigns the node number k to $\rho[i]$, where i is an entry point in the footprint. That is, it finds the node numbers in the structural summary that correspond to the entry points in the query footprint. From the node numbers in ρ, we can derive the corresponding label paths from the structural summary. In our running example, the label paths are `/auction/item`, `/auction/item/location`, and `/auction/item/description`. The persistent mappings for the data synopses stored in our system are $\mathcal{M}_H : p \rightarrow \{(H_p^D, D)\}$ and $\mathcal{M}_F : p \rightarrow \{(F_p^D, D)\}$. That is, given a label path p, $\mathcal{M}_H(p)$ retrieves all the content synopses H_p^D and $\mathcal{M}_F(p)$ retrieves all the positional filters F_p^D, for all documents D. Therefore, documents that have data synopses associated with the above derived label paths can be retrieved from $\mathcal{M}_H(p)$ and $\mathcal{M}_F(p)$. These documents are further filtered out using the containment filtering and qualified document locations are returned to the client.

5.4 Containment Filtering

Based on the derived lists of content synopses and positional filters, we can enforce the element containment constraints in the query using an operation called *Containment Filtering*. Let F be a positional filter of size $L \times M$ and V be a bit vector extracted from a content synopsis whose size is $L \times W$. The *Containment Filtering* $CF(F, V)$ returns a new positional filter F'. The bit $F'[i, m]$ is on iff:

$$\exists k \in [0, L] : \; V[k] = 1 \; \wedge \; \forall j \in [i, k] : \; F[j, m] = 1$$

Basically, the *Containment Filtering* copies a continuous range of one-bits from F to F' if there is at least one position within this range in which the corresponding bit in V is one. Figure 5.1 shows how the data synopses are used to determine whether a document is likely to satisfy the query Q (here $M = 2$). First, we do a containment filtering between the initial positional filter F_2 and the bit vector H_4[“Dallas”]. In the resulting positional filter A, only 5 one-bit ranges out of 7 in $F2$ are left. Counting from bottom to top, the $2nd$ and $4th$ one-bit ranges in $F2$ are discarded in A because there is no any one-bit range in H_4[“Dallas”] that intersects with the $2nd$ or $4th$ range, which means that neither the $2nd$ nor $4th$ **item** element contains a **location** element that contains the term “Dallas”. Similarly, we can do containment filtering between A and the resulting bit vector derived from the bitwise *anding* between H_3[“mountain”] and H_3[“bicycle”]. The 3 one-bit ranges in B indicate that 3 **items** out of 7 in F_2 satisfy all element containment constraints in the query. Thus, the document is considered to satisfy the query.

5.5 Hash-based Query Evaluation

Using only label paths as keys for indexing data synopses may result to inefficiencies, because documents with similar schemas may contain the same label path

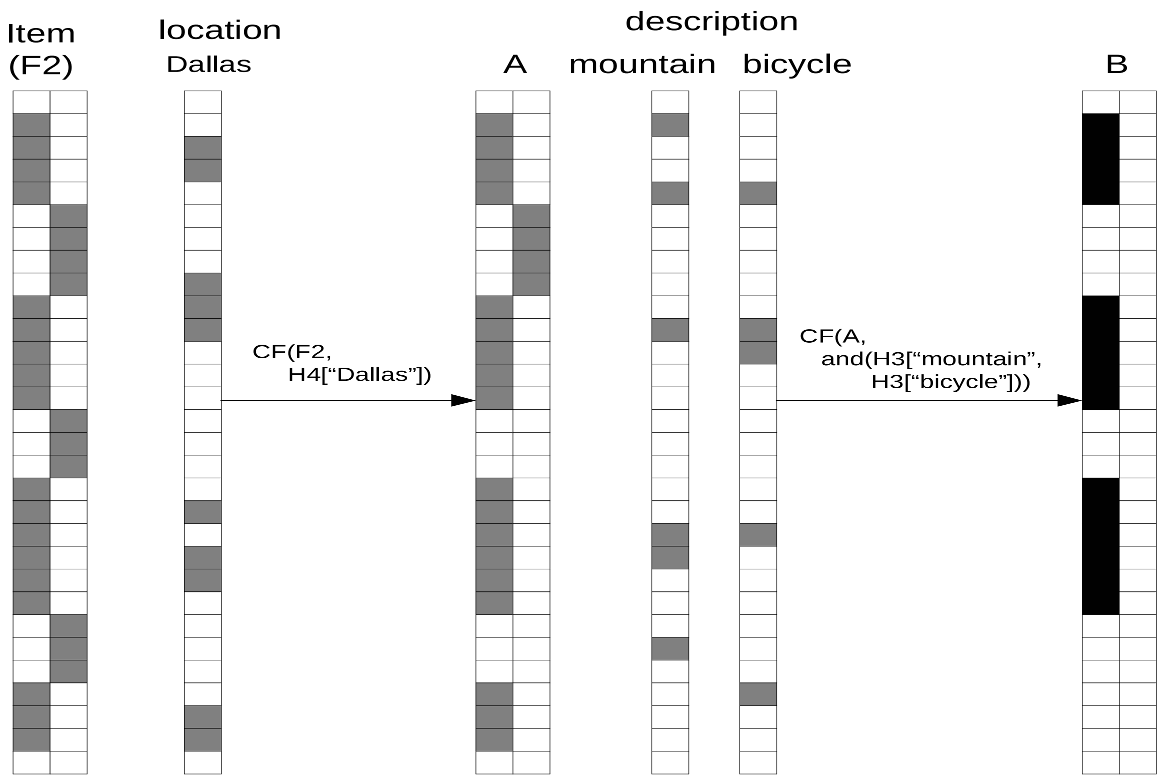


Figure 5.1. Testing Query Q Using Data Synopses.

and thus this path will be associated with a large number of data synopses in the indexes. When retrieved for query processing, these long data synopsis lists may lead to expensive join operations at each step of the containment filtering. Based on this observation, we refine our indexing scheme for data synopses and propose a hash-based two-phase containment filtering algorithm to improve query processing.

In order to reduce the number of content synopses and positional filters retrieved from the local indexes during the containment filtering, we partitioned these lists into buckets. More specifically, when a document D is indexed, the content synopsis H_p^D and the positional filter F_p^D are stored in B^+-tree indexes with key (p, hc) (rather than just p), where p is the full label path and the integer value hc is the hash code of the document ID modulo the fixed number DL, the number of document buckets.

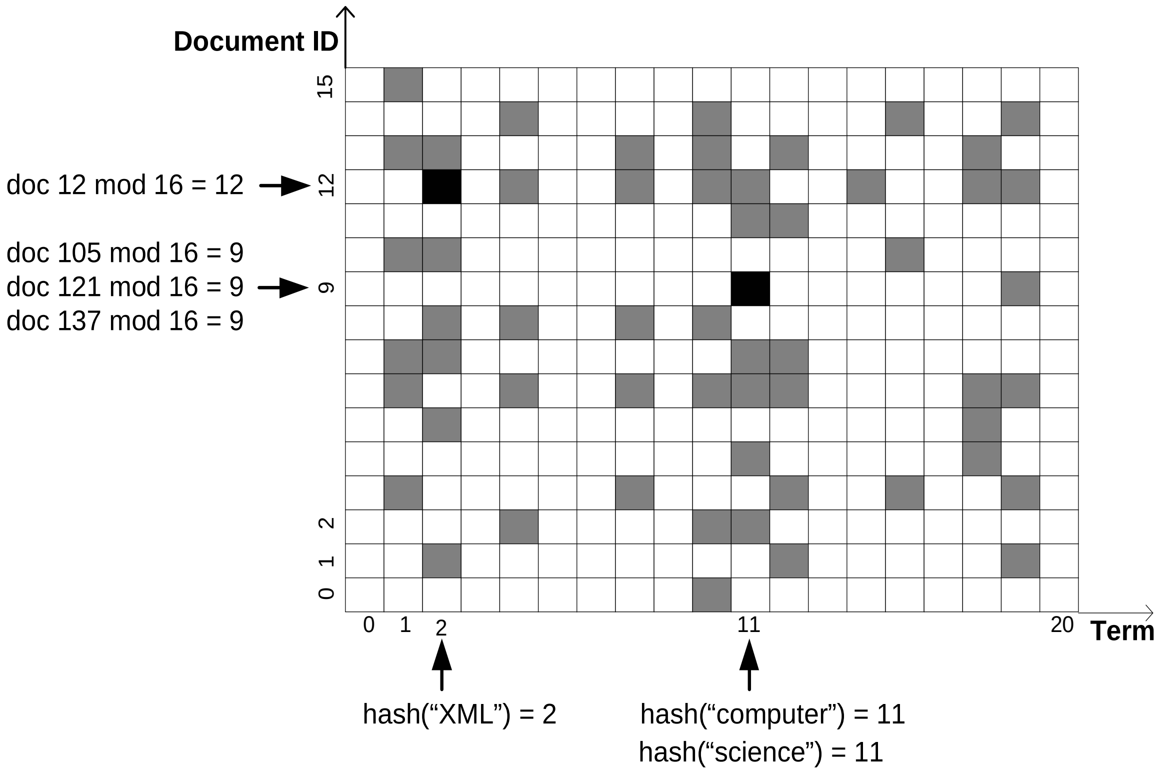


Figure 5.2. Document Synopsis Example.

Combined with a global data structure called *Document Synopsis*, described next, the above partitioning of the indexed data into DL partitions can reduce the total number of data synopses retrieved for the path p during the containment filtering, as is shown in the next section.

5.5.1 Document Synopses

In the first phase of the containment filtering, the goal is to quickly identify the documents that may contain all the path-term pairs derived from the structural summary matching, and prune unqualified documents that contain only partial path-term pairs. This information derived from the first phase will guide the full containment filtering in the second phase. For all the documents that contain a path-term pair in the corpus, we construct a global data structure called *Document Synopsis*, denoted by DS_p. Basically, for each text label path p, a document synopsis is a bit matrix of size $DL \times DW$, where DW is the number of term buckets and DL is the number of document ID buckets. If a document with ID d contains the path-term pair (p, t), then the bit $DL_p[\text{hash}(t) \bmod DW, \text{hash}(d) \bmod DL]$ is set to one, when the document is indexed.

The structure of a document synopsis is shown in Figure 5.2. The dark cells represent the one-bits. Suppose that the document synopsis is associated with the path `/biblio/book/paragraph`. Since document 12 contains the path-term pair (`/biblio/book/paragraph`, "XML"), the corresponding bit is set to one, which is emphasized by a black cell in the figure. Different documents may be mapped to the same document ID slot and different terms may be hashed to the same term bucket. For example, document 105, 121, and 137 are mapped to the slot 9 and term "computer" and "science" are hashed to the bucket 11. In this case, the corresponding bit is set to one only once, which is also emphasized by a black cell in Figure 5.2.

5.5.2 Two-phase Containment Filtering

Based on the new indexing scheme and document synopses, we propose a two-phase containment filtering strategy to optimize our query processing, which is given in Algorithm 1. The first phase is a pre-processing stage that prunes unqualified documents that do not contain all the path-term pairs. The resulting bit vector V_f is a filter that carries information about all the documents that may contain all the path-term pairs $(p_1, t_1), (p_2, t_2), \ldots, (p_n, t_n)$, which is indicated by the one-bits in V_f. In the second phase, the full containment filtering is carried out with the guide of V_f. Basically, at each step of the containment filtering, (p_i, hc_i) is used as the key to retrieve all content synopsis hits or positional filter hits, where p_i is the corresponding

Algorithm 1: Two-phase Containment Filtering

```
Input:  $p_0$   /* the path associated with positional filter */
        $(p_1, t_1)$, $(p_2, t_2)$, ... , $(p_n, t_n)$   /* n path-term pairs associated with content synopses */
Output: $L_{PF}$   /* the list of positional filter hits of qualified documents */
1:  $L_{PF} := emptyList;$
2:  /* Obtain the filtering vector $V_f$ in phase one */
3:  for i = 1 to n do
4:      Use $p_i$ as the key to retrieve $DS_{p_i}$ in local indexes;
5:      Map $t_i$ along the Term axis in $DS_{p_i}$ to obtain the bit vector $V_{p_i}^{t_i}$;
6:  end for;
7:  $V_f := \bigcap_{i=1}^{n} V_{p_i}^{t_i}$;   /* bitwise anding all bit vectors */
8:  /* Do actual containment filtering with the guide of $V_f$ in phase two */
9:  for each one-bit $b_j$ in $V_f$ do
10:     k : = the index number of $b_j$ in $V_f$;
11:     $L_0^{b_j}$ := the positional filter list retrieved using $(p_0, k)$ as the key;
12:     for i = 1 to n do
13:         $L_i^{b_j}$ := the positional filter list retrieved using $(p_i, k)$ as the key;
14:         $L_0^{b_j} := CF(L_0^{b_j}, L_i^{b_j});$
15:     end for;
16:     $L_{PF} := L_{PF} \cup L_0^{b_j}$;
17: end for;
18: return $L_{PF}$;
```

path derived from SS matching, and hc_i is the index number of the one-bit in V_f. The goal is using V_f to avoid accessing unqualified data synopses and retrieve only the data synopses of the documents that contain all the path-term pairs, thus effectively reducing the number of data synopses retrieved from the indexes before the join operation.

Table 5.1. Query Workload over XMark and XBench Dataset

Dataset	Query	Query Expression
XMark	Q1	/site//item[location ~ "United"][payment ~ "Creditcard" and "Check"]/description
XMark	Q2	//regions//item[location ~ "States"][payment ~ "Creditcard" or "Cash"]/name
XMark	Q3	/site//item[location ~ "United"][payment ~ "Creditcard"]/description
XMark	Q4	//regions//item[location ~ "States"][payment ~ "Check"]/quantity
XMark	Q5	/site//item[description//text ~ "gold"]/name
XMark	Q6	/regions//item[description//text ~ "character "]/payment
XMark	Q7	//closed_auction[type ~ "Regular"][annotation//text~ "heat"]/date
XMark	Q8	//closed_auction[annotation//text~ "heat" or "country"]/seller
XMark	Q9	//closed_auction[annotation//text~ "heat" and "country"]/buyer
XMark	Q10	//closed_auction[annotation//text~ "country"]/type
XBench	Q11	/article//body[abstract/p ~ "hockey"][section/p ~ "hockey" and "patterns"]/section
XBench	Q12	//article//body[section/p ~ "regular"][abstract/p ~ "hockey" or "patterns"]/abstract
XBench	Q13	/article//body[section/subsec/p ~ "hockey"][abstract/p ~ "hockey"]/abstract
XBench	Q14	/article//body[section/subsec/p ~ "regular"][abstract/p ~ "patterns"]/section
XBench	Q15	/article//body[section/p ~ "patterns"][abstract/p ~ "patterns"]/abstract
XBench	Q16	/article//body[section/p ~ "hockey"][abstract/p ~ "patterns"]/abstract
XBench	Q17	//prolog[keywords/keyword ~ "bold" or "regular"][title~ "regular"]/authors
XBench	Q18	//prolog[keywords/keyword ~ "bold"][title~ "bold"]/title
XBench	Q19	//prolog[genre ~ "Travel"] [keywords/keyword ~ "bold" or "stealth"]//author/name
XBench	Q20	//prolog[genre ~ "Travel"] [keywords/keyword ~ "bold"]/title

5.6 Experimental Evaluation

The query workload over XMark and XBench data set is shown in Table 5.1. For each data set, we designed 10 queries that exhibit different query structures and various number of full-text search predicates.

