Research

Indexing for XML-based Articles

Canan F. Pembe and Taflan 0. Gündem
Computer Engineering Dept.,
Boaziçi University, 80815 Bebek, 0stanbul, Turkey

Abstract: Traditional information retrieval systems have problems in the relevancy of the search results. They usually return a huge number of results to query requests, mainly because they do not have the ability to utilize the semantic information XML can provide. The solution is XML-based information retrieval systems. Furthermore, an application area of this subject is XML-based article retrieval. In recent years, there has been a great tendency, both in the academia and the industry, to publish articles electronically using XML. In this paper, we present an indexing schema for the article retrieval system that we propose - in which all the articles are formed as XML documents. The indexing schema is geared for efficient processing of queries, considering the specific properties in the retrieval of XML-based articles. Furthermore, we compare the performance of our proposed indexing schema with an alternative approach.

Keywords: Article retrieval; Indexing; XML; Relational Databases

Received 28 March; Revised and Accepted 4 May 2003

1. Introduction

In today's world, the greater part of information comes in the form of documents, such as books, journals, newspapers and technical manuals. Owing to computers and automation, the most of these documents are prepared, stored, displayed and searched electronically today. In this large amount of data, as a result of large document collections, information retrieval in a fast and precise manner has become vital.

Most of current search engines, such as *Yahoo! Inc.*, use agent software –called "spider"- in order to visit web pages on the whole Internet periodically and store their content as indexes in large databases. Some of the problems faced by the users of current search engines are that they cannot differentitate between the contexts a keyword can appear; e.g., the returned results of a query containing "ATM" will probably include a huge amount of garbage because the engine cannot distinguish the "ATM"s in unrelated contexts (such as "Automatic Teller Machine" in banking, "Asynchronous Transfer Mode" in computer networking, or "Anti-Tank Missile" in wars), as stated in *Tanabe, H. and Wang, C.-C.*

Because of its ability to represent semantics of data in a structured and machine-readable form, XML is becoming a standard for the publication and interchange of documents. Electronic publishing is an area in which XML metadata can provide great opportunities. Some organizations have already started to demand and distribute their publications in XML (for example *XML Europe 2001 Conference*, 2001). XML is expected to become widespread in both scientific and technical publications because of its platform and software independent features. There are already standardization efforts for XML-based publications; for example, *DocBook Org.,* (2002) is a DTD (Document Type Definition) well-suited to books and articles.

Indexing expedites the retrieval of information from documents. There has been some work done on indexes for information retrieval from XML documents, for example those by *Zhang, C. et al.,* (2001) and *Seo, C. and Kim, H.-J.,* (2002). In this paper we propose an indexing scheme not for XML documents in general but just for a specific type of XML documents, namely XML-based technical and scientific articles. The indexing schema that we propose is novel and considers the special requirements of XML-based articles distributed over the Internet. We compare the performance of our indexing schema with an alternative indexing schema.

The rest of the paper is organized as follows: Section 2 gives the related work especially in XML-based information retrieval and indexing. Section 3 describes the structure of XML-based articles and the related work. In Section 4, the proposed XML-based article retrieval system and the indexing schema are explained. Finally, our indexing approach is evaluated in Section 5.

2. Related Work

In our design, we are influenced by some of the concepts and ideas presented in the papers which are summarized in this section. However, our work differs considerably from each one of these works.

A recent work on indexing for XML information retrieval systems using relational databases is by *Seo, C. and Kim, H.-J.*, (2002). Basically, *Seo, C. and Kim, H.-J.*, (2002) overviews the traditional inverted index widely used

in information retrieval field. *Seo, C. and Kim, H.-J.*, (2002) show that a recent improvement on the inverted indexing method by *Zhang, C. et. al.,* (2001) that uses a T-index for indexing text words and an E-index for indexing elements (named as 2-INDEX approach) is not suitable for indexing XML documents. Instead *Seo, C. and Kim, H.-J.,* (2002) propose an approach which uses four different types of indexes. In our indexing approach we use basically just two indexes. (The third index we use is for the url addresses of the documents.) Also our index structures have different formats than those proposed by *Seo, C. and Kim, H.-J.,* (2002). The indexing sheme proposed by *Seo, C. and Kim, H.-J.,* (2002) is for documents in general whereas the one that we propose is designed for XML-based article retrieval.

Containment queries definitely form a crucial part of queries for XML information retrieval systems. They are defined as a class of queries based on containment and proximity relationships among elements, attributes, and their contents. In the paper *by Seo, C. and Kim, H.-J.*, (2002), the containment queries are classified into four types and are explained using query expressions similar to XPath. Since we are also going to use containment queries in our explanation, we give a summary of the containment query types that are presented by *Seo, C. and Kim, H.-J.,* (2002). In the explanation of the containment query types we use the XML document in Figure 1 and and its tree representation in Figure 2. As stated, containment queries are classified into four: *Indirect Containment Queries*, *Direct Containment Queries*, *Tight Containment Queries*, and *Proximity Containment Queries*.

