ACM First International Conference on Digital Libraries, Bethesda, Maryland, March 1996

Index Structures for Structured Documents *

Yong Kyu Lee, Seong-Joon Yoo, Kyoungro Yoon
School of Computer and Information Science
Syracuse University

P. Bruce Berra
Dept. of Electrical and Computer Engineering
Syracuse University

Abstract

Much research has been carried out in order to manage structured documents such as SGML documents and to provide powerful query facilities which exploit document structures as well as document contents. In order to perform structure queries efficiently in a structured document management system, an index structure which supports fast document element access must be provided. However, there has been little research on the index structures for structured documents. In this paper, we propose various kinds of new inverted indexing schemes and signature file schemes for efficient structure query processing. We evaluate the storage requirements and disk access times of our schemes and present the analytical and experimental results.

1 Introduction

Since the Standard Generalized Markup Language (SGML) [13] [15] was standardized, many structured document management systems have been built to manage structured documents including [1] [2] [3] [4] [5] [6] [17] [18] [20] [21] [23]. In those systems, structure queries as well as content queries are supported. The content query is based on the content of documents. For example, the query that finds documents which contain a specific keyword is a content query. The structure query is based on the hierarchical logical structure of documents. Thus, we can ask questions based on the logical structure of documents such as chapters, sections, or subsections. The structure query can be combined with the content query to build a more powerful query.

Much research has been performed to design efficient index structures for database and information retrieval systems [7] [10] [11] [12] [14] [24] [25]. In order to perform structure queries efficiently in the structured document management system, an index structure that supports fast element access must be provided because users want to access any kind of document element in the database. However, in the previous systems, little attention has been focussed on structured document indexing.

Recently, Sacks-Davis et al. [23] have proposed some possible inverted index structures for structure query processing based on the conventional inverted index. The first approach is to maintain the inverted index supporting only document access. That is, the inverted list contains only document identifiers, and no element identifiers. If users want to access document elements, document identifiers are first obtained from the document index, and then the elements satisfying the query are identified from the retrieved documents. The disadvantage of this approach is that post-processing is required to locate document elements after identifying documents. The cost of the post-processing is considerable if the number of documents to be retrieved is large. To reduce the post-processing cost, an internal tree representation [20] can be used. However, the storage overhead is great.

The second approach is to maintain a separate index for each element type. However, the index term should appear in every index causing considerable space overhead. In order to reduce the duplication of index terms, a single combined vocabulary with a separate inverted list for each element type per index term can be used. For example, an index term "multimedia" now appears in only one combined vocabulary. However, the storage overhead of the inverted list caused by the element identifiers is still great. For example, if a paragraph

*This work was supported in part by the Electronics and Telecommunications Research Institute of Korea and the New York State Center for Advanced Technology in Computer Applications & Software Engineering.

contains a keyword "multimedia," the inverted lists for the keyword "multimedia" will contain all the element identifiers in the path from the root to the paragraph.

Finally, they have proposed a new scheme, called element locator scheme, for document structure indexing. In this approach, each index term in the inverted list is associated with a path encoding from the root to the leaf and any necessary sibling numbers. This approach also requires considerable storage overhead in the inverted list.

In this paper, we propose new inverted index and signature file schemes for structured documents which reduce the storage overhead considerably. Our schemes use specially designed unique element identifiers (UID's) in order to reduce the number of index entries. By letting the UID carry information about the document structure, we can obtain the UID's of the ancestors and descendents directly from the UID of an element. Thus, we do not need to store all the element identifiers containing a keyword in the index unlike the previous schemes.

2 Document Structure Query

It is important to have some queries based on the logical structure of the structured documents such as SGML documents. The structure query can be a simple structure query that can be resolved by using DTD's (Document Type Definition) only and complex structure query which is combined with the content query. For example, the query that finds documents which have specific elements such as chapter and section is a simple structure query. In the complex structure query, users can find sections whose first paragraph contains a specific keyword. Structure queries are very useful since users can retrieve any parts (elements) of documents based on the combination of the content and structure of documents. The following shows some examples of the structure query.

- Find sections that have a subsection containing the keyword "hypermedia."
- Find the first author of the documents whose conclusion includes the keyword "index."
- Find the images referenced by the last section of an article containing the keyword "multimedia."
- Select all nodes that have children.
- Find the children and grandchildren elements of this_element.

In order to facilitate structure query processing, an index structure supporting fast element access must be provided.

