

High-performance XML Storage/Retrieval System

●Yasuo Yamane ●Nobuyuki Igata ●Isao Namba

(Manuscript received August 8, 2000)

This paper describes a system that integrates full-text searching and database technologies for storing XML (eXtensible Markup Language) documents and retrieving information from them while providing a uniform interface. Our main goal with this system is to achieve high-performance, because there will be a large amount of XML documents in the near future if XML becomes a standard for structured documents and data exchange. We have therefore developed techniques for achieving high-performance storage and retrieval of XML documents. For full-text searches, we improved the Structure Index + Text Index model, which references both indexes alternately at retrieval. In our improved method, a hierarchical structure query is converted into a flat structure query by referencing just the structure index, then the optimized query can be quickly processed using only the text index. For storage, we developed an offset space, which is an address space in secondary memory that can compactly store any structure, for example, a tree. We use the offset space to solve the problem that occurs in other methods which store the analyzed result of XML documents as multiple relations in an RDB. In our method, the analyzed result can be stored in a single page in the best case. This makes it superior to other methods which store the analysis results in multiple relations so that storage of N relations needs at least N pages. As a result, generally, our method greatly reduces I/O costs.

1. Introduction

For various reasons, XML (eXtensible Markup Language) is expected to become a standard for structured documents and data exchange and the next-generation HTML (Hyper Text Markup Language). XML applications are evolving from document exchange, which was XML's first purpose, to the storage, retrieval, and utilization of documents.¹⁾ This evolution is occurring because XML has the following characteristics:

- 1) Tractability: XML is text, not binary, so it can be easily referenced and updated by humans.
- 2) Structure: XML is structured and information can be added to it by adding tags.
- 3) Independence between data and style
- 4) Extensibility: XML can define new types of tags.

- 5) Openness: XML is independent of specific vendors.

- 6) XML can be used with Web technologies.

XML is especially favored as the next-generation HTML because HTML does not have characteristics 3) and 4) above. We therefore expect that, just as there is now a large volume of accumulated HTML documents, there will be a large amount of XML documents in the future. In addition, as automatic tagging in natural language processing technology becomes more mature, it will become easier to reuse common digital documents as XML documents and the volume of XML documents will grow at a faster rate. We therefore forecast that high-performance retrieval and storage will be more necessary in the near future.

As shown in **Figure 1**, an XML document is text, but it can include links to other types of data, for example, multimedia data. Because of these links, like HTML documents, an XML document is inherently usable with other types of data. Users should be able to use XML documents and these other types of data in an integrated and uniform manner. Otherwise, they will need to integrate certain systems and APIs to handle XML documents and other types of data.

2. Problems to solve

Quick retrieval of information from a large amount of XML documents requires a full-text search technology which builds and uses indexes consisting of pairs of keywords and document identifiers. However, because database systems usually use a string search technology which parses strings dynamically, they are very slow when there is a large amount of strings and are therefore unsuitable for searching through a large amount of XML documents. Historically, these two technologies and the systems which implement them have developed separately. We therefore need to integrate the technologies of full-text searches and databases to achieve high-performance XML retrieval and storage. Also,

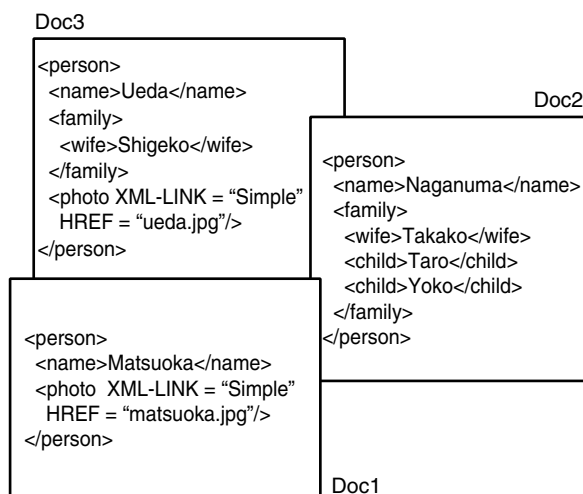


Figure 1
An example of XML.

when integrating these two heterogeneous systems, it is desirable that the interface looks uniform to users.