Indirect Containment Queries consist of predecessor-descendant relationships among elements, attributes, and their contents. For example, */publications//author//'Mary Johns'* is an indirect containment query, where the first "/" indicates the root element, and "//" represents a predecessor-descendant relationship. The query means retrieving all XML documents in which "publications" root element has "author" descendant elements and in turn "author" elements have "Mary Johns" descendant content.

Direct Containment Queries consist of parent-child relationships among elements, attributes, and their contents. For example, in the query *//book/author/name*, the first "//" means that "book" does not have to be a root element, and "/" represents a parent-child relationship. The query means retrieving all XML documents in which "book" elements have "author" child elements and in turn "author" elements have "name" child elements.

Tight Containment Queries consists of elements, attributes and their contents. For example, the query *//name='John Johns'* means retrieving all XML documents in which "name" elements contain only "John Johns" in their contents.

Proximity Containment Queries are based on the proximity between two words in contents. In XML-based information retrieval systems, commonly, the hybrid forms of these four containment queries are used.

```
<publications>
    <book>
      <isbn> 123-567 </isbn>
      <author>
        <name> "Mary Johns" </name>
        <name> "John Johns" </name>
      </author>
      <title> "Web Master" </title>
      <abstract> "Book for web masters" </abstract>
    </book>
</publications>
```

Figure 1. An XML Document

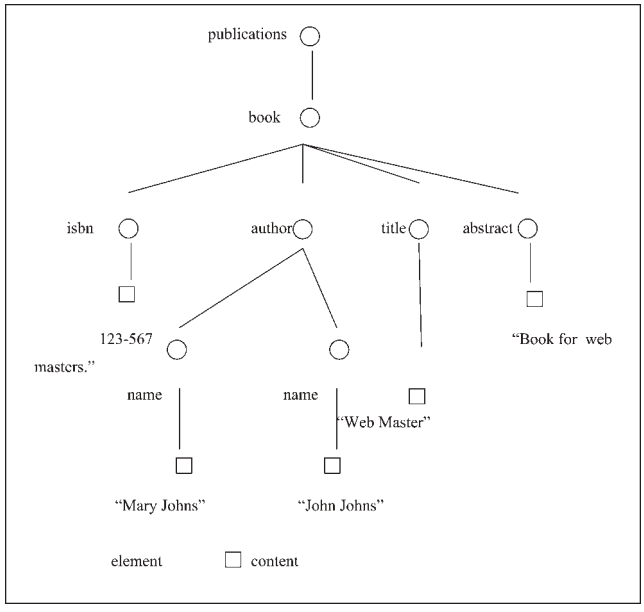


Figure 2. Tree representation of the XML document in Figure 1.

According to *Egnor, D. and Lord, R.*, (2000), there are three ways semantically structured queries can be applied to XML documents. *Proactively structured queries* are the queries with full structure submitted either as a

query language formulation such as a query in XPath or through the fields in a search form. *Reactively structured queries* start as structureless queries and added structure by the user during a refinement process. *Automatically structured queries* also start as structureless queries and are added structure during an automatic process, e.g. by applying natural language processing to the query.

In the proposal of *Egnor, D. and Lord, R.,* (2000), structure is added to user's query in order to improve search precision. The user is provided with a user interface in order not to oblige them to formulate complex database queries, i.e. reactively structured queries are used. The search begins with an unstructured keyword query, such as "Ford", whose result is a list of matching schemas. As the user selects the schema he is interested in, he is presented with a search form corresponding to the selected schema. There is exactly one field for each unique XML path in the selected type of documents, and the form may be presented visually in order to express specific metadata effectively. Our idea is that such a visual form can also be used in an XML-based article retrieval system. Thus, we designed a user interface as in Figure 4.

Egnor, D. and Lord, R., (2000) gives the address of a content as a sequence of numbers representing the outer elements containing this content rather than the actual offset from the beginning of the document. We borrowed the structured address idea (which is explained in section 4.1.) in our design from *Egnor, D. and Lord, R.,* (2000).

In the paper by *Barber, D.,* (2001), the subject of *electronic journals* are overviewed with the experience of OhioLINK - a consortium of university and college libraries which has built one of the largest electronic journal archives. The users of electronic journal archives may retrieve articles through two types of search: *searching of full-text* and *searching of article metadata*. With full-text searching, users can retrieve articles by specifying certain words or phrases to be present in text of the article. Searching with article metadata enables retrieval according to the bibliographic data of an article (such as its title, authors and publication date). *Barber, D.,* (2001) states that the ability to search the full-text of articles may be questionable with large archives. The reason is that searching such a big amount of journals of several subject disciplines with full-text searching would return too many results most of which are useless. As a result, some archive designs are using only metadata search. Therefore, in our design, we preferred searching with metadata.

3. XML-based Articles

XML documents can be classified into three different categories as explained by *Klettke, M.,* (2001): Data-centric, document-centric and mixed approach XML documents. Examples for each of these approaches are shown in Figure 3.