5.6.1 Query Precision Measurement

We chose XMark2 as the data set for this experiment because the number of documents is the key factor for this experiment and XMark2 has the maximum number of documents among all the data sets. Since our data synopsis correlates content with positional information, it is equivalent to a Two-Dimensional Bloom Filter (TDBF). We compared the query precision of our TDBF with that of the traditional One-Dimensional Bloom Filter (ODBF). The result is shown in Figure 5.3. The false positive rate of a query is defined as:

```
1 - the relevant set size/the answer set size
```

We exploited XQuery engine Qizx/open [47] to evaluate each XMark query over the data set to obtain the accurate relevant set for the query. From Figure 5.3, we can see that for queries Q_1, Q_2, Q_3, Q_4 and Q_7, the false positive rate of ODBF is over two times higher than that of TDBF because each of these queries contains multiple search predicates and the positional dimension in TDBF can effectively remove false positives during the containment filtering. For queries Q_5, Q_6, Q_8 and Q_{10}, two approaches produce the same false positive rate because each query contains only a single search predicate and the search predicate contains only one term or the boolean operator is "or". In that case, TDBF performs only a bitwise *oring* operation and the positional information is not helpful to reduce the false positives. For query Q_9, TDBF is better than ODBF because although it contains a singe search predicate, the boolean operator is "and", in which case the positional information can reduce the false positives during the containment filtering. The above result shows that TDBF is superior to ODBF when multiple predicates are presented in the query or a single predicate contains multiple disjunctive terms.

5.6.2 Efficiency of Optimization Algorithm

Similarly, we chose XMark2 as our data set for this experiment because the number of documents is the key factor for this experiment. To examine the efficiency of our two-phase containment filtering algorithm, we evaluated all the XMark queries

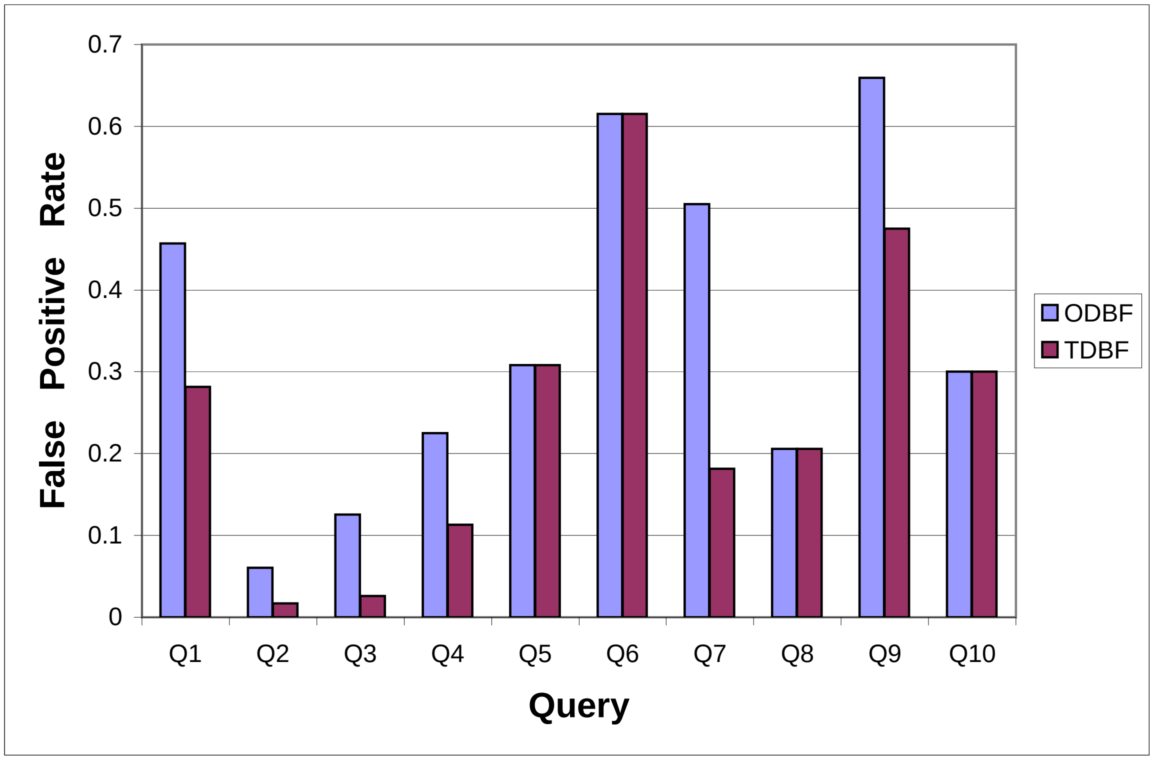


Figure 5.3. Query Precision Comparison with One Dimensional Bloom Filter.

in Table 5.1 and compared the query response times between Two-Phase Containment Filtering (TPCF) and One-Phase Containment Filtering (OPCF). As we can see from Figure 5.4, for queries Q_1, Q_2, Q_3, Q_4 and Q_7, TPCF is one time faster than OPCF because these queries contain more search predicates, which may involve more containment filtering steps and more joins between long data synopsis lists during the query evaluation. In that case, TPCF can effectively prune the unqualified document locations in the first phase, thus reduce the overall query response time. For the remaining queries, since each query only contains one search predicate and only one containment filtering step is needed in the query evaluation, the query efficiency improvement from TPCF is not very significant. This result indicates that our two-

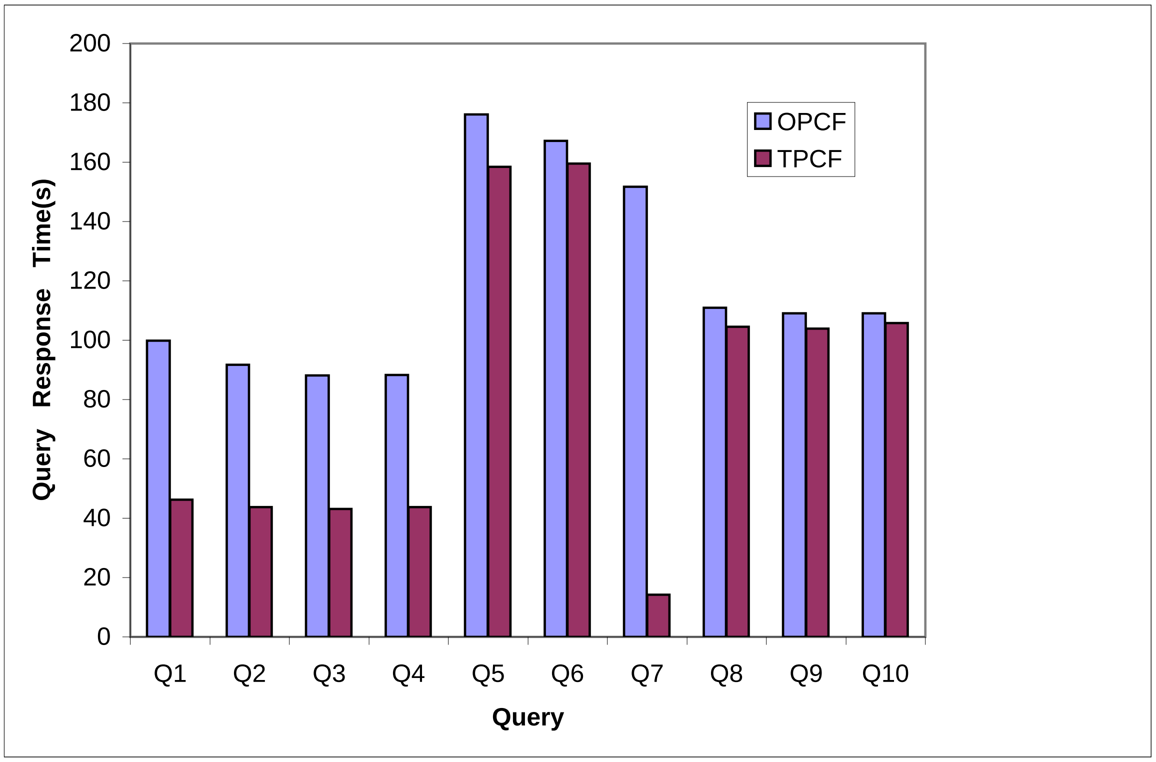


Figure 5.4. Efficiency of Two Phase Containment Filtering Algorithm.

phase containment filtering algorithm is more efficient when multiple search predicates are present in the user query.

CHAPTER 6

RELEVANCE RANKING

Since a query footprint may match a large number of structural summaries and there may be a large number of documents that match each structural summary, it is desirable to rank all the qualified documents using a scoring function and return the top k document locations to the client. Therefore, the problem we address in this chapter is, given a full-text XPath query Q and an XML document D, to design a good scoring function based on the data synopses of D to measure the relevant importance of D with respect to Q. The main challenge is that, in contrast to standard IR methods that deal with simple search keyword queries, our queries are structured XPath queries that may contain IR-style search predicates. Furthermore, only data summaries and synopses are available for ranking documents. Hence, the key to our relevance ranking method is to adapt the traditional IR approaches to calculate document scores based on the query structure and the indexed data synopses. We first extend $tf * idf$ ranking to score a document. Then we enhance it with a positional weight derived from containment filtering. Finally, we combine term proximity with the enhanced scoring to further improve the quality of ranking results.

6.1 Extended TF*IDF Scoring

In IR systems, the *TF*IDF* ranking for keyword queries is defined over a document collection [48]. This ranking mechanism considers two factors: (1) *TF*, the term frequency, that measures the relative importance of a query keyword in the candidate document, and (2) *IDF*, the inverse document frequency, that measures the global importance of a query keyword in the entire collection of documents. We extend the vector space model used in IR to measure the content similarity of an XML document D to a query Q. Due to the inherent hierarchical structure of XML data, instead of calculating the weight of a term, we consider a path-term pair as the unit for content scoring. More specifically, each path-term pair (p, t) in Q has weight w in D that is equal to $pw * tf * idf$, where pw is a *positional weight* that represents the percentage of qualified path-term pairs during the containment filtering or bitwise *anding* operation. This factor makes the scoring function more effective and precise. Finally, the cosine distance between two path-term pair vectors is calculated to represent the content score of D.

We first give the definitions of *TF* and *IDF* scores of a path-term pair, and then use an example to illustrate their calculations.

Definition 1 ***TF Score of a Path-Term Pair****. Let D be an XML document associated with the pair (p, t), where p is a full text label path from its structural summary and t is a term. Let* paths(D) *be the set of tuples $(t_x, p_x, b_x, e_x, i_x)$ for all terms t_x in D, where b_x/e_x is the begin/end position of the element that directly contains t_x, p_x is a full label path that reaches t_x, and i_x is the document position of t_x in D. The* TF *score of (p, t) relevant to D is defined as:*

$$TF^D(p, t) = |\{i_x | (t_x, p_x, b_x, e_x, i_x) \in paths(D) \land \ p = p_x \ \land \ t = t_x\}| \tag{6.1}$$

Basically, $TF^D(p, t)$ counts the number of (p, t) pairs in the document D. Notice that, if t occurs n times in the same element reachable by p in D, then (p, t) will be counted n times.

Definition 2 ***IDF Score of a Path-Term Pair****. Let N be the total number of documents in the corpus. Let D_j, $1 \leq j \leq N$, be an XML document associated with the pair (p, t), where p is a full text label path derived from structural summary* *matching and t is the corresponding term in the query. The* IDF *score of (p,t) is defined as:*

$$IDF(p,t) = log\left(\frac{N_p}{N(p,t)}\right) \tag{6.2}$$

where N_p and $N(p,t)$ are calculated as follows:

$$N_p = \sum_{j=1}^{N} |\{j|(t_x, p_x, b_x, e_x, i_x) \in paths(D_j) \ \wedge \ p = p_x\}| \tag{6.3}$$

$$N(p,t) = \sum_{j=1}^{N} |\{(t_x, p_x)|(t_x, p_x, b_x, e_x, i_x) \in paths(D_j) \wedge \ p = p_x \ \wedge \ t = t_x\}| \tag{6.4}$$

Basically, N_p counts the total number of documents that contain path p and $N(p,t)$ counts the total number of documents that contain the path-term pair (p,t) in the corpus.

We now walk through an example to illustrate *TF* and *IDF* score calculation. Suppose we have three documents in the corpus only, D_1, D_2, and D_3. The path-term pair (p,t) is (/auction/item/location, "Dallas"). Suppose D_1 contains 30 paths /auction/item/location that can reach the term "Dallas" and D_1 contains 100 paths /auction/item/description that can reach the term "bicycle". Suppose that all the three documents contain the path /auction/item/location, but only the paths /auction/item/location in D_1 and D_2 can reach the term "Dallas". Then the *TF* and *IDF* scores can be calculated as follows:

TF^{D_1}(/auction/item/location, "Dallas")=30

IDF(/auction/item/location, "Dallas")=log(3/2)=log1.5

6.2 Enhanced Scoring with Positional Weight

A path-term pair (p,t) corresponds to a positional bit vector V in the content synopsis associated with p. A one-bit range in V represents an element that contains

t and is reachable by p. For instance, in Figure 5.1, the bit vector $H_3[\text{“mountain”}]$ corresponds to the pair (/auction/item/description, “mountain”) and it contains 5 bit-on ranges. The number of one-bit ranges in the vector reflects the *TF* score of the pair (/auction/item/description, “mountain”). However, after the bitwise *anding* operation, only 3 one-bit ranges are left in the resulting bit vector, which indicates that among those 5 **description** elements, only 3 of them contain both “mountain” and “bicycle”. Similarly, after the containment filtering between the positional filter of item(F_2) and $H_4[\text{“Dallas”}]$, only 5 **item** elements are left out of 7 in the resulting positional filter(A). Thus, it is desirable to take into account this percentage of qualified elements as we calculate the weight of a path-term pair. To make the weight calculation of a path-term pair more accurate, we introduce the *positional weight*, which is the percentage of qualified path-term pairs found during the containment filtering or bitwise *anding* operation, and is used to calculate the weight of the path-term pair.