Regarding the storage of XML documents, often XML documents are analyzed and then the analysis results (hereafter, referred to as an “analyzed tree”) are stored. To store the analyzed trees, it has been proposed that they be stored in an RDB.²⁾ Various ways to store the trees in an RDB can be considered; one of them is shown in **Figure 2**. In this method, an analyzed tree is stored among relations, and links between tuples are traversed using indexes or these relations are joined. That is, the analyzed tree of an XML document is segmented and stored in many pages, which makes the I/O costs high at retrieval. Therefore, we think that the analyzed tree should be stored in as few pages as possible.

Because ordinary full-text search engines can only be used for plain text, they cannot exploit XML’s ability to represent repetition and nesting. Therefore, to achieve a high performance, we must adapt these search engines to XML. We describe this problem in detail in Section 4.2.

To increase the processing speed for the re-

Person relation

Id	Name	Family	Photo
1	Matsuoka		1
2	Naganuma	1	
3	Ueda	2	2

Family relation

Id	Wife	Child
1	Takako	1
2	Shigeko	

Child relation

Id	Child	Next
1	Taro	2
2	Yoko	

Attribute relation

Id	X-Link	HREF
1	Simple	matsuoka.jpg
2	Simple	ueda.jpg

Figure 2
XML representation by relation.

trieval and storage of XML documents, it will be necessary to solve the above problems.

3. Uniform interface and system architecture

To present a uniform interface to users, we selected relations (tables) which are almost the same as the relations of a relational database³⁾ (**Figure 3**) and are easy to understand and use. For the data types, we supported the array type for multiple values and the XML type in addition to the integer, string, and variable (variable length) data types.

An XML document is stored into or retrieved from an XML type field as text, so the contents of such a field superficially look like text. However, when it is stored, the contents are analyzed by an XML parser and stored as an analyzed tree. Then, when the contents are retrieved, the analyzed tree is used to compose an XML document. Retrieval operations on XML documents are performed as a selection operation on the field. This makes it possible to retrieve requested information as a set, because generally there are multiple XML documents in a relation.

Figure 4 shows our system architecture for implementing the interface mentioned above. The Relation Engine (RE) performs basic operations for relations and supports the basic operations of

database systems, for example, B-tree, buffering, transaction, and recovery.

In the RE, each data type is an independent module, and a new data type can be added. The XML type was added using this mechanism. The XML type includes a full-text search engine which was improved to cope with the hierarchical characteristics of XML documents. This full-text search engine consists of a structure engine and a text engine. The structure engine is used to parse XML documents and manage structure information about them in the structure index. The text engine searches the required XML documents using the text index. The XML type also includes the XML storage, which is used for storing and retrieving the analyzed trees to and from the repository. In the case of retrieval, a query is passed to the XML type through the RE, the text search engine searches for the required documents, and the required parts of the required documents are extracted by the XML storage from the analyzed tree. Then, the result of the query is passed from the XML type to the user through the RE.

4. Implementation

As mentioned in Chapter 3, full-text search technology is built into the XML type and a uniform interface is realized by using a relational interface. In this chapter, we explain the tech-

Data type
Integer String Array (String) XML Variable




Id	Name	Hobby	Profile	Photo
1	Matsuoka	baseball	<person> <name>Matsuoka</name> <photo XML-LINK = "Simple" HREF= "matsuoka.jpg"/> </person>	
		jazz		
2	Naganuma	soccer	<person> <name>Naganuma</name> <family> ...	
3	Ueda	golf	<person> <name>Ueda</name> <family> ...	
		tennis		

Figure 3
A relation as a user interface.

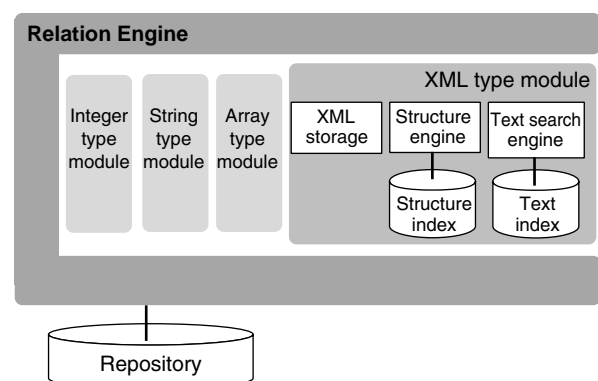


Figure 4
System architecture.

niques we developed for high-performance XML document storage and retrieval.