Data-centric

```
<order>
   <customer>John</customer>
   <position>
      <isbn>1-234-56788-0</isbn>
      <number>4</number>
      <price currency=„Euro“> 37.00</price>
   </position>
</order>
```

Document-centric

```
<content>
   <section>
      <para>The solution is XML-based information retrieval systems. Furthermore, an application area of this subject is XML-based <emph>article retrieval</emph>. In recent years, there has been a great tendency, both in the academia and the industry, to publish articles electronically using <emph>XML</emph>. In this paper, we present an indexing schema …..
      </para>
   <section>
   ……..
</content>
```

Mixed approach

```
<book>
   <author>Neil Nillson</author>
   <title>XML storage</title>
   <isbn>1-234-56789-0</isbn>
  <content>
      <para><emph>XML</emph> builds on the principles of two existing languages,
```

Figure 3. Examples of Data-centric, Document-centric and Mixed approach XML documents

The first category includes data-centric XML documents characterized by highly regular structure. That is, the structure of data-centric XML documents is generally common to all pages describing a particular kind of information. For example, consider an XML document consisting of product orders. Each product order in the

document has elements for customer name, order position, and price in common. Moreover, these elements mostly contain fine-grained data.

The second category includes document-centric XML documents characterized by less regular or irregular structure, and larger grained data. For example, consider a webpage which consists of several sections each of which contains several paragraphs. Anywhere within a paragraph, there can be any type of other elements, e.g. <emph> to emphasize certain words. This is the source of irregularity because this means that unlike the product order example, a webpage is not restricted to a certain structure. And also, the data is larger grained. That is, there can be large paragraphs between the tags, whereas in data-centric approach the data between the tags is concise.

The third category includes the mixed approach documents which contain both data-centric and document-centric portions such as a book. The regular parts of a book may be its title, author, isbn etc. which are common to all books. The irregular parts of it may be its content whose structure may differ greatly as shown by *Klettke, M.,* (2001) and *Bourret, R., (2002)*.

XML-based articles usually conform to the mixed approach category. Considering the structure of XML-based articles on the Internet, we derived a general structure with the basic components an article must have. An example XML-based article is given in Figure 4. The article is enclosed in “article” tags. It has three main parts: article header, article body and bibliography. The data-centric parts are the header (where information such as the title, authors and keywords of the article are stored) and the bibliography (where the references made in the article are stored). The body part is mostly document-centric and consists of the sections of the article, which in turn consist of paragraphs.

```
<article>
    <artheader>
        <title>An example article</title>
        <authorgroup>
            <author>
                <firstname>first name</firstname>
                <surname>surname</surname>
                <affiliation>company name</affiliation>
                <address>
                    <email>foo@example.com</email>
                    <city>Vancouver</city>
                    <country>Canada</country>
                </address>
            </author>
        <authorgroup>
        <abstract>
            <para>Here is the abstract of the article.</para>
        </abstract>
        <keywords>
            <keyword>Schema</keyword>
            <keyword>XML</keyword>
        </keywords>
        <Journal>ACM Transactions on Database Systems</Journal>
        <Year>1992</Year>
        <Month>September</Month>
    </artheader>
    <section>
        <title>My first section</title>
        <para>This is the first section in my article.</para>
        <subsection>
            <title>My first sub-section</title>
            <para>This is the first sub-section in my article.</para>
        </subsection>
    </section>
        .
        .
        .
    <bibliography>
        <biblioentry>
            <title>Title of the entry</title>
            <authorgroup>
                <author>
                    <surname>Surname of author</surname>
                </author>
            </authorgroup>
            <pubdate>1990</pubdate>
        </biblioentry>
        ..........
    </bibliography>
</article>
```

Figure 4. An example XML-based article

4. Proposed System

In the following, we present the summary of the XML-based article retrieval system that we propose. The proposed system contains an *indexing agent* and a *search agent*. The indexing agent is responsible for parsing the XML documents representing articles and building indexes over them. The search agent is responsible for processing queries (i.e. retrieving articles satisfying the conditions specified in a query). The user interface, shown in Figure 5, is designed for specifying queries to the system.

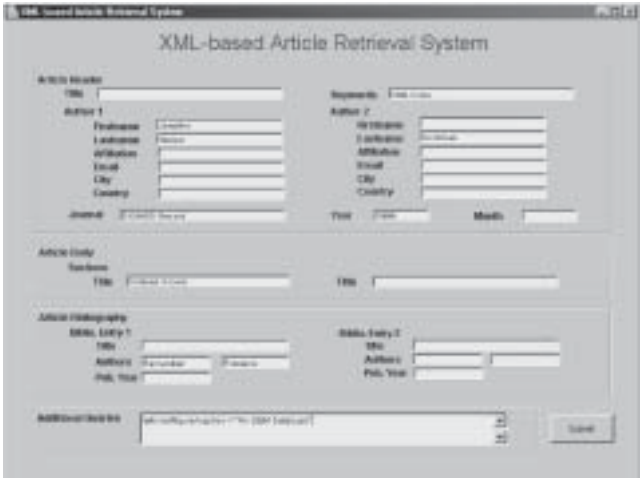


Figure 5. User interface of the system

The general workflow of the system is as follows. The indexing agent is run whenever a new document is to be added to the retrieval system. It takes the URL of the XML document as input and accesses the document. The task of the indexing agent is to parse the document and navigate through it in order to build the necessary indexes. The indexes are stored in a relational database. As with most information retrieval systems for the Internet, only indexes for the original document (not the document itself) are stored and the retrieval is done based on these indexes. Using relational database management systems makes the whole system easy to implement, economical (i.e. Smaller amount of code has to be written.) and dependable (i.e. Relational databases have been around for several decades now.).