Definition 3 ***Positional Weight of a Path-Term Pair***. *Let D be an XML document, $PF_0^D(p,t)$ be the positional filter associated with (p,t), and $PF^D(p,t)$ be the result from containment filtering or bitwise anding operation. In addition, let $N_{PF_0^D(p,t)}$ be the number of one-bit ranges in $PF_0^D(p,t)$ and $N_{PF^D(p,t)}$ be the number of one-bit ranges in $PF^D(p,t)$. The positional weight of (p,t) in D is defined as:*

$$PW^D(p,t) = \frac{N_{PF^D(p,t)}}{N_{PF_0^D(p,t)}} \tag{6.5}$$

To illustrate the calculation of the positional weight of a path-term pair, we refer to our running example. In Figure 5.1, before the containment filtering, there are 7 one-bit ranges in F_2. After the containment filtering, 5 one-bit ranges remain in A. Similarly, before the bitwise anding operation, there are 5 and 4 one-bit ranges in $H_3[\text{“mountain”}]$ and $H_3[\text{“bicycle”}]$ respectively. After this operation, 3 one-bit ranges are left. Thus, the positional weight of each path-term pair is calculated as follows:

PW^D(/auction/item/location, "Dallas")=5/7=0.71

PW^D(/auction/item/description, "mountain")=3/5=0.6

PW^D(/auction/item/description, "bicycle")=3/4=0.75

Combining the *TF* score, *IDF* score, and the positional weight, the definition of the weight of (p, t) in D is determined by the following equation:

$$W^D(p,t) = PW^D(p,t) \times TF^D(p,t) \times IDF(p,t) \tag{6.6}$$

Finally, we give the definition of the enhanced content score of D relevant to Q.

Definition 4 ***Enhanced Content Score of a Document***. *Let Q be the query and D be an XML document. Let $W_i^Q(p_i^Q, t_i^Q)$ be the weight of the path-term pair (p_i^Q, t_i^Q) in Q and $W_i^D(p_i^D, t_i^D)$ be the weight of the corresponding path-term pair (p_i^D, t_i^D) in the document D. The content score of D relevant to Q is defined as*

$$ECS(D,Q) = \frac{\sum_{i=1}^{n} W_i^Q(p_i^Q, t_i^Q) \times W_i^D(p_i^D, t_i^D)}{\sqrt{\sum_{i=1}^{n} W_i^Q(p_i^Q, t_i^Q)^2} \times \sqrt{\sum_{i=1}^{n} W_i^D(p_i^D, t_i^D)^2}} \tag{6.7}$$

where n is the number of path-term pairs.

In order to calculate the $TF * IDF$ score of a path-term pair, we construct two additional synopses named **TF Synopsis** and **IDF Synopsis**. When a document D is indexed, for each text label path p in the structural summary of D, a *TF synopsis* is constructed to summarize the frequencies of all the terms associated with p in D. More specifically, a *TF synopsis* is a hash table that maps a term to a pair consisting of **DistinctTerms** that indicates the number of distinct terms in D reachable by p, and **Frequencies** that counts the total number of term occurrences reachable by p in D. In our implementation, a *TF synopsis* is attached to the content synopsis associated with p. To obtain the TF score of the path-term pair (p, t) during query evaluation, path p is used as the key to retrieve the corresponding content synopsis.

Then term t is hashed into some bucket over the attached *TF synopsis*, using the value of **Frequencies/DistinctTerms** as the term frequency of (p, t). To achieve IDF score of a path-term pair, for each text label path p, we construct an *IDF synopsis*, which contains two members named **Documents** and **Synopsis**, where **Documents** counts the total number of documents containing the path p, and **Synopsis** is a hash table that maps the term t to the total number of documents containing (p, t). *IDF synopses* have to be updated as new related XML documents are indexed on the server.

6.3 Aggregated Scoring with Term Proximity

We incorporate term proximity into the scoring to further improve the ranking scheme. Since XML data is a hierarchical tree, term proximity not only must be based on the distance of the terms in the document but should be based on the depth distance between the terms as well. Here we use the size of the *lowest common ancestor* (LCA) of the full label paths derived from structural summary matching to measure the depth term proximity.

Definition 5 ***Depth Term Proximity****. Let Q be the query and D be an XML document. Let (p_i^D, t_i^D), $1 \le i \le n$, be a matching path-term pair in D. Then r_{lca} is the root of the tree rooted at LCA of all paths p_i^D. Let $DIST(p_i^D, r_{lca})$ be the number of steps between the leaf node of p_i^D and r_{lca}. The* depth term proximity *of D relevant to Q is defined as*

$$DTP(D, Q) = \frac{1}{\sum_{i=1}^{n} DIST(p_i^D, r_{lca})} \tag{6.8}$$

Since it is possible that two terms are very close in the tree rooted at the LCA of the paths in a path combination, but very far in the actual document, we have to take the actual distance of the terms in the document into account, which is measured by *width* *term proximity*. At the end of our query processing, for each qualified document, a non-zero positional filter PF is derived after the Containment Filtering. Each one-bit range in PF represents an element that is associated with the PF entry in the query footprint. The smaller the size of this element, the closer the search terms in the document. In addition, more one-bit ranges in PF indicates that the document contains more qualified elements, so the document should be ranked higher. Thus, we use the average length of one-bit range and the number of one-bit ranges in the PF to measure the width term proximity.

Definition 6 ***Width Term Proximity***. *Let Q be the query and D be an XML document. Let PF be the final positional filter after the containment filtering. Let $N_{obr}^{PF}(D,Q)$ be the total number of one-bit ranges in PF and $L_{avg}^{PF}(D,Q)$ be the average length of one-bit ranges in PF. The* width term proximity *of D relevant to Q is defined as*

$$WTP(D,Q) = \frac{N_{obr}^{PF}(D,Q)}{L_{avg}^{PF}(D,Q)} \tag{6.9}$$

The final score of the document D is determined by the following equation:

$$S(D,Q) = ECS(D,Q)^{\alpha} \times DTP(D,Q)^{\beta} \times WTP(D,Q)^{\gamma} \tag{6.10}$$

where α, β, and γ are experimental parameters.

6.4 Experimental Evaluation

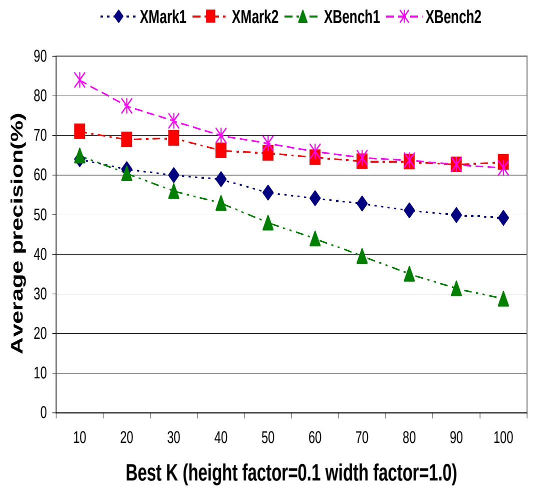
6.4.1 Effectiveness of Content Scoring

To examine the effectiveness of our ranking function, we used two widely accepted metrics, precision and recall. We first measured our ranking scheme based on content similarity by setting the parameters in Equation 6.10 to the following values: $\alpha = 1$, $\beta = 0$, and $\gamma = 0$ and fixing the positional weight to 1 in Equation 6.6.

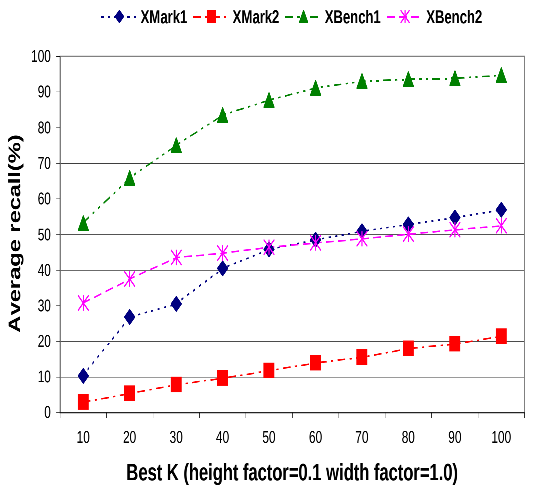
Then we incorporated positional weight and term proximity into the ranking function to demonstrate the improvements of ranking quality. Note that our two data sets from XBench were generated from Text-Centric Multiple Document (TC/MD) class, which are very suitable for our ranking measurement. We evaluated all the XMark (XBench) queries in Table 5.1 over each XMark (XBench) data set to measure the average precision and recall. To construct the accurate relevant set for each query, we exploited Qizx/open [47] to evaluate the query over each data set to obtain the strict relevant set. In another word, we put a document in the relevant set only if the document can exactly satisfy the query. Note that in the following ranking-related figures,the **width factor** is the ratio between the width of a content synopsis and the number of terms the associated SS node contains in a document. The **height factor** is the ratio between the height of a content synopsis or positional filter and the number of begin/end tags in a document.

6.4.1.1 Varying bestK Value

For these experiments, we first fixed the height and width of data synopses and varied the bestK value, i.e., the number of documents returned, to measure the average precision and recall of a query. Note that the meaning of bestK value is different from that in the classical topK algorithms, such as Fagin algorithm [49]. The bestK value here is just the number of returned documents for relevance ranking. The results in Figure 6.1(a) show that as the bestK value varies from 10 to 100, the average precision of a query over each data set drops smoothly, which indicates that our scoring function can effectively rank the relevant documents on the top of the ranked list so that the precision does not drop too much when the number of returned documents increases. Since the average size of the relevant set for a query over each XBench data set is relatively small, the average precision of a query over each XBench data set is a little lower. Figure 6.1(b) shows the recall over XMark1 is greater than that over XMark2 and the recall over XBench1 is larger than that over XBench2. The reason is that when the number of documents increases, the size of relevant set increases, which leads to a lower recall.



(a)

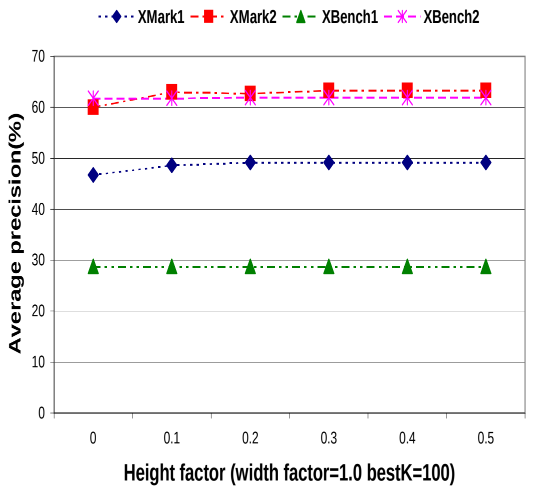


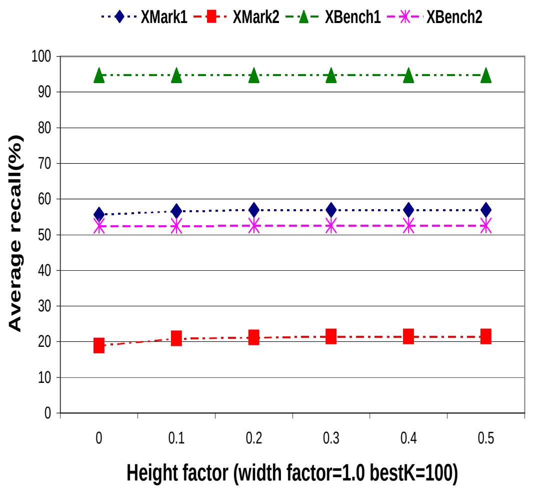
(b)

Figure 6.1. Varying BestK Value (a) Average Precision (b) Average Recall.

6.4.1.2 Impact of Height Factor

In the second set of experiments, we fixed the bestK value and the width of data synopses to see the impact of different height factors on precision and recall. As we can see from Figure 6.2(a) and Figure 6.2(b), as the height factor varies from 0.1 to 0.5, the precision and recall almost remain at the same value for each data set. This was expected because when the height of data synopses is reduced, a query may get more false positives, but our ranking function can effectively rank the most relevant documents close to the top, while moving false positives near the bottom of the answer set so that the precision and recall almost do not change.





(a) (b)

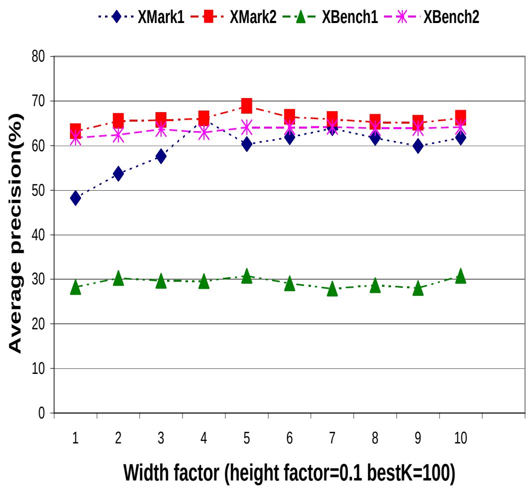
Figure 6.2. Impact of Height Factor (a) Impact of Height Factor On Precision (b) Impact of Height Factor On Recall.

6.4.1.3 Impact of Width Factor

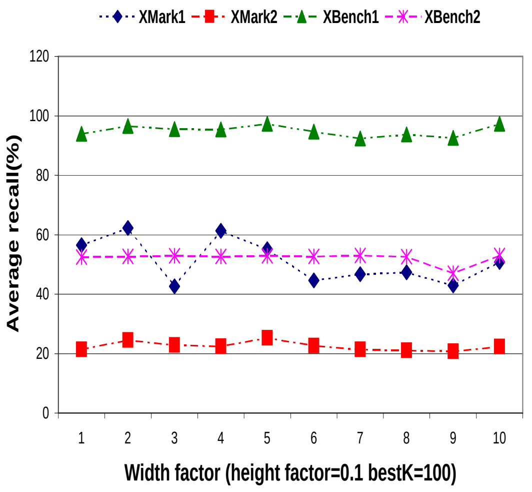
Finally, we fixed the size of the bestK value and the height of data synopses to see the impact of different width factors on precision and recall. The results are shown in Figure 6.3(a) and Figure 6.3(b). As we can see, the precision and recall change a little more than those in Figure 6.2(a) and Figure 6.2(b). This result implies that if we want to decrease the size of data synopses to reduce the data storage overhead but still keep high precision, the height factor should be adjusted first.