4.1 XML storage

To solve the RDB storage problem described in Chapter 2, we identified two important requirements regarding the storage of analyzed trees.

- 1) Links must be traversed efficiently.
- 2) Analyzed trees must be stored in as few pages as possible.

The basic idea is to construct a space, called an “offset space,” which is similar to a main memory space on a secondary storage, and then use it to store the analyzed tree.

An offset space is addressed using an offset ranging from 0 to 2 G. Generally, as shown in **Figure 5**, an offset space consists of multiple pages whose sizes range from 4 KB to 1 MB. In this regard, an offset space is very similar to a main memory space; it also has characteristics common to the file system of UNIX. We can obtain an area in this space in a way similar to the way used for main memory. Areas are obtained sequentially starting from the top page to maintain clustering (i.e., to keep them as close as possible on a disk). When an offset space overflows, new areas are obtained in the next page. Offsets are used as pointers to link the data structures, so we can use them to construct complex structures such as trees

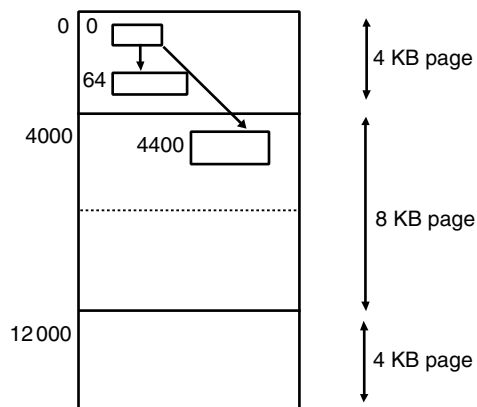


Figure 5
Offset space.

and networks. Using this mechanism, our analyzed tree can easily be stored, as shown in **Figure 6**. Part (a) of this figure shows the analyzed tree of Doc1 in Figure 1, and part (b) shows the analyzed tree of Doc 2. Later, this analyzed tree is improved further. Usually, the processing to traverse these links by offsets is done in a page, which makes the processing very fast and meets the first requirement described above.

Our analyzed tree has a structure that reflects the hierarchy of XML; it consists of various structures such as the node, content, attribute, and string. A node corresponds to a tag, a content to the content of a tag, and an attribute to an attribute. A feature of this structure is that we can store it in a single page in the best case. This differs from the RDB method mentioned above, achieves good clustering, minimizes I/O costs, and meets the second requirement of storage in as few pages as possible. We further improved the analyzed tree by:

- 1) Constructing the same types of structures as an array,
- 2) using the number of an array element as a link instead of an offset, then making a link include the type of the linked structure, and
- 3) constructing node lists corresponding to the kinds of tags.

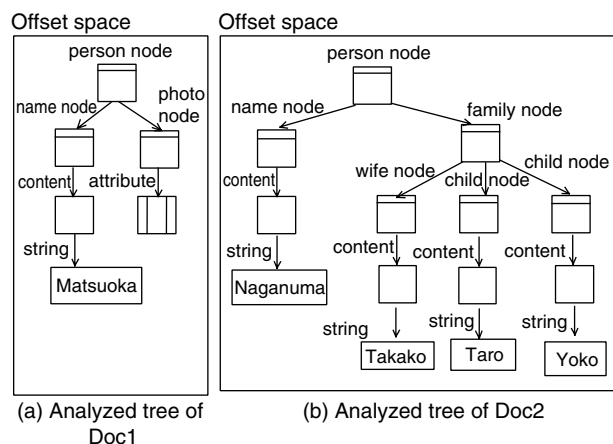


Figure 6
Representation using offset space.

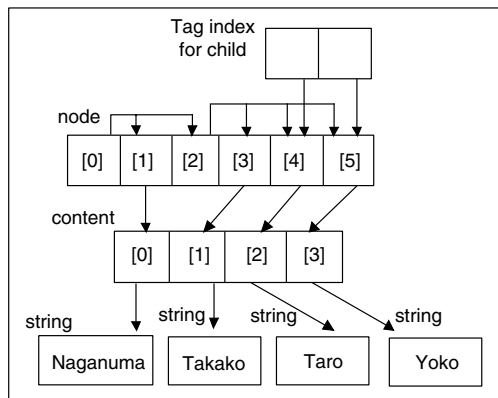


Figure 7
Improved stored analyzed tree.