The search agent is run upon a request from a user who specifies queries on the articles that are indexed. The user fills in the form in the user interface (Figure 5) to specify his/her query. Each field in the form corresponds to one component of the query. For example, specifying “Indexing XML” in the title field within the article header part corresponds to the XPath expression /article/artheader/title = “Indexing XML”. The user is provided with a final field which is for additional subqueries he/she cannot specify with the standard fields. Additional queries are to be entered as XPath expressions. The search agent finds the URLs of the documents; i.e. articles, that satisfy the conditions specified in the query. The URLs are computed by accessing the indexes available in the system. In our system no pre-defined DTD is assumed for the articles. In the following section we explain the indexes that we use in our system.

4.1. Proposed Indexing Schema

Since the presence of indexes are of utmost importance in expediting the processing of queries, we focus on the indexes used in our system in this section. In general, the type of indexes selected depends on the type of the document (which we have already explained) and also the type of the queries asked. The queries to retrieve the relevant articles are based on the containment relationships explained in Section 2. A query consists of conjunctions of the containment query types. In the following, some sample query components which may be posed to an XML-based article retrieval system are given.

- / article / artheader / title = ”Indexing XML”
- / article / artheader / authorlist / author / firstname = ”Jennifer” and surname = ”Widom”
- / article // figure / caption = “Benchmark Param eters”
- // abstract // acronym = ”XSL”

The following is a valid query in the system. It is the conjunction of the above query components as stated:

- (/ article / artheader / title = ”Indexing XML”) **AND**

  (/ article / artheader / authorlist / author / firstname = ”Jennifer”and surname = ”Widom”) **AND** (/ article // figure / caption = “Benchmark Param eters”)

  AND (// abstract // acronym = ”XSL”)

In order to suggest an efficient indexing structure specifically for XML-based articles, we considered the possible queries which can be entered by the users. We classified the possible queries into two categories. The first category includes containment queries on data-centric portions of an article (for example, /article/artheader/title=”Indexing XML”). These queries are on “standard” and regular portions of articles. The second category includes queries on document-centric portions (for example, //abstract//acronym=”XSL”). These queries are on textual and mostly irregular parts.

As a design decision for such an article retrieval system, we preferred not to use full-text indexing for document-centric portions of articles (e.g., for the text portions under <abstract> and <para> tags). This means that not every individual word within paragraphs are indexed. Only a portion of words within article paragraphs

(which are in special tags) are indexed. For example, not all the words within the paragraphs in the abstract part of an article are indexed; instead, content within special tags is. This design decision is based on performance and precision reasons stated by *Barber, D.,* (2001).

After considering different indexing approaches in XML-based information retrieval, and the needs and characteristics of an XML-based article retrieval system, the following three index types are suggested. One for storing the document IDs (document index); one for storing schema information (path index); and one for storing instance information (value index). These indexes of the XML article retrieval system are stored as three different relations, namely DocumentIndex, PathIndex and ValueIndex, in a relational database.

The document index stores the URLs of the XML documents by assigning a unique document identifier to each document, as shown in Figure 6. Each time a new document is added to the system, this table is updated.

The path index, shown in Figure 7, stores all different path types which exist between the root and any leaf node (value node) in an XML document; that is, absolute paths consisting of direct containment relationships, e.g., /article/artheader/title. When a new article is added to the system, the possible path types are obtained and the new ones are added to the path index. Each path type is assigned a unique ID.

URL	DocID
http://www.papers.com/xml/article1.xml	1
http://www.papers.com/xml/article2.xml	2
http://www.papers.com/xml/article3.xml	3
http://www.papers.com/xml/article4.xml	4
http://www.papers.com/xml/article5.xml	5

Figure 6. A sample document index

Path	PathID
/article/artheader/title	1
/article/artheader/authorgroup/author/firstname	2
/article/artheader/authorgroup/author/surname	3
/article/artheader/authorgroup/author/affiliation	4
/article/artheader/authorgroup/author/address/city	5
/article/artheader/authorgroup/author/address/country	6
/article/artheader/keywords/keyword	7
/article/artheader/year	8
/article/artheader/month	9
/article/section/title	10
/article/section/para/emph	11
/article/section/para/acronym	12
/article/section/image/caption	13
/article/bibliography/biblientry/title	14
/article/bibliography/biblientry/authorgroup/author/surname	15
/article/bibliography/biblientry/pubdate	16

Figure 7. A sample path index

PathID	DocID	FullPath	Value
1	1	1 1 1 1	Searching Text-rich XML Documents with Relevance Ranking
2	1	1 1 1 2 1 1	Yoshihiko
3	1	1 1 1 2 1 2	Hayashi
4	1	1 1 1 2 1 3	NTT Cyberspace Laboratories
5	1	1 1 1 2 1 4 1	Yokosuka
6	1	1 1 1 2 1 4 2	Japan
2	1	1 1 1 2 2 1	Jun-ji
3	1	1 1 1 2 2 2	Tonita
4	1	1 1 1 2 2 3	NTT Cyberspace Laboratories
5	1	1 1 1 2 2 4 1	Yokosuka
6	1	1 1 1 2 2 4 2	Japan
2	1	1 1 1 2 3 1	Genichiro
3	1	1 1 1 2 3 2	Kiku