6.4.2 Effectiveness of Aggregated Scoring

In order to measure the effectiveness of aggregated scoring, we set the parameters in Equation 6.10 to the following values: $\alpha = 1$, $\beta = 1$, and $\gamma = 1$ and incorporate the positional weight in Equation 6.6 to measure the average precision and recall of a query. Note that since most XML ranking schemes in the literature [50, 20, 17] focus on ranking XML elements in original XML documents, rather than ranking XML



(a)



(b)

Figure 6.3. Impact of Width Factor (a) Impact of Width Factor On Precision (b) Impact of Width Factor On Recall.

documents based on data synopses, it is inappropriate to make direct comparisons with those ranking schemes. Instead, we fixed the number of returned documents to 50 and compared the three ranking schemes over our data synopses: Content Similarity Scoring (CS), Enhanced Scoring with Positional Weight (CS-PW) and Combined Scoring with Term Proximity (CS-PW-TP). We chose XMark2 and XBench2 as the data sets for this experiment. Figure 6.4 and Figure 6.5 show that for both data sets, the average precision (recall) of CS-PW-TP is higher than that of CS-PW, which is in turn higher than the precision (recall) of CS. Note that for XBench (XMark) data set, the average precision (recall) is a little lower because its relevant set is smaller (larger). The above results demonstrate that the combination of content similarity, positional weight and term proximity can improve the quality of ranking results effectively.

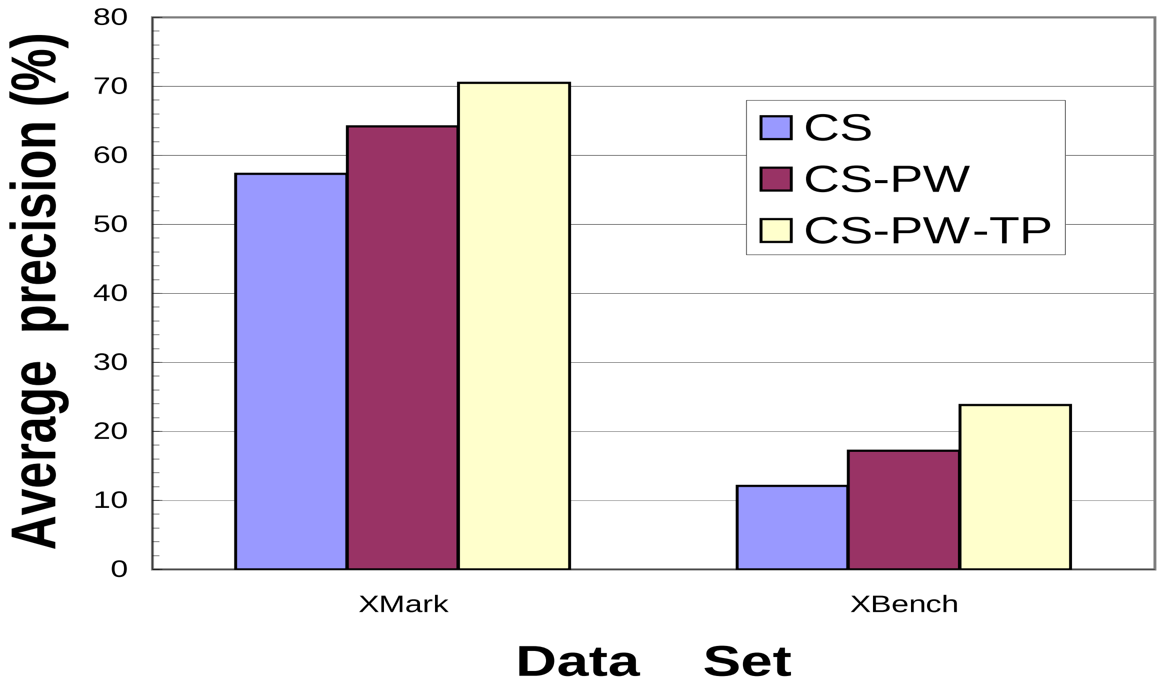


Figure 6.4. Average Precision.

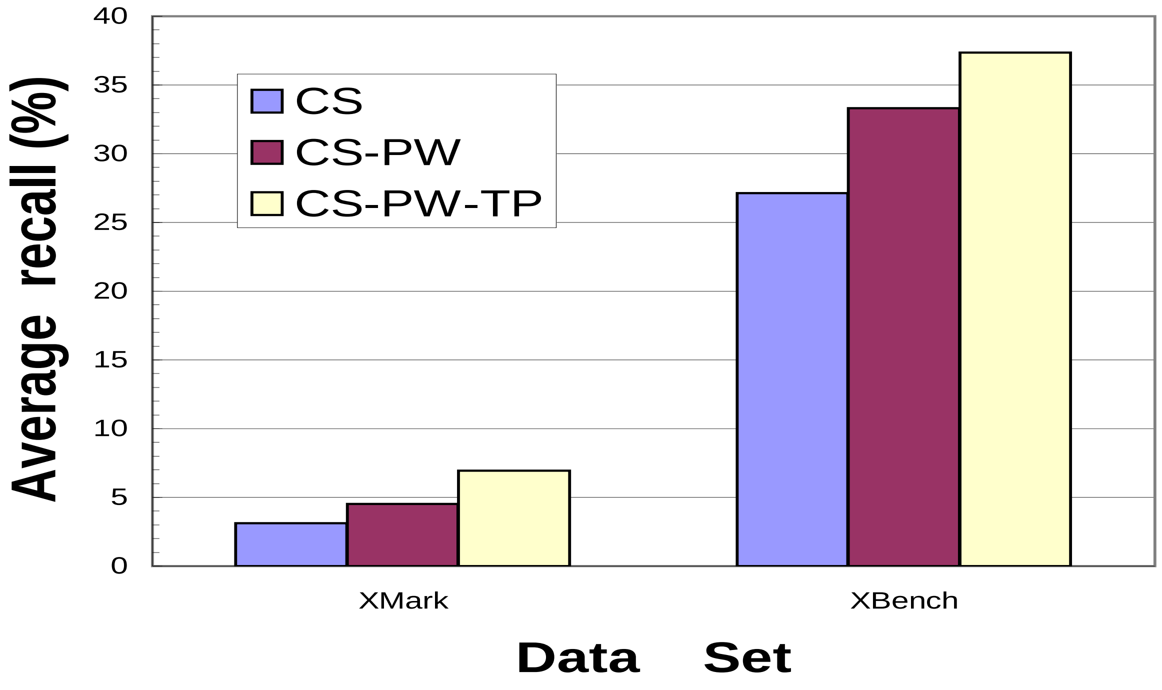


Figure 6.5. Average Recall.

6.4.3 Ranking Measurement Based on Ranked Relevant Set

In order to further demonstrate the effectiveness of our relevance ranking scheme, we constructed a ranked relevant set to evaluate the relevance ranking scheme. We exploited Qizx/Open [47] to construct the ranked relevant set for each query. To be more specific, for a given query Q and the data set, we evaluated Q over each XML document D in the data set. If D matches Q, D will be put in the ranked relevant set. The score of D is determined by the number of elements returned when Q is evaluated over D. The larger the number of returned elements, the higher the score of document D. Then we fixed the top k value to 10 and evaluated 20 queries over indexed meta-data and measured the precision for each query in top-10 relevant set. The precision is calculated based on the top-10 relevant set instead of the whole relevant set. We evaluated the ranking scheme on both XMark and XBench data sets. The experimental results for XMark data set are shown in Figure 6.6. As we can see from Figure 6.6, the maximum precision is 1.0 and the minimum precision is 0.1 among queries from Q1 to Q10. The average precision is 0.47, which indicates that on average, 4.7 answers out of 10 in the answer set are in the top-10 relevant set. The experimental results for XBench data set are shown in Figure 6.7. As we can see from Figure 6.7, the maximum precision is 1.0 and the minimum precision is 0 among queries from Q11 to Q20. The average precision is 0.39, which indicates that on average, 3.9 answers out of 10 in the answer set are in the top-10 relevant set. For queries Q11, Q13, and Q16, the precision is zero. The reason is that for each of these queries, the relevant documents in the relevant set are only 3 , which is much less than 10 and thus leads to zero precision value. The above experimental results demonstrate that the proposed relevance ranking scheme is effective.

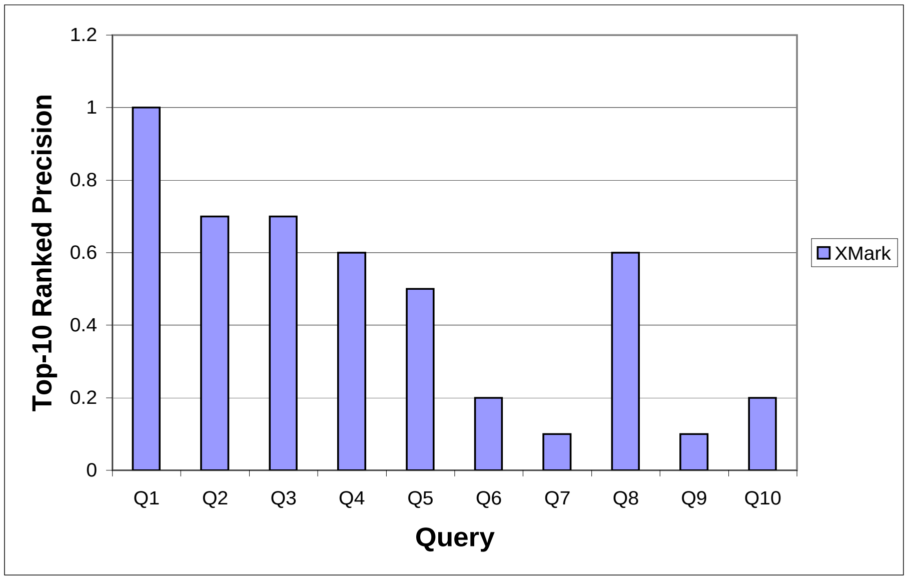


Figure 6.6. Top-10 Ranked Precision for XMark.

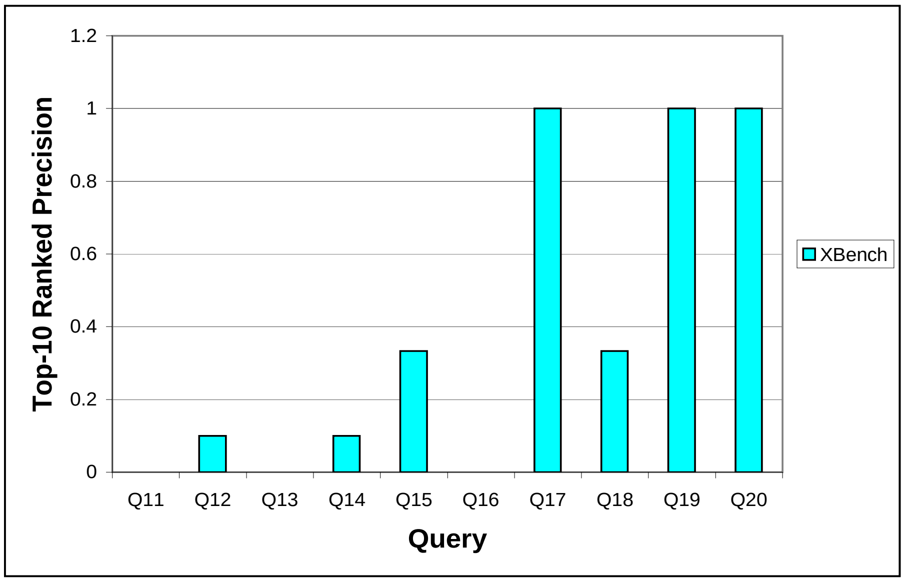


Figure 6.7. Top-10 Ranked Precision for XBench.

CHAPTER 7

EXTENSION TO PEER-TO-PEER NETWORKS

Due to the popularity of peer-to-peer (P2P) computing in the past few years, XML data management in P2P environments has attracted significant attention in the database community. In this chapter, we extend our framework to DHT-based structured P2P networks. We first give a brief introduction to the background of DHT-based structured P2P networks. Then, we overview the system architecture of a peer in our framework. Following that, we present our data placement strategies in a distributed hash table and our query evaluation in structured P2P networks. We also propose effective schemes to handle network updates and load balancing in our system. Finally, we present our experimental results.

7.1 Preliminaries of DHT-Based P2P Networks

7.1.1 Introduction to Distributed Hash Table

Distributed hash tables (DHTs) are a class of decentralized distributed systems that provide a lookup service similar to a hash table: (name, value) pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given name. Responsibility for maintaining the mapping from names to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows DHTs to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures [51].

DHTs form an infrastructure that can be used to build more complex services, such as distributed file systems, peer-to-peer file sharing and content distribution systems, cooperative web caching, multicast, anycast, domain name services, and instant messaging. Notable distributed networks that use DHTs include BitTorrent, eDonkey, and eMule [51].

DHT research was originally motivated, in part, by unstructured peer-to-peer systems, such as Napster, Gnutella, and Freenet, which took advantage of resources distributed across the Internet to provide a single useful application. In particular, they took advantage of increased bandwidth and hard disk capacity to provide a file sharing service [51].

DHTs characteristically emphasize the following properties [51]:

- Decentralization: the nodes collectively form the system without any central coordination.
- Scalability: the system should function efficiently even with thousands or millions of nodes.
- Fault tolerance: the system should be reliable (in some sense) even with nodes continuously joining, leaving, and failing.

A key technique used to achieve these goals is that any one node needs to coordinate with only a few other nodes in the system, most commonly, log(n) of the n participants, so that only a limited amount of work needs to be done for each change in membership [51].

7.1.2 Data Placement in DHT-Based P2P Networks

In a DHT-based structured P2P system, the location of data is determined by some global scheme. More specifically, as a data item I is published by some node, a global hash function is used to map the key K of I to an IP address of a node N in the P2P network and I is placed on node N. One popular mapping approach is called consistent hashing [11], in which both a key and node Id are mapped to the same identifier space. Identifiers are ordered in an identifier circle modulo $|2^m$, where m is the length of an identifier. The data item I associated with the key K is assigned to its successor node, which is the first node whose identifier is equal to or follows the identifier of K in the identifier space. In our framework, we employ Pastry [12] as our P2P back end, in which the key K is assigned to the node whose Id is closest to the Id of the key. The Id of K is derived from a base hash function such as SHA-1 [11].

7.2 Peer Architecture

A peer in our framework can make any number of its local XML documents public to the other peers through the process of *publishing*. Once published, an XML document is available to any peer for querying, until is removed by the document owner through the process of *unpublishing*. Our P2P network serves as a distributed index for routing queries to the peers who own the documents that can answer the queries. Document updates are done by unpublishing the entire document and publishing it again (as is done in most IR systems), rather than updating the distributed index to reflect the document updates.