That is, we improved the method shown in Figure 6 (b) into the method shown in **Figure 7**. The merit of improvement 1) described above is that because areas are grouped together, they can be quickly obtained.

On secondary storage, we often reconstruct pages when many areas are deleted or clustering is poor. The merit of improvement 2) is that it makes it unnecessary to update the values of links when moving structures and it enables us to compress the values of links. This is because when links are implemented by offsets, the values of links must be updated but the numbers of array elements are invariable when the structures are moved. An offset is represented by 4 bytes and the number of an array element is usually a small integer, so it is possible to compress them (4 bytes to 1 byte in the minimum case).

When searching analyzed trees, there are many cases where the specified tags must be accessed; for example, when the value of tag <child> is requested. It is expensive to traverse from the root of the analyzed tree every time this is requested. Improvement 3) enables direct access to the structure corresponding to the requested tag. Often the same tags are repeated, which makes this mechanism more effective.

If the analyzed tree is large, it is stored across many pages, and it then becomes a problem how to connect them. For example, if the pages are

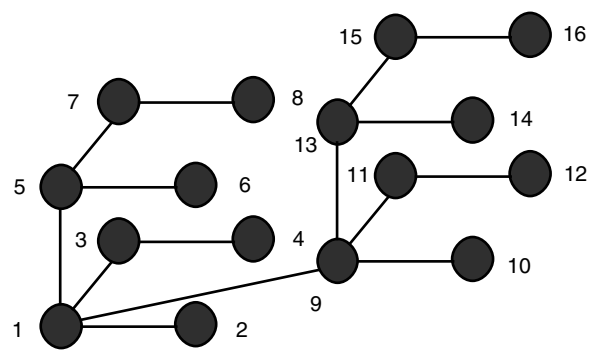


Figure 8
Hypercube structure of offset space.

linked in a list, the average I/O cost to access each page is very large, but the space for links is very small. On the other hand, for example, if we let the first page have links to other pages, the average I/O cost is very small, but the size of the area for the first page is decreased and may even become too small to contain all the links. To solve this problem, we adopted a method for connecting pages as a hypercube as shown in **Figure 8**. We chose a hypercube structure because it can achieve a good balance between the number of links and the average distance from the root page to other pages. (The hypercube is used in parallel computers for a similar reason; namely, because it achieves a good balance between the average distance and the number of links.)

4.2 Retrieval method of structured document databases

In the field of information retrieval, research has been done on retrieval methods for structured documents since the Standard Generalized Markup Language (SGML) appeared in 1986.

However, regarding the new retrieval methods that have come from this research, R. Baeze-Yates has stated that “They are not in general as mature as the classical ones. Not only they lack the long process of testing and maturing that traditional models have enjoyed, but also many of them are primitive as software systems, having been implemented mainly as research

prototypes.”⁴⁾

The retrieval methods proposed so far are briefly described below.

- **Hybrid model**⁵⁾

This method divides the document into fields and registers the field information in an index with a document ID and positional information. Many full-text search engines adopt this model because it is the simplest technique and its implementation is also easy. However, because the field is a flat structure, the Hybrid model cannot support a hierarchical structure search.

- **Overlapped Lists model**⁶⁾

This method registers structural area information collectively referred to as a “region” in an index, as well as the usual word information. This method needs not only Boolean algebra but also region algebra.

- **Structure Index + Text Index model**⁷⁾

This method has two kinds of indices: a structure index for the structure of documents and a text index for the content of documents. This method leads to a complex system configuration and query processing, but it can handle hierarchical structure queries.

4.2.1 Our system design goals

When we designed the structured document search engine, we considered the following points.

- **Large-scale document databases**

The system should be able to support a large-scale document database.

For instance, a retrieval method that uses region algebra cannot be expected to achieve high-speed retrieval in a large-scale document because the number of comparisons of positional information increases as the number of target documents increases.