Figure 8. A sample value index

The value index stores the instances of the path types stored in the path index, as shown in Figure 8. For each instance, the full path is stored as a sequence of numbers representing outer elements (from outermost to innermost in a hierarchical order) that contain the content specified in the value field. Each number identifies the position of an element at a certain level. In Figure 9, a portion of a document is given to illustrate this addressing scheme. Using the document in Figure 9, an instance of path type */ article / artheader / authorgroup / author / address / city* can be uniquely identified by the document ID, 1, and the sequence of numbers "1 1 2 1 4 1". Full-path addressing together with document ID uniquely identifies each path instance. This is because there is only one element at a certain position at each level in a document.

```
1<article>
    1<artheader>
        1<title>Searching Text-rich XML Documents with Relevance Ranking</title>
        2<authorgroup>
            1<author>
                1<firstname>Yoshihiko</firstname>
                2<surname>Hayashi</surname>
                3<affiliation>NTT Cyberspace Laboratories</affiliation>
                    4<address>
                    1<city>Yokosuka</city>
                    2<country>Japan</country>
```

Figure 9. Example full path addressing

Query evaluation for a query such as */article/artheader/ title = "Database"*, can be summarized as follows.

a) First extract the PathID of Path "/article/ artheader/title" from PathIndex.

b) Then from ValueIndex, extract the DocID of tuples with Value equal to "Database" and PathID equal to that obtained at step a).

c) At last from DocumentIndex, extract the URLs of documents whose DocIDs are obtained at step b).

The SQL statement to evaluate this query (i.e. */article/artheader/title = "Database"*) is as follows:

```
SELECT URL
FROM DocumentIndex, PathIndex, ValueIndex
WHERE PathIndex . Path = "/article/artheader/title" AND
      ValueIndex . value = "Database" AND
      PathIndex . PathID = ValueIndex . PathID AND
      ValueIndex . DocID = DocumentIndex . DocID
```

If a query includes "//" (indirect containment), every occurrence of "//" is replaced with "%/" in the string representing the path and LIKE is used instead of = in the first conjunct of the WHERE clause of the SQL statement. For example, the SQL statement used in evaluating the query *// title / "Database"* is the same as that for */ article / artheader / title / "Database"*, except that instead of PathIndex.Path = "/ article / artheader / title" in the first conjunct, we have PathIndex . Path LIKE "% / *title / "Database".*

For any query we have the same SQL statement except that the first two conjuncts are different. Thus we have only two join operations among DocumentIndex, PathIndex and ValueIndex regardless of the length of the path in the query (i.e. the number of containment relationships). Since join operations are time-consuming operations, the number of join operations executed is important in determining the performance of a system.

5. Performance Evaluation

In this section, the performance of the index structures of the proposed XML-based article retrieval system is evaluated. With this purpose, the proposed indexing structure is compared with an alternative approach. We are not aware of any indexing method in the literature specifically used for XML-based articles. Thus, we came up with an alternative approach after considering most likely candidates among indexing methods for general XML documents and modifying it appropriately.

In coming up with an alternative approach, we are influenced by the 2-INDEX approach (known as inverted indexes), which is widely used in general-purpose XML-based information retrieval systems. However, we did not use the original 2-INDEX approach for comparison because, unlike our system, it is a full-text approach. We greatly modified it in order to make it comparable to our method as will be explained in the following paragraphs. As a result, the alternative method used is no more a full-text indexing method. We chose this approach as an alternative because it is the most likely approach for comparison.

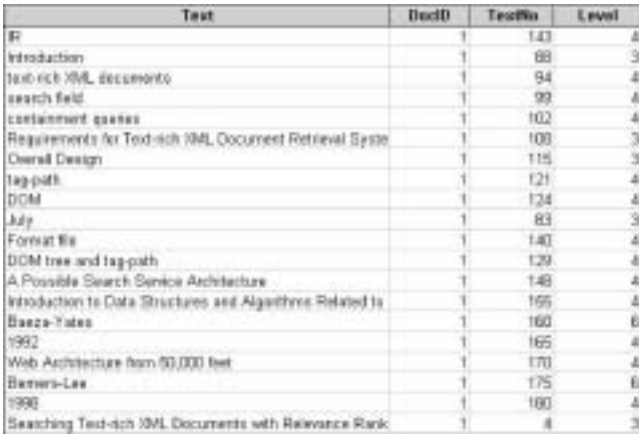
In the following discussion, the system works on local files rather than remote files for the remote file access would take the same amount of time for both approaches anyway.

Three types of indexes are used in the alternative approach: *document index* is used for storing the document IDs, *E-Index* is used for "Elements" and *T-Index* is used for "Texts". These three indexes are stored as three different relations in a relational database. The document index is identical to our document index shown in Figure 6.