The architecture of a peer in our framework is shown in Figure 7.1. The XML document pool is a repository of local XML documents published by this peer. The structural summary indexes and the persistent data synopses are stored locally using a lightweight storage manager (Berkeley DB [42]). There are three main components in a peer: the publisher, the plan evaluator, and the XQuery processor. When a peer publishes a local document, its publisher sends its structural summary and data synopses to the appropriate peers using messages. When a peer receives such a message, it stores this information into its local indexes. The plan evaluator is the most important component of the peer. It evaluates the incoming plan against both the incoming list of document locations and its local database and generates a new

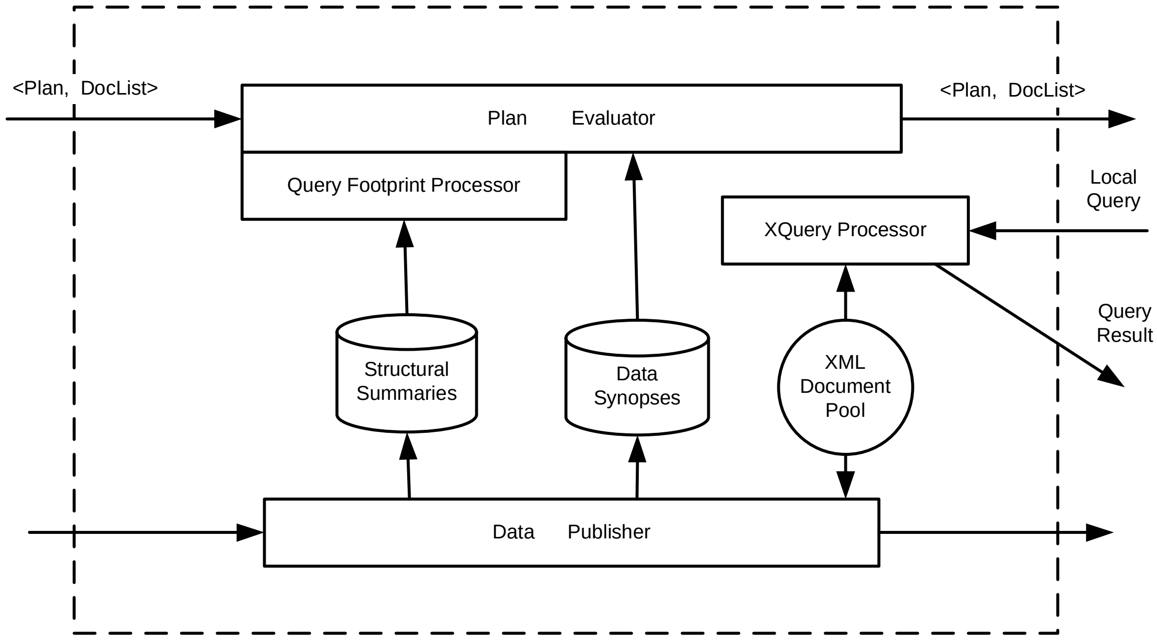


Figure 7.1. Architecture of a Peer.

plan and a new list to be routed to the next peer. It basically implements a single step towards the evaluation of a query. After the final document locations of a query are retrieved by the plan evaluators, they are returned to the query client. The query client may pick some of these document locations to send the original XML query for evaluation. This is accomplished by the document owner's centralized XQuery processing engine that supports full-text search against its local documents. The results of the XQuery engine are the actual XML fragments that satisfy the query, which are ranked and routed back to the query client. This engine is typically a native XML storage manager that indexes local XML documents using inverted lists of tagnames and keywords.

Although our framework is independent of the underlying DHT-based P2P architecture, our system is implemented on top of Pastry [12]. Pastry is a completely decentralized, scalable and self-organizing substrate for P2P applications. It maps both keys and node IP addresses to the same identifier space using 128-bit Node Ids. A key is mapped to a peer node whose Node Id is the closest to the key's Id. In contrast to other DHT-based systems, the query routing scheme in Pastry is more complex and more efficient since it considers network proximity to make the number of physical hops of message routing as small as possible. The most important operation in Pastry is route(msg,key), which routes a message, msg, to the peer whose Node Id is numerically closest to key. It requires at most $log(n)$ hops from peer to peer, for n peers. Although Pastry updates the peer routing tables automatically to cope with node departures and failures, it is left to the application to provide a suitable data storage for each peer and a method to redistribute the data among peers in the event of a DHT update.

7.3 Data Placement

One of the contributions in this thesis work is in using the data synopses to route queries to peers who are likely to own documents that satisfy the query. To accomplish this, we introduce methods for placing and indexing data synopses over a P2P network and for locating documents relevant to a query based on these indexes. Our placement strategy for structural summaries is very simple: they are routed to peers using every distinct tagname from the structural summary as a routing key. Thus, locating all structural summaries that match the structural footprint of a query is accomplished by a single DHT lookup by choosing a tagname uniformly at random from the query footprint as the routing key. A data synopsis is placed on the P2P network using its label path as the routing key. Since a label path may belong to multiple documents, all the relevant synopses from all these documents are placed at a single peer, the one whose Node Id is numerically closest to the Id of the label path. Thus, locating all document locations that satisfy the simple query p $\sim$"term", for a label path p, is accomplished by a single DHT lookup by using the label path p as the routing key. Then, the document locations that satisfy this simple query are those whose content synopses, projected over "term", give a non-zero positional filter.

When a peer N publishes an XML document D, it extracts its structural summary S and assigns a locally unique number D_N to D. Then, for each distinct tagname *tag* in S, it inserts the mapping from *tag* to S into the distributed hash table (DHT) of the P2P network. This is accomplished by sending a message with key, hash(*tag*), which is the hash code of *tag*. This message is received by the peer whose Node Id is closest to the Id of the message key and is stored in the peer's local database. One improvement to this process is having the publisher pick a tagname uniformly at random from the structural summary as the routing key of a single message and wait for a reply to this message. If the reply indicates that the summary has already been published, then the publisher will be kept from routing the remaining tagnames.

Each content synopsis H_p^D and each positional filter F_p^D of D is sent via a message with key hash(p), where p is the label path. In summary, the total number of messages needed to publish a document depends on the size of its structural summary S only, since it is equal to the total number of distinct tagnames in S and the number of nodes in S.

7.4 XPath Query Routing

When a client peer submits a query over the P2P network, it extracts its structural footprint Q and chooses a tagname uniformly at random from Q (the tagname must be selected uniformly at random for better load balancing). For the running query Q, assume that the chosen tagname is "item". Then the client peer sends a single message whose key is the chosen tagname, that is, the Id derived from

```
route( “⟨item⟩”, ∅,
      $\lambda L_0$. for each $[p_0, p_1, p_2]$ in $\mathcal{P}[\![$//auction//item:0[location:1][description:2]/price$]\!]$
            route( $p_1$, ∅,
                  $\lambda L_1$. route( $p_2$, cf( $\mathcal{M}_F(p_0)$, find( “Dallas”, $p_1$ ) ),
                              $\lambda L_2$. return( cf( $L_2$, and( find( “mountain”, $p_2$ ), find( “bicycle”, $p_2$ ) ) ) ) ) ) )
```

Figure 7.2. The Query Plan of Query Q.

hash(“item”) for the example query, which contains the query evaluation plan. This message is received by the peer whose Node Id is closest to the Id of this key. The peer who receives this message, called the *query evaluator*, is responsible for dispatching parts of the query plan to other peers. Upon receiving the query, the query evaluator uses its local database to find all possible distinct label paths for the footprint entry points. Even though there may be numerous matching structural summaries, the number of distinct label path combinations is typically small.

The query plan for running query Q sent to the query evaluator is shown in Figure 7.2. The unit of communication between the peers that evaluate parts of the query plan is a *hit list* L, which is a set of triples (F, N, D), where F is a positional filter of size $M \times L^D$ associated with a document published by the peer N, which has been assigned the number D by N. The positional filter F indicates which positions in the document D satisfy the query at the current point of query evaluation. It may be refined further or completely eliminated when more search specifications and containment constraints are evaluated during the query evaluation.

The expression route$(p, L, plan)$ sends a message to the peer who owns the data synopses associated with the label path p, that is, to the peer whose Id is closest to the key p. The message contains the current hit list L and the query plan, *plan*, to be executed at the destination. Each *plan* is an abstract syntax tree that, when evaluated at the destination site, it binds the variable x to L and evaluates the plan e under this binding. Note that the *plan* is evaluated at the destination site.

The expression find(w, p) returns a hit list by projecting $\mathcal{M}_H(p)$ over the column associated with the word w:

$$\text{find}(w,p) \;=\; \{\ (H_p^D[w], N, D) \mid (H_p^D, N, D) \in \mathcal{M}_H(p)\ \}$$

where $H_p^D[w]$ projects H_p^D over the column associated with term w and repeats this column M times to construct a positional filter. The function 'and' over hit lists is basically a bitwise operation between their positional filters:

$$\begin{aligned}\text{and}(L_1, L_2) \;=\; \{\ (\text{AND}(F_1,F_2), N, D) \mid\ & (F_1, N, D) \in L_1 \\ \wedge\ & (F_2, N, D) \in L_2 \\ \wedge\ & \text{AND}(F_1, F_2) \neq \emptyset\ \}\end{aligned}$$

where the function 'AND' is a bitwise 'and' operation. Function 'cf' over hit lists is a containment filtering CF between their positional filters (defined in Section 5.4):

$$\begin{aligned}\text{cf}(L_1, L_2) \;=\; \{\ (\text{CF}(F_1,F_2), N, D) \mid\ & (F_1, N, D) \in L_1 \\ \wedge\ & (F_2, N, D) \in L_2 \\ \wedge\ & \text{CF}(F_1, F_2) \neq \emptyset\ \}\end{aligned}$$

Function return(L) returns the hit list L back to the query client. We can see that, for each matching label paths p_0, p_1, and p_2 at the query evaluator site, the query plan in Figure 7.2 is dispatched to 3 peers. They basically evaluate the expression:

cf($\mathcal{M}_F(p_0)$,

    cf(find("Dallas", p_1),

        and(find("mountain", p_2), find("bicycle", p_2))))

in a distributed fashion.

7.5 Handling Network Updates

There are three types of network updates that need to be handled by any P2P system: arrival, departure, and failure of nodes. While it is very important to maintain the integrity of the routing indexes, the integrity of data, which in our case are document references, is of secondary importance, as is apparent in web search engines that may return outdated links to documents.

Both arrivals and departures can be handled without disrupting the query routing process. When a new node joins the overlay network and is ready to send and receive messages, it invokes the Pastry method notifyReady(). Our implementation of this method includes code that sends a message to the new node's successor to transport parts of its database to the new node. The node successor is derived from the node's Leaf set and is basically the immediate neighbor in the Id space with a larger Id. When the successor receives the message, it moves all structural summaries and data synopses whose routing Ids are less than or equal to the new node's Id to the new node. Therefore, the arrival of a new node requires two additional messages to transport data to the new node. When a node willingly leaves the overlay network, it routes all structural summaries and data synopses to its successor using one message only. The departing peer also has the choice of unpublishing its local documents, which is very costly, since it involves sending a large number of messages to the owners of data synopses. The alternative, which we adopt in our framework, is to leave the references to the local documents dangling and let the system remove them lazily (as described below).

A node failure is the most difficult network update to handle. When a peer P_1 receives a search request based on a tagname tag_1 to find all structural summaries that match a query footprint and does not find one, there are two possibilities: either the query was incorrect and there was really no matching structural summary indexed in the DHT, or the predecessor node, who was closest to tag_1, had failed. Given that P_1 knows when its predecessor in the Id space fails (since, when this happens, it will receive a special message from Pastry), it can always distinguish the two cases: if the tagname Id is smaller than the failed predecessor Id, then it is the latter case. In that case, P_1 will choose another tagname tag_2 uniformly at random from the query footprint and relay the search request to another peer P_2 under the new key tag_2. In addition to the message relay, P_1 sends another message to P_2 asking to return all structural summaries associated with tag_1 to be published in P_1 (since P_1 now gets the requests for tag_1 keys). That way, the structural summaries of the failed peer are republished one-by-one lazily and on demand. Similarly, when a peer gets a request for a data synopsis based on the label path p and does not find one, and its predecessor had failed, it will check whether the Id from p is less than the Id of the failed predecessor. If so, it will abort the query and will scan the list of document publishers from the hit list routed along with query and will send a message to each publisher to publish the data synopses for path p again. Therefore, the restoring of data synopses associated with a failed peer in the P2P network is done lazily and on demand, and each time only one query has to be aborted.

When a peer departs or fails, all references to its local documents become dangling pointers until the peer joins or becomes alive again. Although data integrity in a P2P database is not as important as in traditional databases, eventually these references must be removed with the smallest effort possible to preserve freshness of data. Removing outdated document references can be accomplished by attaching expiration dates to the data synopses and, when expired, by probing the publishers on the status of the published documents. This is a pull-based data refreshing method, to be contrasted with a push-based one in which the publisher probes the storage nodes periodically [38].

7.6 Load Balancing

From our data placement policy, we can see that data synopses are better distributed on the P2P network than structural summaries. Structural summary distribution is purely based on single tagnames, which implies that peers associated with popular tagnames, such as name and price, will unfairly burden a higher load of structural summaries than others and will have to handle more messages at publication and query times. On the other hand, data synopses are better distributed since their DHT keys are based on label paths, which are tagname sequences.

One way to improve load balancing in structural summary indexing and routing is to use a sequence of two tagnames (A, B) as a key, where B is a descendant of A (but $A \neq B$) in the structural summary. That is, a structural summary is published using all possible pairs (A, B) for a descendant B of A, which requires $\mathcal{O}(n^2)$ messages for n tagnames, rather than n messages. Then, given a query footprint with at least two tagnames, two tagnames A and B are selected uniformly at random from the query footprint and used as a routing key, so that $A \neq B$ and B appears to be a descendant of A in the query. This approach has a higher publication overhead but gives better balancing on both data distribution and query evaluation. In [27], a similar approach has been proposed for load balancing, but it uses parent-child tagnames as keys which, although requires $\mathcal{O}(n)$ messages for publishing, it makes query processing very hard since it requires that a query has a A/B (parent-child) component.

Another improvement to load balancing is to slice the content synopses into slices of constant width W, by dividing the width W_p^D of a content synopsis into $\lceil W_p^D/W \rceil$ partitions. Instead of routing a content synopsis H_p^D to a single peer using the key p, each partition i of H_p^D is routed to a peer based on the key (p, i), which is derived by xoring the SHA-1 hash codes of p and i. This improvement does not affect

Table 7.1. Characteristics of Data Sets

	files	total size	avg file size	tags	label paths
XMark 1	230	1.7 MBs	7.6 KBs	83	341
XMark 2	2300	17 MBs	7.6 KBs	83	424
XMark 3	11500	82 MBs	7.3 KBs	83	436
XMach	5000	87 MBs	17.8 KBs	1616	9206

XPath query processing because, given a term in a query, we simply extract a single column from H_p^D based on the term hash value, which can be done using one peer lookup in either case. This approach though complicates XQuery processing because the evaluation of each join between two documents requires that the entire content synopses be present, which basically means that all slices must be retrieved.