- **Hierarchical structured query processing**

The system should support not only a flat document structure, but also a hierarchical document structure. The Structure Index + Text Index model can process more complex

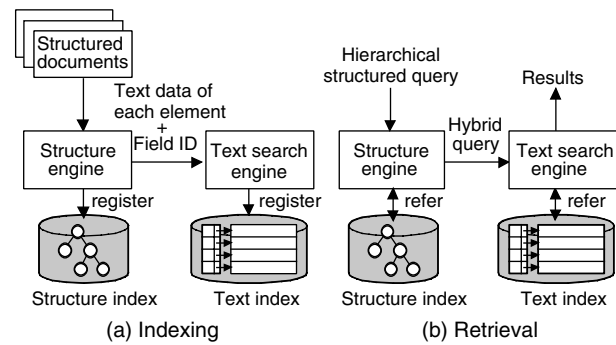


Figure 9
Structure Index + Text Index model.

queries than the other methods.

- **Ease of implementation**

To make it easy to implement, the system should use existing resources as effectively as possible. The implementation cost can be reduced if we can use the framework of a traditional full-text search engine without modification.

4.2.2 Our model and indexing and retrieval technique

We adopted the Structure Index + Text Index model with certain modifications.

The problem with the unmodified model is that the system configuration and processing become complex, which results in a slow processing speed. Especially, in retrieval processing, the system needs to refer to the structure index and text index alternately and store temporary data. Consequently, the retrieval speed is decreased.

We solved this problem by developing a technique by which a hierarchical structure query is converted into a flat structure query by referencing just the structure index and then the optimized query is quickly processed using only the text index.⁸⁾

- **Indexing technique**

The basic idea of the indexing technique is to allocate a field ID to each text data item of the XML element and to register it in the structure index and text index (Figure 9 (a)).

The structure index manages the hierarchi-

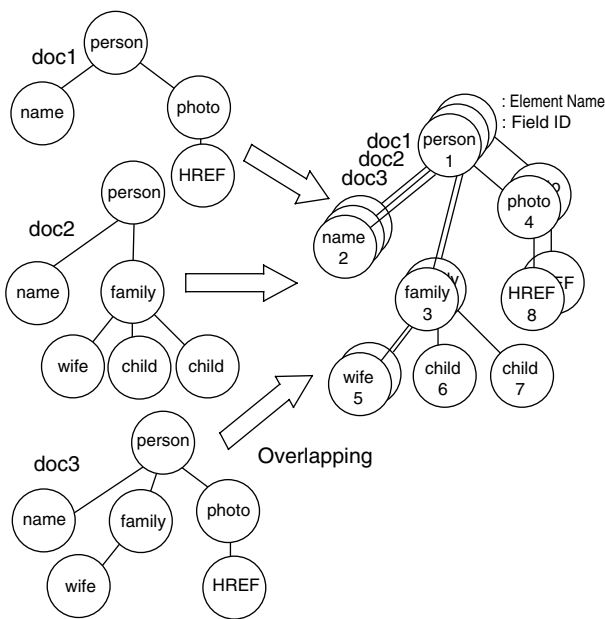


Figure 10
Example of a structure index.

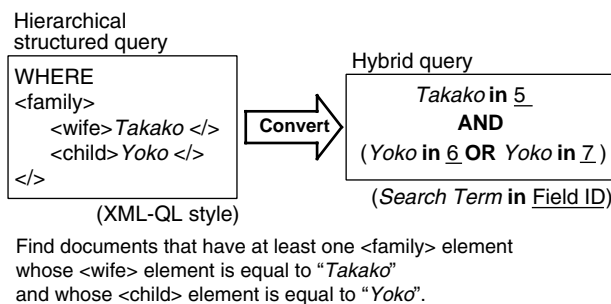


Figure 11
Query conversion.

cal structure of each field, and the text index manages the field ID and document ID in which query words appear.

The structure index is one big data tree and represents the overlapped structure of documents (**Figure 10**). Each node in the data tree corresponds to a field. Overlapping of the structures of documents is judged according to rules, for example, rules about the positions of elements in a document, element names, and brother elements.

The text index uses the same data structure as the Hybrid model in a traditional full-text search engine. It stores the document IDs and

field IDs in which query words appear.

• Retrieval technique

In retrieval processing, the system makes one reference each to the structure index and text index (**Figure 9 (b)**). The retrieval procedure is as follows.