An E-Index is used for indexing elements. In an E-Index, each occurrence of an element is indexed according to its document number, its beginning and ending positions as word counts, and its nesting level within the document. The schema of an E-Index is (Element, DocID, Begin, End, Level), as shown in Figure 10. T-Index is used for indexing words in a text. Each text content is indexed with its document number, its position, and its nesting level within the document. The schema of a T-Index is (Text, DocID, TextNo, Level), as shown in Figure 11. Similar to our proposed indexing approach, in the alternative approach, not all the words within document-centric parts of articles are indexed; instead, content within special tags is indexed. Furthermore, similar to our proposed approach, text content within tags is not indexed word by word as in the original 2-INDEX approach; instead each text content is indexed "as a whole". For example, the T-Index contains one index for "containment queries" instead of separate indexes for the words "containment" and "queries".

Element	DocID	Begin	End	Level
emph	1	93	95	3
keywords	1	65	78	2
keyword	1	66	68	3
keyword	1	69	71	3
keyword	1	72	74	3
keyword	1	75	77	3
year	1	79	81	2
month	1	82	84	2
section	1	86	105	1
title	1	87	89	2
image	1	127	131	2
para	1	92	96	2
address	1	55	62	4
para	1	97	104	2
emph	1	98	100	3
emph	1	101	103	3
section	1	106	106	1
title	1	107	109	2
para	1	110	111	2
section	1	112	113	1

Figure 10. A sample E-Index of the alternative approach



Text	DocID	TextNo	Level
IR	1	143	4
Introduction	1	88	3
text-rich XML documents	1	94	4
search field	1	99	4
containment queries	1	102	4
Requirements for Text-rich XML Document Retrieval Syste	1	108	3
Overall Design	1	115	3
tag-path	1	121	4
DOM	1	124	4
July	1	83	3
Format file	1	140	4
DOM tree and tag-path	1	129	4
A Possible Search Service Architecture	1	148	4
Introduction to Data Structures and Algorithms Related to	1	155	4
Baeza-Yates	1	160	6
1992	1	165	4
Web Architecture from 50,000 feet	1	170	4
Berners-Lee	1	175	6
1998	1	180	4
Searching Text-rich XML Documents with Relevance Rank	1	8	3

Figure 11. A sample T-Index of the alternative approach

The experiments are performed on a 500 MHZ Pentium-III machine running under Microsoft Windows XP with a main memory size of 128 MB. Microsoft Access 2002 is used as the relational database management system. For the comparisons of our proposed indexing approach and the alternative approach, the following queries are used:

- Query-1: //emph="containment queries"
- Query-2: //para//acronym="IR"
- Query-3: /article/artheader/title/"Database"
- Query-4: /article/artheader//author//country="Italy"
- Query-5: /article/artheader/authorgroup/author/surname="Rizzolo"
- Query-6: /article/artheader/authorgroup/author/firstname="Alberto" and surname="Mendelzon"

As seen, in our experimental queries, only query components are used, instead of conjunctions of query components (See Section 4.1, for an explanation of query components and their conjunctions in our system). The reason is that getting a performance result for the components, will be enough to provide the necessary information for the performance of the queries resulting from their conjunctions.

The queries are executed on three different-sized XML document collections containing 1200 (Experiment 1), 4800 (Experiment 2), and 9600 (Experiment 3) articles respectively (One may refer to *Pembe, F. C*. 2002 for the experiment data). In Tables 1a, 1b, and 1c, the two approaches are compared on index sizes. As seen, the proposed indexing approach introduces less storage overhead than the alternative approach.

Table 1a. Index sizes in Experiment 1

	Size (MB)	Number of Articles
Our Approach	3.180	1200
Alternative Approach	5.236	1200

Table 1b. Index sizes in Experiment 2

	Size (MB)	Number of Articles
Our Approach	10.640	4800
Alternative Approach	16.408	4800

Table 1c. Index sizes in Experiment 3

	Size (MB)	Total Number of Records
Our Approach	20.616	9600
Alternative Approach	40.812	9600

The execution times of the two approaches for each query is given in Tables 2a, 2b, and 2c for each of the three experiments. The graphical outputs of the experiments are given in Figure 12 and Figure 13.

Table 2a. Query execution times (in msec) in Experiment 1

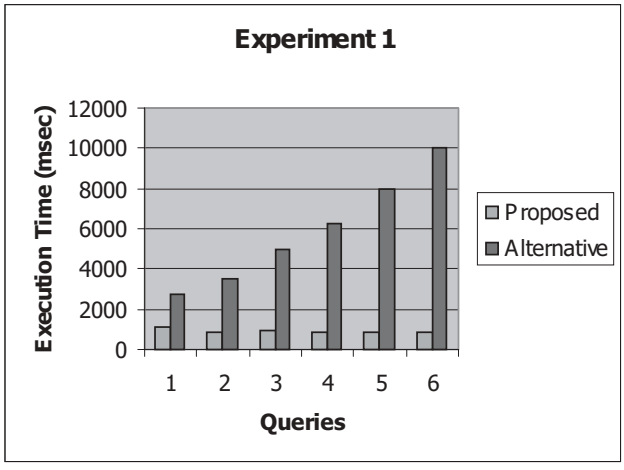
Query #	Our Approach	Alternative Approach
1	1072	2744
2	822	3545
3	951	4947
4	851	6219
5	862	7991
6	851	10014