7.7 Experimental Evaluation

We have built a prototype system to test our P2P framework. It is built on top of Pastry [52] and uses Berkeley DB Java Edition [42] as a lightweight storage manager. The platform used for our experiments was a Intel Core 2 Quad Processor 2.66GHz with 4GB memory on a PC running Linux. The simulation was done using Java (J2SE 6.0) with a 768MBs maximum memory allocation pool. The experiments were performed over a cluster of 100, 1000, and 2000 peers in a simulated network on a single processor. We used four data sets for our experiments, which were synthetically generated by the XMark [43] and XMach [53] benchmarks. Characteristics of data sets are described in Table 7.1.

Table 7.2. Scalability Measurements

	msgs/file	synopses/file	msgs/query
XMark 1	41.7	17.9	3.93
XMark 2	41.3	17.6	3.63
XMark 3	41.4	17.6	3.87
XMach	28.9	17.7	3.52

While a single DTD was used for the XMark data sets, 320 different DTDs were used for the XMach data set so that there were between 2 and 100 documents per DTD.

Our query workload consisted of 1000 random XPath queries generated from 50 files selected uniformly at random from each data set. More specifically, each selected file was read using a DOM parser and was traversed using a random walk, generating an XPath step at each node (by randomly selecting one of the steps: `/tag`, `//tag`, `/*`, `//*`, skipping the node, using a predicate, or using a search specification). When a text node was reached and a search specification was selected, search terms were selected from the text uniformly at random. That is, each generated query was constructed in such a way that it satisfied at least one document (the parsed document). An example of a medium size random query for XMark is:

```
//namerica/item[*[text~"flew"]/text~"proposes"]/*
```

Based on the four data sets described in Table 7.1 and the above query workload, we first derived the measurements based on data synopses and query messages shown in Table 7.2, where *msg/file* is the average number of messages needed to publish one file from the data set, *synopses/file* is the average number of data synopses produced by each file in the data set, and *msgs/query* is the average number of messages needed to evaluate one query from the query workload. The data shown in Table 7.2 demonstrate that our system is scalable in terms of data publishing and query efficiencies.

Based on the XMark data sets and the query workload (1000 random queries), we measured the load distribution for a network of 100, 1000, and 2000 peers. The results are shown in Figure 7.3. More specifically, for each one of the 3 XMark data sets, we grouped and counted the peers based on

1. the distribution of the number of messages needed to publish all the documents in a data set;
2. the distribution of the number of content synopses after all the documents in a data set have been published;
3. the distribution of the number of messages needed to evaluate 1000 randomly generated queries.

For example, out of 100 peers, 13 receive between 1 and 40 messages and 11 receive between 41 and 80 messages during the publishing of the data set XMark 1. These results can be easily explained given that the documents generated by XMark match a single DTD, resulting to a very small number of distinct tagnames and text label paths. For instance, from a network of 2000 peers, at most 436 peers are expected to receive all data synopsis placement/search requests, while the rest get none. For a network of 100 peers, though, the load is more evenly distributed. This load balancing skew, however, is not likely to happen with a heterogeneous data set, when the number of distinct label paths is comparable to the network size. For example, the XMach data set, which uses multiple DTDs, gives a better load distribution in processing 1000 randomly generated queries, as shown in Figure 7.4. More specifically, out of 2000 peers, 52.2% receive between 1 and 15 messages and 47.1% receive no messages (while for XMark 3, 98% receive no messages, leaving the burden of query processing to the other 2%).

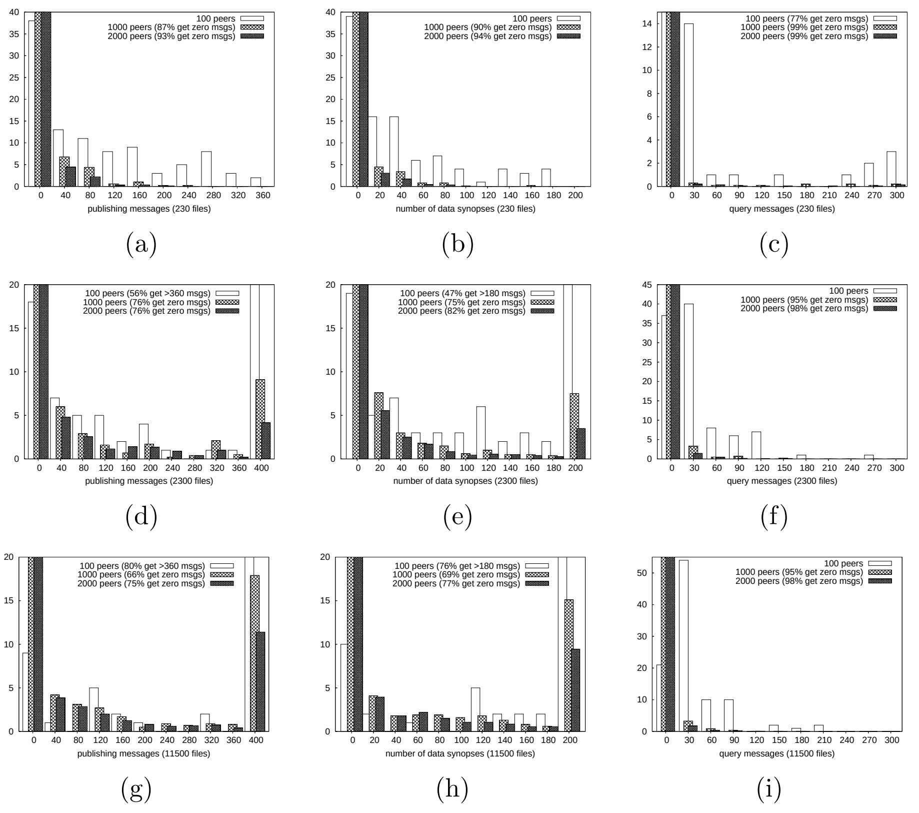


Figure 7.3. Load Distribution based on XMark Datasets (a) Distribution of Publishing Messages on XMark1 (b) Distribution of Data Synopses on XMark1 (c) Distribution of Query Messages on XMark1 (d) Distribution of Publishing Messages on XMark2 (e) Distribution of Data Synopses on XMark2 (f) Distribution of Query Messages on XMark2 (g) Distribution of Publishing Messages on XMark3 (h) Distribution of Data Synopses on XMark3 (i) Distribution of Query Messages on XMark3.

The second set of experiments was designed to measure the accuracy of data synopses. It was based on the XMark 1 data set on a single peer (since precision is not affected by the network size). The results are shown in Figure 7.5 and Figure 7.6. For the first data synopsis precision experiments, a special care was taken to generate queries that satisfy only one document from the data set. This was accomplished by always picking words from the text that appear on the selected document exclusively. This was done by extracting all the words that appear once only in the data set using the unix command `uniq -u`. Figure 7.5 and Figure 7.6 show the average number of

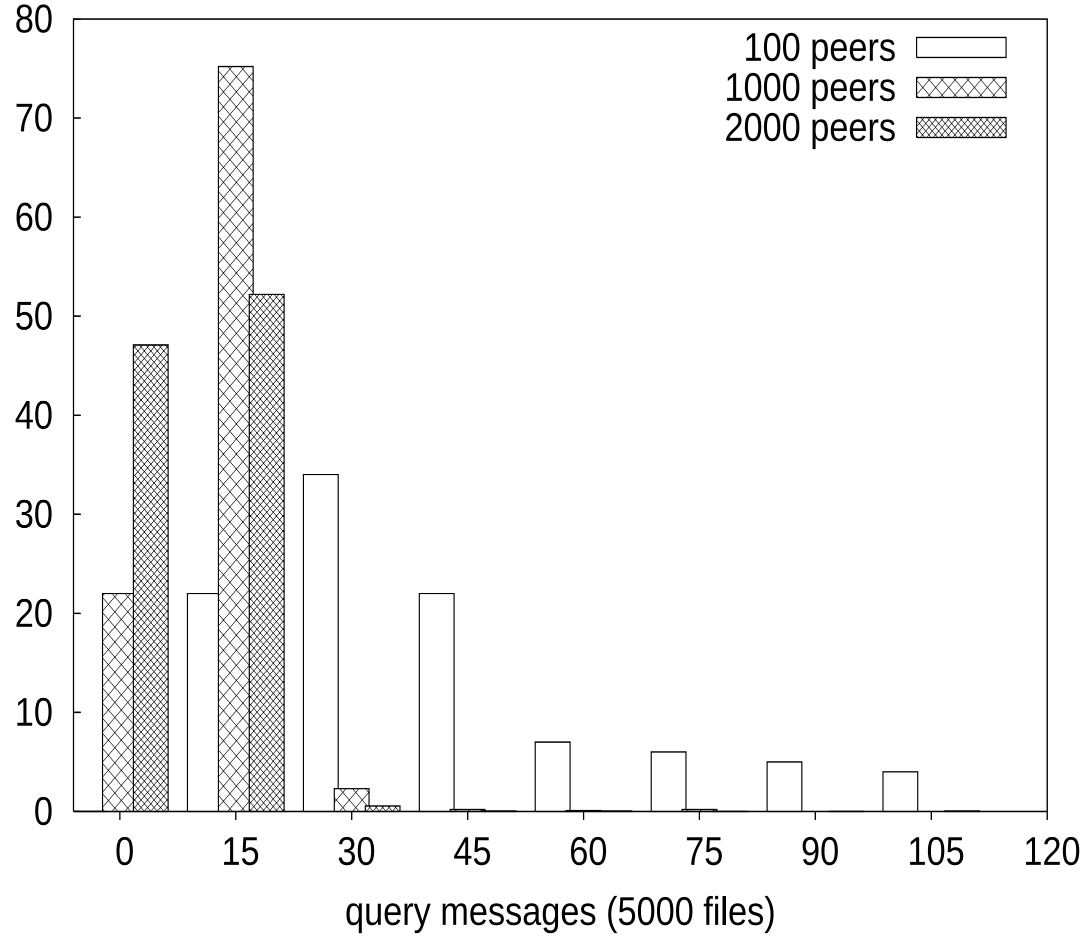


Figure 7.4. Query Load Distribution for XMach.

false positives for various sizes of data synopses. Given a label path and a document, the width of its content synopsis is the number of document elements reachable by this path multiplied by the width factor. The height factor, when multiplied with the document size, gives the content synopsis heights. When the height factor was set to zero, then Bloom filters were used instead of content synopses (zero height). The size M is the height of positional filters. When M=0, then no positional filters were used. From Figure 7.5 and Figure 7.6, we can see that, for random queries, the width factor affects precision more than the height factor (Ideally, the number of false positives should be zero.) However, to measure the precision improvement gained by using two-dimensional bitmaps and positional filters compared to plain Bloom filters, we had to generate a special set of random queries that, although ideally should have returned an empty result because of containment constraints on data, they return false positives. To generate these queries, we started with random queries of the form $p_1[p_2p_3 \sim s_1][p_2p_4 \sim s_2]$, for random paths p_i and search specifications s_1, s_2, that return a single document only (the parsed document – as above), so that the paths p_2 in the two predicates start from different children of p_1. Then, we transformed this query into the query $p_1p_2[p_3 \sim s_1][p_4 \sim s_2]$, which is guaranteed not to match any document. But with no positional filters, the latter query would return false positives since there is no way to check whether $p_3 \sim s_1$ and $p_4 \sim s_2$ start from different data nodes. Figure 7.6 indicates that, using plain Bloom filters (ie, with a zero height factor) yields twice as many false positives as when the height factor is ≥ 0.1.

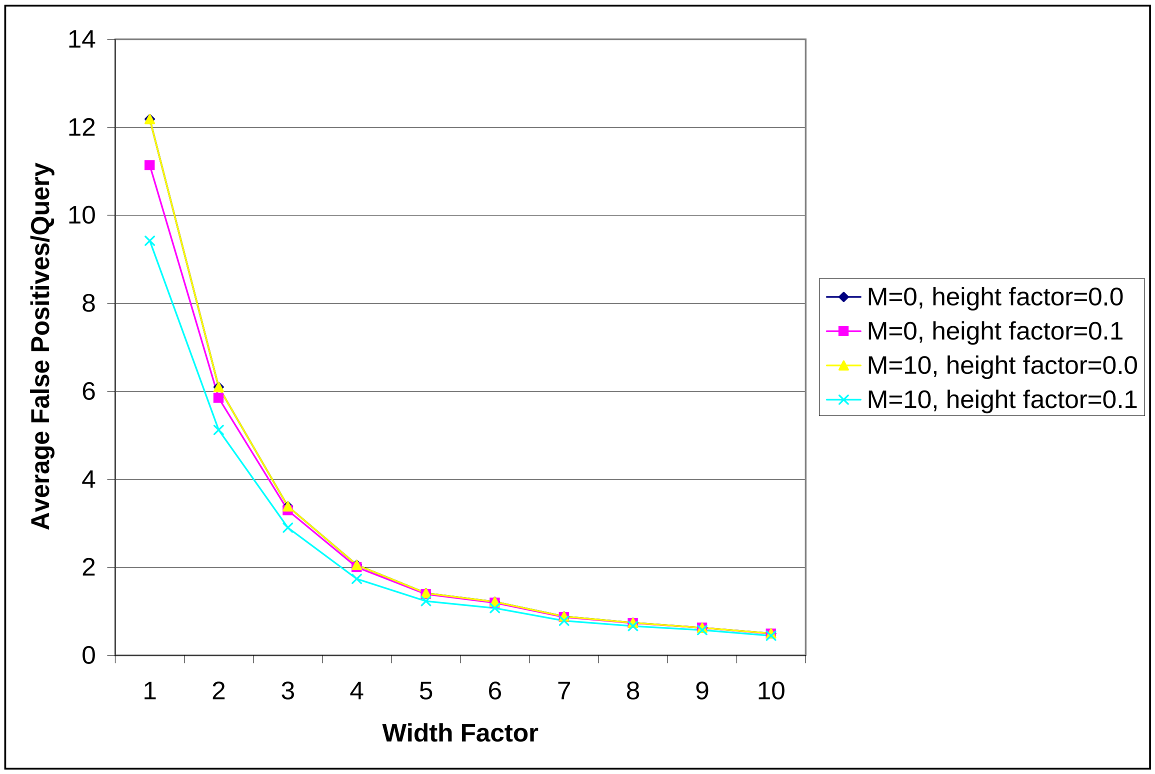


Figure 7.5. Data Synopsis Accuracy Varying Width Factor.