- 1) The hierarchical structured query is expressed by the tree data. The structure of the query is verified by matching the tree data of the structure index.
- 2) A hierarchical structured query is converted into a flat structure query (hybrid query) using the field ID of the node in the structured index (**Figure 11**).
- 3) Sets of document IDs that match the query using the text index are retrieved. This step is the same as the last step of the Hybrid model search engine.

5. Conclusion

As discussed above, we think that our system can uniformly cope with various types of data, including XML documents, by combining full-text search technologies and database technologies. Also, we think that it can process a large amount of XML documents at high speed.

For high-performance retrieval, we developed a technique in which the hierarchical structure query is converted into a flat structure query by referencing just the structure index and then the optimized query is quickly processed using only the text index. As Figure 11 shows, a processing structured query created via hybrid query conversion requires an OR execution of many terms, which slows down the processing. However, our experiment on actual data shows that, on average, the processing is only five times slower than flat query processing. Our system can process 20 queries per second for a flat structure query of 1 GB of Japanese data. Considering the complexity of structured query processing, we think our approaches for structured queries is plausible.

Also, by storing the analyzed tree of XML documents in an offset space, we solved the problem

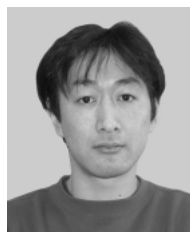
that occurs when they are stored in an RDB. If an analyzed tree is stored in N relations, it can cost more than N pages to retrieve it; however, in our method, retrieval costs a single page in the best case, which we think leads to good efficiency. The merit of using an RDB is that it enables the use of mature technology, for example SQL. As yet, our method has been implemented only at the function-level and is primitive. A future task, therefore, will be to make our system easier to use.

References

- 1) H. Ishikawa: XML and Databases – Expansion from Exchange to Storage and Query. (in Japanese), *IPSJ Magazine*, **41**, 1, pp.68-73 (2000).
- 2) J. Shanmugasundaram et al.: Relational Databases for Querying XML Documents: Limitations and Opportunities. 25th Intl. Conf. on VLDB, pp.302-314, 1999.
- 3) E. F. Codd: A Relational Model of Data for Large Shared Data Banks. *Comm. ACM*, **13**, 6, pp.377-387, 1970.
- 4) R. Baeze-Yates and G. Navarro: Integrating Contents and Structure in Text Retrieval. *ACM SIGMOD Record*, **25**, 1, pp.67-79 (1996).
- 5) R. Baeze-Yates: A Hybrid Query Model for Full Text Retrieval System. Technical Report DCC-1994-2, Dept. of Computer Science, Univ. of Chile, 1994.
- 6) C. Clarke, G. Cormack, and F. Burkowski: An Algebra for Structured Text Search and a Framework for its Implementation. *The Computer Journal*, **38**, 1, pp.43-56 (1995).
- 7) G. Navarro and R. Baeza-Yates: Proximal Nodes: A Model to Query Document Databases by Content and Structure. *ACM Trans. On Information System*, **15**, 4, pp.400-435 (1997).
- 8) N. Igata and I. Namba: A Method of Indexing and Searching for Large Scale Structured Document Database. (in Japanese), Laboratory report of Information Processing Society of Japan, 2000-FI-57, pp.9-16, 2000.



Yasuo Yamane received the B.S. degree in Pure and Applied Science and the M.S. degree in Coordinated Science from Tokyo University, Japan in 1979 and 1981, respectively. He joined Fujitsu Laboratories Ltd., Japan in 1981 and has been engaged in research and development of distributed databases, parallel databases, object-oriented databases, and document repositories. He is a member of the Information Processing Society of Japan (IPSJ) and a member of the Institute of Electronics, Information and Communication Engineers (IEICE) of Japan.



Nobuyuki Igata received the B.S. degree in Mechanical Engineering and the M.S. degree in Information Sciences from Tohoku University, Sendai, Japan in 1993 and 1995, respectively. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1995 and has been engaged in research and development of natural language processing and information retrieval systems. He is a member of the Information Processing Society of Japan (IPSJ).



Isao Namba received the B.S. and M.S. degrees in Linguistics from Kyoto University, Kyoto, Japan in 1987 and 1989, respectively. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1989 and has been engaged in research and development of natural language processing and information retrieval systems. He is a member of the IEEE and ACM.