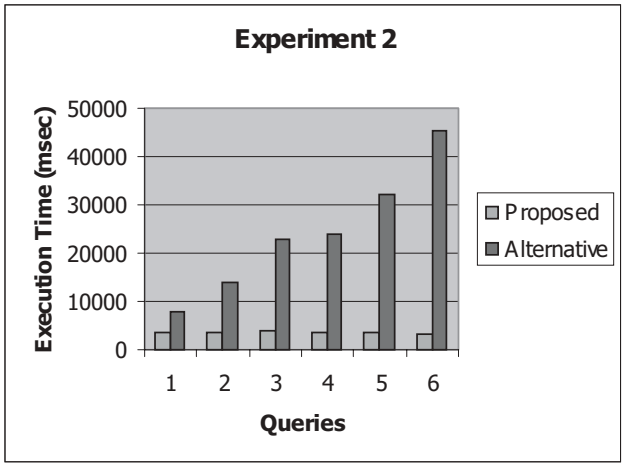
TABLE 2b. Query execution times (in msec) in Experiment 2

Query #	Our Approach	Alternative Approach
1	3715	8011
2	3665	13970
3	3805	22913
4	3585	23925
5	3635	32076
6	3315	45522

Table 2c. Query execution times (in msec) in Experiment 3

Query #	Our Approach	Alternative Approach
1	7762	19458
2	6639	27920
3	7451	40108
4	6549	50112
5	6400	61338
6	14931	> 100000





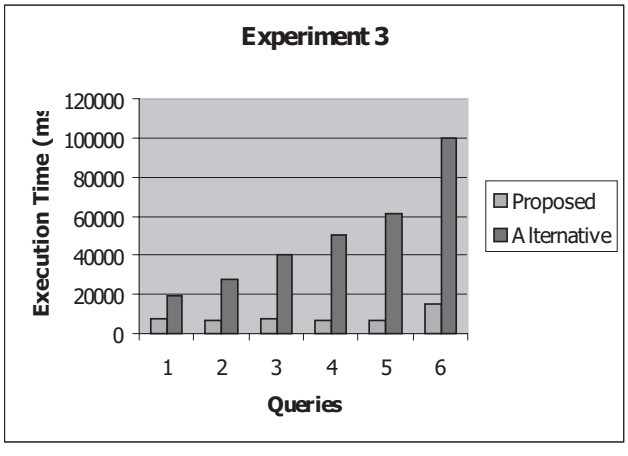
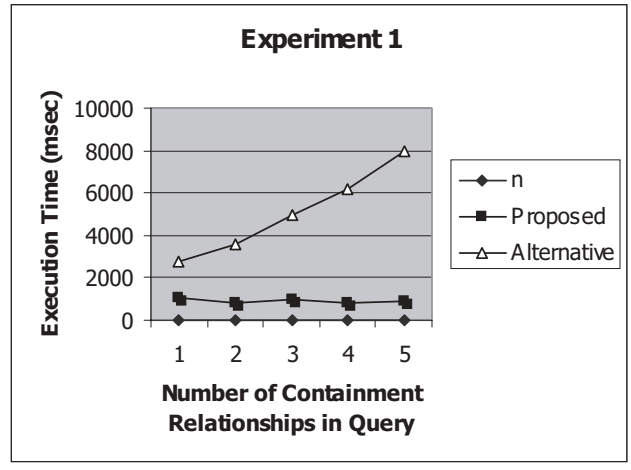
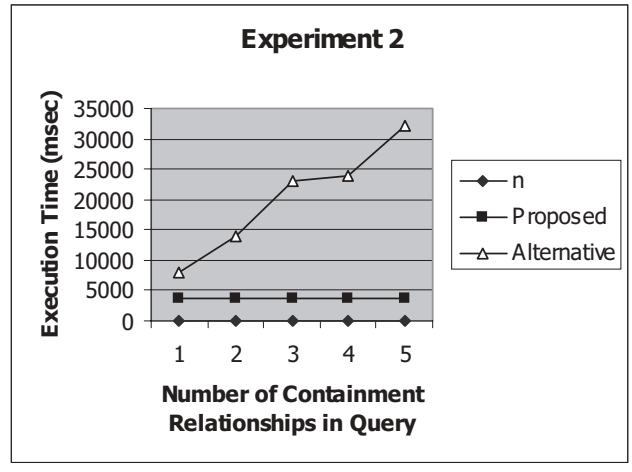


Figure 12. Performance comparison of our proposed approach and the alternative approach

The experiments show that the alternative approach introduces a serious performance problem in the processing of containment queries as expected. Considering the example query */article/artheader/title/"Database"*, the query processing using the alternative approach contains three join operations, one for each direct containment relationship in the query. First, "article" and "artheader" direct containment is evaluated by a self-join on "Elements" relation with the conditions *article.docno = artheader.docno and article.begin < artheader.begin and article.end > artheader.end and article.level = artheader.level – 1*. Similarly, "artheader" and "title" direct containment is evaluated by a self-join on "Elements" relation. Finally "title" and "Database" direct containment is evaluated by a join between "Elements" relation and "Texts" relation. In this example, three join operations are required, since the path length of the query is three. As seen, the number of the join operations required to evaluate the whole query is equal to the path length of a query. This bottleneck becomes even worse when the XML documents are big in size.