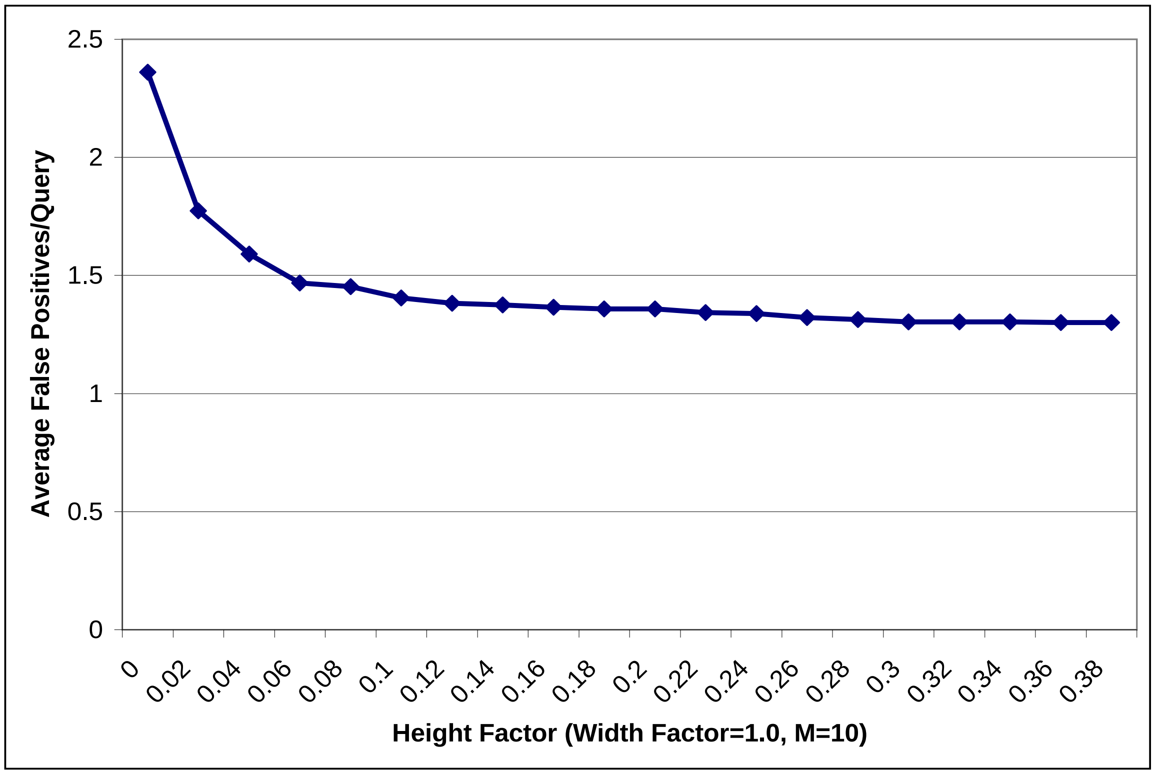


Figure 7.6. Data Synopsis Accuracy Varying Height Factor.

CHAPTER 8

CONCLUSION AND FUTURE WORK

In this thesis, we have presented a framework for efficient indexing and searching XML documents based on condensed summaries extracted from the structural and textual content of the documents. Our data synopses, which summarize the textual content of XML documents and correlate content with positional information, can result in a more accurate evaluation of textual and containment constraints in a query. Our two-phase containment filtering strategy can prune unqualified documents before the expensive join operations and thus accelerate the searching process. Our XML document ranking scheme, which aggregates content similarity, positional weight and term proximity and incorporates the scoring process into the query evaluation, can effectively improve the quality of ranking results. Finally, we extend our framework to DHT-based structured P2P networks. Our experiments show that our meta-data indexing scheme is more efficient than traditional indexing schemes based on full inverted lists. Our data synopses can achieve a lower false positive rate than ordinary one-dimensional Bloom filters. Our two-phase containment filtering algorithm is more efficient than the single-phase brute force algorithm. Our relevance ranking scheme is effective in terms of precision and recall. Our extension to DHT-based P2P networks is smooth in the sense that our meta-data indexing scheme is scalable and both data and query workload are appropriately balanced in the P2P networks.

Several future work can be envisioned in different aspects.

Relevance Ranking in Structured P2P Networks: Currently, our extension to structured P2P networks is limited to data indexing and query processing. Extending and adapting our $TF * IDF$ cost function to structured P2P networks is a potential challenging work to be explored. The main challenge here is that the calculation of the IDF score for a document hit usually requires global information, which is difficult to obtain in a decentralized P2P environment.

Top-K Processing Beyond Relevance Ranking: We also plan to explore the possibility of adopting some well-known top-K algorithms, such as TA algorithms, to further reduce the query processing time in our framework. This is an even more challenging and interesting work. For example, how to determine the m lists for top-K processing? XML data are nested and recursive data, rather than flat relational data, which makes it much more difficult to determine the stopping condition of a top-K algorithm.

Schema Summary-based Friendly GUI: Since our framework supports complex XPath queries with full-text search predicates, one challenging task for a user is to issue a meaningful full-text XPath query. However, the schema information of data is usually not available in a distributed P2P network. As such, it is better to provide some schema summaries to the user such that he/she can issue a query by navigating through the nodes in the schema summaries.

P2P Search Engine For Multimedia Data: With the popularity of P2P as a new generation of network infrastructure, a large number of users would like to share their multimedia data such as photos or video clips with others on the Web. For example, YouTube is a video sharing website where users can upload, view and share video clips. However, the data sharing of YouTube is still in a centralized and less autonomous mode. More importantly, its keyword-based search mechanism is not precise enough. We plan to adapt my meta-data indexing scheme to index distributed multimedia data, develop efficient query evaluation strategies over XML-encoded meta-data, and eventually build a more flexible and effective multimedia search engine in a P2P environment.

XML-Enabled Bioinformatics Data Retrieval: Nowadays, biologists would like to share biological data with their colleagues in different institutions. Most biological repositories provide only a simple keyword search mechanism over huge biological data to return all the XML documents that satisfy the keyword queries. For a keyword query, the whole XML document instead of a specific fragment in the document is returned to the user. We would like to extend my current framework to enable more precise and intuitive search of biological data.

REFERENCES

[1] G. Koloniari and E. Pitoura, “Peer-to-peer management of xml data: issues and research challenges,” *SIGMOD Record*, vol. 34, pp. 6–17, June 2005.

[2] W3C, “Xml path language (xpath) 2.0,” 2007. [Online]. Available: http://www.w3.org/TR/xpath20/

[3] ——, “Xquery 1.0: An xml query language,” 2007. [Online]. Available: http://www.w3.org/TR/xquery/

[4] Napster, “Napster,” 1999. [Online]. Available: http://www.napster.com/index.html?darwin=aladdinV2

[5] Gnutella, “Gnutella,” 2000. [Online]. Available: http://www.gnutella.com/

[6] ——, “Query routing for the gnutella network,” 2001. [Online]. Available: http://rfc-gnutella.sourceforge.net/src/qrp.html

[7] Freenet, “The free network project,” 2008. [Online]. Available: http://freenetproject.org/

[8] Y. Chawathe, S. Ratnasamy, and L. Breslau, “Making gnutella-like p2p systems scalable,” in *Proceedings of ACM SIGCOMM 2003*, 2003.

[9] M. Bawa, B. F. Cooper, A. Crespo, and N. Daswani, “Peer-to-peer research at stanford,” *SIGMOD Record*, vol. 32, pp. 23–28, Sept. 2003.

[10] S. Ratnasamy, P. Francis, M. Handley, and R. Karp, “A scalable content-addressable network,” in *Proceedings of ACM SIGCOMM 2001*, 2001.

[11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” in *Proceedings of ACM SIGCOMM 2001*, 2001.

[12] A. Rowstron and P. Druschel, “Pastry:scalable,dencentralized object location and routing for large-scale peer-to-peer systems,” in *Proceedings of IFIP/ACM International conference on Distributed Systems Platforms(Middleware2001)*, 2001.

[13] B. Y. Zhao, L. Huang, J.Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, “Tapestry: A resilient global-scale overlay for service deployment,” *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 41–53, Jan. 2004.

[14] W3C, “Xquery and xpath full-text 1.0,” 2008. [Online]. Available: http://www.w3.org/TR/xquery-full-text/

[15] R. Kaushik, R. Krishnamurthy, J. F. Naughton, and R. Ramakrishnan, “On the integration of structure indexes and inverted lists,” in *Proceedings of ACM SIGMOD 2004*, 2004.

[16] B. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Communications of the ACM*, vol. 13, pp. 422–426, July 1970.

[17] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, “Xrank: Ranked keyword search over xml documents,” in *Proceedings of ACM SIGMOD 2003*, 2003.

[18] Y. Xu and Y. Papakonstantinou, “Efficient keyword search for smallest lcas in xml databases,” in *Proceedings of ACM SIGMOD 2005*, 2005.

[19] V. Hristidis, Y. Papakonstantinou, and A. Balmin, “Keyword proximity search on xml graphs,” in *Proceedings of ICDE 2003*, 2003.

[20] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv, “Xsearch: A semantic search engine for xml,” in *Proceedings of VLDB 2003*, 2003.

[21] S. Al-Khalifa, C. Yu, and H. V. Jagadish, “Querying structured text in an xml database,” in *Proceedings of ACM SIGMOD 2003*, 2003.

[22] S. Amer-Yahia, L. V. Lakshmanan, and S. Pandit, “Flexpath: Flexible structure and full-text querying for xml,” in *Proceedings of ACM SIGMOD 2004*, 2004.

[23] N. Polyzotis and M. Garofalakis, “Structure and value synopses for xml data graphs,” in *Proceedings of VLDB 2002*, 2002.

[24] ——, “Xcluster synopses for structured xml content,” in *Proceedings of ICDE 2006*, 2006.

[25] W. Wang, H. Jiang, H. Lu, and J. X. Yu, “Bloom histogram: Path selectivity estimation for xml data with updates,” in *Proceedings of VLDB 2004*, 2004.

[26] S. Cho, N. Koudas, and D. Srivastava, “Meta-data indexing for xpath location steps,” in *Proceedings of ACM SIGMOD 2006*, 2006.

[27] L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt, “Locating data sources in large distributed systems,” in *Proceedings of VLDB 2003*, 2003.

[28] A. Bonifati, U. Matrangolo, A. Cuzzocrea, and M. Jain, “Xpath lookup queries in p2p networks,” in *Proceedings of WIDM 2004*, 2004.

[29] S. Abiteboul, I. Manolescu, and N. Preda, “Constructing and querying peer-to-peer warehouses of xml resources,” in *Proceedings of ICDE 2005*, 2005.

[30] S. Abiteboul, I. Manolescu, N. Polyzotis, N. Preda, and C. Sun, “Xml processing in dht networks,” in *Proceedings of ICDE 2008*, 2008.

[31] G. Koloniari and E. Pitoura, “Content-based routing of path queries in peer-to-peer systems,” in *Proceedings of EDBT 2004*, 2004.

[32] Q. Wang, A. Jha, and M. Ozsu, “An xml routing synopsis for unstructured p2p networks,” in *Proceedings of WAIMW 2006*, 2006.

[33] J. Bremer and M. Gertz, “On distributing xml repositories,” in *Proceedings of WebDB 2003*, 2003.

[34] E. Pitoura, S. Abiteboul, D. Pfoser, G. Samaras, and M. Vazirgiannis, "Db-globe:a service-oriented p2p system for global computing," *SIGMOD Record*, vol. 32, pp. 77–82, Sept. 2003.

[35] R. Kaushik, P. Bohannon, J. Naughton, and H. Korth, "Covering indexes for branching path queries," in *Proceedings of ACM SIGMOD 2002*, 2002.

[36] V. Papadimos, D. Maier, and K. Tufte, "Distributed query processing and catalogs for peer-to-peer systems," in *Proceedings of CIDR 2003*, 2003.

[37] R. Huebsch, B. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi, "Querying the internet with pier," in *Proceedings of VLDB 2003*, 2003.

[38] ——, "The architecture of pier: an internet-scale query processor," in *Proceedings of CIDR 2005*, 2005.

[39] B. T. Loo, J. M. Hellerstein, R. Huebsch, S. Shenker, and I. Stoica, "Enhancing p2p file-sharing with an internet-scale query processor," in *Proceedings of VLDB 2004*, 2004.

[40] I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyska, G. Miklau, and P. Mork, "The piazza peer data management project," *SIGMOD Record*, vol. 32, pp. 47–52, Sept. 2003.

[41] R. Kaushik, P. Bohannon, J. Naughton, and P. Shenoy, "Updates for structure indexes," in *Proceedings of VLDB 2002*, 2002.

[42] Oracle, "Berkeley db," 2008. [Online]. Available: http://www.oracle.com/database/berkeley-db/index.html/

[43] XMark, "Xmark," 2008. [Online]. Available: http://www.xml-benchmark.org/

[44] XBench, "Xbench," 2008. [Online]. Available: http://softbase.uwaterloo.ca/ ddbms/projects/xbench/

[45] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman, “On supporting containment queries in relational database management systems,” in *Proceedings of ACM SIGMOD 2001*, 2001.

[46] G. Graefe, “Query evaluation techniques for large databases,” *ACM Computing Surveys*, vol. 25, pp. 73–169, June 1993.

[47] Qizx, “Qizx/open,” 2008. [Online]. Available: http://www.xmlmind.com/qizx/

[48] R. B. Yates and B. R. Neto, *Modern Information Retrieval.* Boston, MA, USA: ACM Press., 1999.

[49] R. Fagin, A. Lotem, and M. Naor, “Optimal aggregation algorithms for middleware,” in *Proceedings of PODS 2001*, 2001.

[50] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman, “Structure and content scoring for xml,” in *Proceedings of VLDB 2005*, 2005.

[51] Wikipedia, “Distributed hash table,” 2008. [Online]. Available: http://en.wikipedia.org/wiki/Distributed_hash_table/

[52] Freepastry, “Freepastry2.0,” 2007. [Online]. Available: http://freepastry.rice.edu

[53] XMach, “Xmach,” 2008. [Online]. Available: http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html

[54] B. F. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, and M. Shadmon, “A fast index for semistructured data,” in *Proceedings of VLDB 2001*, 2001.

[55] Q. Li and B. Moon, “Indexing and querying xml data for regular path expressions,” in *Proceedings of VLDB 2001*, 2001.

[56] S. Khalifa, H. V. Jagadish, N. Koudas, J. M. patel, D. Srivastava, and Y. Wu, “Structural joins: A primitive for efficient xml query pattern matching,” in *Proceedings of ICDE 2002*, 2002.

[57] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes, "Exploiting local similarity for indexing paths in graph-structured data," in *Proceedings of ICDE 2002*, 2002.

[58] T. Grust, "Accelerating xpath location steps," in *Proceedings of ACM SIGMOD 2002*, 2002.