On the other hand, our proposed approach requires only one join operation between the PathIndex relation and the ValueIndex relation, regardless of the number of containment relationships between two elements, i.e. the path length of the query. This contrast between the two indexing approaches is shown in Fig. 13. The x-axis is the number of containment relationships in Queries 1, 2, 3, 4, and 5 respectively. The query execution time in the alternative approach increases drastically with the increase in the path length of the query. Whereas in our approach it almost remains the same.





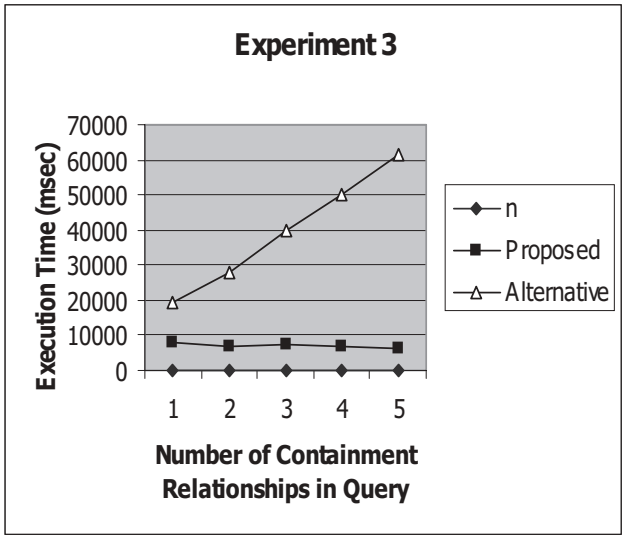


Figure 13. Execution times according to path length of queries

6. Conclusions

We expect that in the near future, a large and increasing amount of XML-based technical and scientific articles will be available on the Web. In this paper, we give a summary of an XML-based information retrieval system used in retrieving scientific and technical articles distributed on the Internet.

Indexing is very important for the efficiency of information retrieval systems. In this paper, we propose a novel indexing scheme for efficient XML-based article retrieval. The proposed indexing scheme takes into account the structure of XML-based articles and the requirements of XML-based article retrieval. Containment queries are important parts of queries processed by XML-based information retrieval systems. The indexing scheme that we propose is designed to process containment queries efficiently.

We compared our indexing approach with a most likely alternative approach. Our approach outperforms the alternative approach in all the six experimental queries and 3 different-sized article collections. Moreover, it turns out that the performance ratio of our proposed approach to the alternative approach increases drastically as the path length, i.e. the number of containment relationships, of the queries increases.

References

Barber, D., (2001). Barber, D., Electronic Publishing: A Guide To Electronic Journal Archives. Online Journal of Issues in Nursing 2(11) April. http://www.nursingworld.org/ojin/.

Bourret, R., 2002. Bourret, R., XML and Databases, http://www.rpbourret.com/xml/XMLAndDatabases.htm/.

Bray, T., (2000). Bray, T., P. Jean, C. M. Sperberg-McQueen & E. Maler, Extensible Markup Language (XML) 1.0 (Second Edition), World Wide Web Consortium, http://www.w3.org/TR/REC-xml/.

Clark, J. and DeRose, S., (1999). Clark, J. and DeRose, S., XML Path Language (XPath) Version 1.0, World Wide Web Consortium, http://www.w3.org/TR/xpath/.

DocBook Org., (2002). DocBook DDT, DocBook Org., http://www.docbook.org/.

Egnor, D., Lord, R., (2000). Lord, Structured Information Retrieval using XML, XYZFind Corp., http://www.haifa.il.ibm.com/sigir00-xml/final-papers/Egnor/

Klettke, M. (2001). Klettke, M., "Optimisation Techniques for Mapping Documents to Databases", XML DevCon 2001.

McKeown, J., & Jung, B., (2001). Electronic Publishing with XML, http://www.xml.com/.

Pembe, F. C., (2002). Index Structure Design and Performance Evaluation for XML-Based Publications, Computer Engineering Dept. report, Boaziçi University, Istanbul, Turkey, June 2002.

Seo, C. and Kim, H.-J., (2002). An Efficient Inverted Index Technique for XML Documents using RDBMS, Seoul National University, http://oopsla.snu.ac.kr/~cyseo

Tanabe, H. and Wang, C.C. (2000) XML-Based Dynamic Search Agent for the Internet Information Retrieval, Thesis, Carnegie Mellon University Information Networking Institute.

Walsh, N., (1998). A Technical Introduction to XML, ArborText, Inc.

Zhang, C., Naughton, J., DEWitt, D., Luo, Q., and Lohman, G.(2001). On Supporting Containment Queries in Relational Database Management Systems. Proceedings of the ACM SIGMOD International Conference on the Management of Data.

XML Europe 2001 Conference, (2001). XML Europe 2001 Conference, Graphic Communication Association, http://www.gca.org/papers/xmleurope2001/.

Yahoo Inc., http://www.yahoo.com/