[59] C. W. Chung, J. K. Min, and K. Shim, "Apex: An adaptive path index for xml data," in *Proceedings of ACM SIGMOD 2002*, 2002.

[60] H. Jiang, H. Lu, W. Wang, and B. C. Ooi, "Xr-tree: Indexing xml data for efficient structural joins," in *Proceedings of ICDE 2003*, 2003.

[61] Q. Chen, A. Lim, and K. W. Ong, "D(k)-index: an adaptive structural summary for graph-structured data," in *Proceedings of ACM SIGMOD 2003*, 2003.

[62] H. Wang, S. Park, W. Fan, and P. S. Yu, "Vist: A dynamic index method for querying xml data by tree structures," in *Proceedings of ACM SIGMOD 2003*, 2003.

[63] P. Buneman, M. Grohe, and C. Koch, "Path queries on compressed xml," in *Proceedings of VLDB 2003*, 2003.

[64] P. Rao and B. Moon, "Prix: Indexing and querying xml using prufer sequences," in *Proceedings of ICDE 2004*, 2004.

[65] Y. Chen, S. B. Davidson, and Y. Zheng, "Blas: An efficient xpath processing system," in *Proceedings of ACM SIGMOD 2004*, 2004.

[66] H. Jiang, H. Lu, and W. Wang, "Efficient processing of xml twig queries with or-predicates," in *Proceedings of ACM SIGMOD 2004*, 2004.

[67] S. Abiteboul, O. Benjelloun, and B. Cautis, "Lazy query evaluation for active xml," in *Proceedings of ACM SIGMOD 2004*, 2004.

[68] A. Balmin, F. Ozcan, K. S. Beyer, R. J. Cochrane, and H. Pirahesh, "A framework for using materialized xpath views in xml query processing," in *Proceedings of VLDB 2004*, 2004.

[69] Y. Li, C. Yu, and H. V. Jagadish, "Schema-free xquery," in *Proceedings of VLDB 2004*, 2004.

[70] K. Tajima and Y. Fukui, "Answering xpath queries over networks by sending minimal views," in *Proceedings of VLDB 2004*, 2004.

[71] P. Buneman, B. Choi, W. Fan, R. Hutchison, R. Mann, and S. D. Viglas, "Vectorizing and querying large xml repositories," in *Proceedings of ICDE 2005*, 2005.

[72] K. Beyer, R. J. Cochrane, V. Josifovski, J. Kleewein, G. Lapis, G. Lohman, B. Lyle, F. Ozcan, H. Pirahesh, N. Seemann, T. Truong, B. V. Linden, B. Vickery, and C. Zhang, "System rx: one part relational, one part xml," in *Proceedings of ACM SIGMOD 2005*, 2005.

[73] K. Beyer, D. Chamberlin, L. S. Colby, F. Ozcan, H. Pirahesh, and Y. Xu, "Extending xquery for analytics," in *Proceedings of ACM SIGMOD 2005*, 2005.

[74] B. Mandhani and D. Suciu, "Query caching and view selection for xml databases," in *Proceedings of VLDB 2005*, 2005.

[75] S. Amer-Yahia, I. Fundulaki, and L. V. Lakshmanan, "Personalizing xml search in pimento," in *Proceedings of ICDE 2007*, 2007.

[76] X. Dong and A. Halevy, "Indexing dataspaces," in *Proceedings of ACM SIGMOD 2007*, 2007.

[77] G. Gou and R. Chirkova, "Efficiently querying large xml data repositories: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, pp. 1381–1403, Oct. 2007.

[78] S. Amer-Yahia, C. Botev, and S. Shanmugasundaram, “Texquery: A full-text search extension to xquery,” in *Proceedings of WWW 2004*, 2004.

[79] S. Amer-Yahia, E. Curtmola, and A. Deutsch, “Flexible and efficient xml search with complex full-text predicates,” in *Proceedings of ACM SIGMOD 2006*, 2006.

[80] D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer, “Searching xml documents via xml fragments,” in *Proceedings of ACM SIGIR 2003*, 2003.

[81] C. Clarke, “Controlling overlap in content-oriented xml retrieval,” in *Proceedings of ACM SIGIR 2005*, 2005.

[82] Z. Liu and Y. Chen, “Identifying meaningful return information for xml keyword search,” in *Proceedings of ACM SIGMOD 2007*, 2007.

[83] Z. Han, J. Le, and B. Shen, “Effectively scoring for xml ir queries,” in *Proceedings of DEXA 2006*, 2006.

[84] B. Sigurbjornsson, J. Kamps, and M. D. Rijke, “Processing content-oriented xpath queries,” in *Proceedings of CIKM 2004*, 2004.

[85] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava, “Keyword proximity search in xml trees,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, pp. 525–539, Apr. 2006.

[86] J. Kamps, M. Marx, M. D., Rijke, and B. Sigurbjornsson, “Structured queries in xml retrieval,” in *Proceedings of CIKM 2005*, 2005.

[87] W. Luk, H. V. Leong, S. Dillon, T. S. Chan, W. Croft, and J. Allan, “A survey in indexing and searching xml documents,” *Journal of the American Society for Information Science and Technology*, vol. 53, pp. 415–437, May 2002.

[88] N. Bruno, L. Gravano, and A. Marian, “Evaluating top-k queries over web-accessible databases,” in *Proceedings of ICDE 2002*, 2002.

[89] A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava, “Adaptive processing of top-k queries in xml,” in *Proceedings of ICDE 2005*, 2005.

[90] N. Mamoulis, K. Cheng, M. Yiu, and D. Cheung, "Efficient aggregation of ranked inputs," in *Proceedings of ICDE 2006*, 2006.

[91] S. Michel, P. Triantafillou, and G. Weikum, "Klee: A framework for distributed top-k query algorithms," in *Proceedings of VLDB 2005*, 2005.

[92] P. Cao and Z. Wang, "Efficient top-k query calculation in distributed networks," in *Proceedings of PODC 2004*, 2004.

[93] M. Theobald, G. Weikum, and R. Schenkel, "Top-k query evaluation with probabilistic guarantees," in *Proceedings of VLDB 2004*, 2004.

[94] M. Theobald, R. Schenkel, and G. Weikum, "An efficient and versatile query engine for topx search," in *Proceedings of VLDB 2005*, 2005.

[95] H. Gast, D. Majumdar, R. Schenkel, M. Theobald, and G. Weikum, "Io-top-k index-access optimized top-k query processing," in *Proceedings of VLDB 2006*, 2006.

[96] G. Das, D. Gunopulos, and N. Koudas, "Answering top-k queries using views," in *Proceedings of VLDB 2006*, 2006.

[97] M. Theobald, R. Schenkel, and G. Weikum, "The topx db&ir engine," in *Proceedings of ACM SIGMOD 2007*, 2007.

[98] N. Polyzotis, M. Garofalakis, and Y. Ioannidis, "Approximate xml query answers," in *Proceedings of ACM SIGMOD 2004*, 2004.

[99] J. Spiege, E. Pontikakis, S. Budalakoti, and N. Polyzotis, "Aqax: A system for approximate xml query answers," in *Proceedings of VLDB 2006*, 2006.

[100] T. Suel, C. Mathur, J. W. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram, "Odissea: A peer-to-peer architecture for scalable web search and information retrieval," in *Proceedings of WebDB 2003*, 2003.

[101] C. Tang, Z. Xu, and S. Dwarkadas, "Peer-to-peer information retrieval using self-organizing semantic overlay networks," in *Proceedings of ACM SIGCOMM 2003*, 2003.

[102] D. Zeinalipour-Yazti, V. Kalogeraki, and D. Gunopulos, "Exploiting locality for scalable information retrieval in peer-to-peer networks," *Information Systems*, vol. 30, pp. 277–298, Apr. 2005.

[103] B. C. Ooi, Y. Shu, and K. Tan, "Relational data sharing in peer-based data management systems," *SIGMOD Record*, vol. 32, pp. 59–64, Sept. 2003.

[104] W. S. Ng, B. C. Ooi, K. Tan, and A. Zhou, "Peerdb:a p2p-based system for distributed data sharing," in *Proceedings of ICDE 2003*, 2003.

[105] W. S. Ng, B. C. Ooi, and K. Tan, "A self-configurable peer-to-peer system," in *Proceedings of ICDE 2002*, 2002.

[106] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. J. Miller, and J. Mylopoulos, "The hyperion project:from data integration to data coordination," *SIGMOD Record*, vol. 32, pp. 53–58, Sept. 2003.

[107] M. A. A. Kementsietsidis and R. J. Miller, "Mapping data in peer-to-peer systems:semantics and algorithmic issues," in *Proceedings of ACM SIGMOD 2003*, 2003.

[108] W. Nejdl, W. Siberski, and M. Sintek, "Design issues and challenges for rdf and schema-based peer-to-peer systems," *SIGMOD Record*, vol. 32, pp. 41–46, Sept. 2003.

[109] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch, "Edutella:a p2p networking infrastructure based on rdf," in *Proceedings of ACM WWW 2002*, 2002.

[110] W. Nejdl, M. Wolpers, W. S. C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Laser, "Super-peer-based routing and clustering strategies for rdf-based peer-to-peer networks," in *Proceedings of ACM WWW 2003*, 2003.

[111] A. Halevy, Z. Ives, P. Mork, and I. Tatarinov, "Piazza:data management infrastructure for semantic web applications," in *Proceedings of ACM WWW 2003*, 2003.

[112] J. Madhavan and A. Halevy, "Composing mappings among data sources," in *Proceedings of VLDB 2003*, 2003.

[113] I. Tatarinov and A. Halevy, "Efficient query reformulation in peer data management systems," in *Proceedings of ACM SIGMOD 2004*, 2004.

[114] V. Papadimos and D. Maier, "Mutant query plans," *Information and Software Technology*, vol. 44, pp. 197–206, Apr. 2002.

[115] L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt, "Processing queries in a large peer-to-peer system," in *Proceedings of CAiSE 2003*, 2003.

[116] W. Balke, W. Nejdl, W. Siberski, and U. Thaden, "Progressive distributed top-k retrieval in peer-to-peer networks," in *Proceedings of ICDE 2005*, 2005.

[117] A. Crespo and H. G. Molina, "Routing indices for peer-to-peer systems," in *Proceedings of ICDCS 2002*, 2002.

[118] B. Yang and H. G. Molina, "Designing a super-peer network," in *Proceedings of ICDE 2003*, 2003.

[119] B. Yang and H. G. Molina, "Improving search in peer-to-peer networks," in *Proceedings of ICDCS 2002*, 2002.

[120] O. D. Sahin, A. Gupta, D. Agrawal, and A. E. Abbadi, "A peer-to-peer framework for caching range queries," in *Proceedings of ICDE 2004*, 2003.

[121] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica, “Complex queries in dht-based peer-to-peer networks,” in *Proceedings of IPTPS 2002*, 2002.

[122] K. Aberer, P. C. Mauroux, A. D. Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt, “P-grid: A self-organizing structured p2p system,” *SIGMOD Record*, vol. 32, pp. 29–33, Sept. 2003.

[123] D. Tsoumakos and N. Roussopoulos, “A comparison of peer-to-peer search methods,” in *Proceedings of WebDB 2003*, 2003.

[124] N. Daswani, H. G. Molina, and B. Yang, “Open problems in data-sharing peer-to-peer systems,” in *Proceedings of ICDT 2003*, 2003.

[125] K. Aberer and M. Hauswirth, “Peer-to-peer information systems: Concpets and models, state-of-the-art, and future systems,” in *Proceedings of ICDE 2002*, 2002.

[126] I. Podnar, M. Rajman, T. Luu, F. Klemm, and K. Aberer, “Scalable peer-to-peer web retrieval with highly discriminative keys,” in *Proceedings of ICDE 2007*, 2007.

[127] R. Geambasu, M. Balazinska, S. D. Gribble, and H. M. Levy, “Homeviews: Peer-to-peer middleware for personal data sharing applications,” in *Proceedings of SIGMOD 2007*, 2007.

[128] S. Michel1, M. Bender, P. Triantafillou, and G. Weikum, “Iqn routing: Integrating quality and novelty in p2p querying and ranking,” in *Proceedings of ACM EDBT 2006*, 2006.

[129] M. Bender, S. Michel, P. Triantafillou, and G. Weikum, “Global document frequency estimation in peer-to-peer web search,” in *Proceedings of WebDB 2006*, 2006.

[130] Y. Joung and L. Yang, “Kiss: A simple prefix search scheme in p2p networks,” in *Proceedings of WebDB 2006*, 2006.

[131] Y. He, Y. Shu, S. Wang, and X. Du, “Efficient top-k query processing in p2p network,” in *Proceedings of DEXA 2004*, 2004.

[132] P. Linga, A. Crainiceanu, J. Gehrke, and J. Shanmugasudaram, “Guaranteeing correctness and availability in p2p range indices,” in *Proceedings of ACM SIGMOD 2005*, 2005.

[133] P. Ganesan, M. Bawa, and H. G. Molina, “Online balancing of range-partitioned data with applications to peer-to-peer systems,” in *Proceedings of VLDB 2004*, 2004.

[134] H. Jagadish, B. C. Ooi, and Q. H. Vu, “Baton: A balanced tree structure for peer-to-peer networks,” in *Proceedings of VLDB 2005*, 2005.

[135] A. Bonifati, E. Q. Chang, T. Ho, L. V. S. Lakshmanan, and R. Pottinger, “Heptox: Marrying xml and heterogeneity in your p2p databases,” in *Proceedings of VLDB 2005*, 2005.

[136] Galax, “Galax,” 2008. [Online]. Available: http://www.galaxquery.org/

[137] Galatex, “Galatex,” 2008. [Online]. Available: http://www.galaxquery.com/galatex/

[138] L. Fegaras, W. He, G. Das, and D. Levine, “Xml query routing in structured p2p networks,” in *Proceedings of DBISP2P 2006*, 2006.

[139] W. He, L. Fegaras, and D. Levine, “Indexing and searching xml documents based on content and structure synopses,” in *Proceedings of BNCOD 2007*, 2007.

[140] W. He, L. Fegaras, and D. Levine, “Locating and ranking xml documents based on content and structure synopses,” in *Proceedings of DEXA 2007*, 2007.

[141] W. He and L. Fegaras, “Approximate xml query answers in dht-based p2p networks,” in *Proceedings of DASFAA 2008*, 2008.

[142] W. He and L. Fegaras, “Answering xpath queries with search predicates in structured p2p networks,” *International Journal of Computer Systems Science and Engineering*, vol. 23, pp. 35–55, Mar. 2008.

BIOGRAPHICAL STATEMENT

Weimin He was born in Kunming, China. He received his B.S. and M.S. degrees in computer science from Yunnan University, China. He received his Ph.D. degree in computer science from The University of Texas at Arlington in 2008.