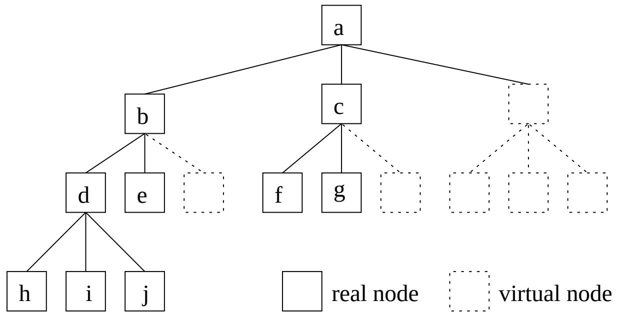

Figure 1: Example Document Tree

Table 1: Unique Element Identifiers

element	UID	element	UID
a	1	f	8
b	2	g	9
c	3	h	14
d	5	i	15
e	6	j	16

3 Document Element Identifier

Our indexing schemes use specially designed UID's in order to reduce the index entries. We interpret a document structure as a k-ary tree [9] and assign each element (node) a UID according to the order of the level-order tree traversal. For example, for the document tree of Figure 1, we assign UID's as shown in Table 1 assuming a 3-ary tree. Here, we can consider the node a is a *book*, b a *chapter*, d a *section*, and h a *subsection*. Because we assume the tree is complete, there are some virtual nodes which do not exist.

The UID's of the parent and j-th child of a node whose UID is i can be obtained by the following functions.

$$parent(i) = \left\lfloor \frac{(i-2)}{k} + 1 \right\rfloor \quad (1)$$

$$child(i, j) = k(i-1) + j + 1 \quad (2)$$

It is possible to use the tree location address of the HyQ [16] as the UID. In this case, the UID is a concatenation of the children identifiers in the path from the root to the node. For example, the tree location address of the node which is the fourth child of the third child of the second child of the root is '1 2 3 4'. We can use '2 3 4' as the UID of the node. Here, we need not include the root identifier since it is always '1'. From the UID, the UID's of the ancestors and descendents can be obtained using the shift operations.

4 Indexing Structured Documents

Since users can access any kind of document element in the document tree, it is necessary to build an index structure that facilitates access to an element in the database. We propose various kinds of new inverted index structures and signature file structures which support fast access to document elements.

4.1 Inverted Index

In order to support direct element access, it is required to include all the index terms of elements in the inverted list. In the document tree, leaf nodes are associated with data while internal nodes represent only structure relationships between document elements.

Suppose that a document has three leaf nodes and each leaf element has index terms as shown in Figure 2. Here, we can consider the node A is a chapter, B and C are sections, and D and E are subsections.

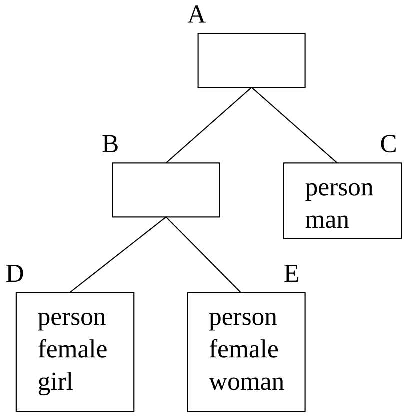

Figure 2: Document Tree with Index Terms

Even though the internal nodes have no associated document data, the data at the subtrees must be considered as their data. Thus, the index terms for each node are as follows:

```
index(A) = {person,female,man,girl,woman}
index(B) = {person,female,girl,woman}
index(C) = {person,man}
index(D) = {person,female,girl}
index(E) = {person,female,woman}
```

Now the problem is how to maintain an inverted index structure for fast access to document elements.

4.1.1 Inverted Index for All Nodes with Replication (ANWR)

The first naive approach is to replicate all index terms of the children to their ancestor nodes as shown in Figure 3.

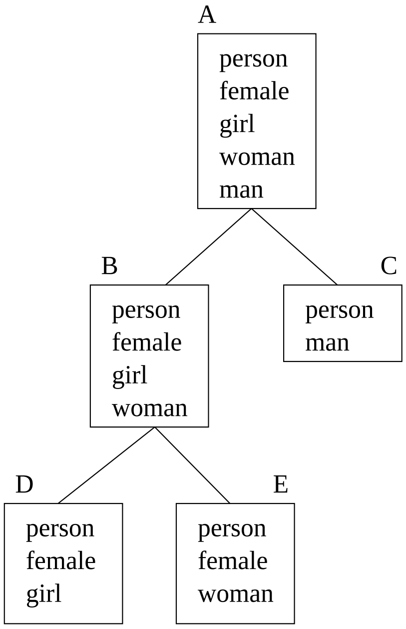

Figure 3: Indices for All Nodes with Replication

In this scheme, the inverted list for an index term must include all the UID's of the elements which contain it. The inverted list for the example is as follows:

```
invert-list(person) = {A,B,C,D,E}
invert-list(female) = {A,B,D,E}
invert-list(man) = {A,C}
invert-list(girl) = {A,B,D}
invert-list(woman) = {A,B,E}
```

Using the inverted list we can access any element at any level in the document tree. However, this scheme causes many duplications in the inverted list.

4.1.2 Inverted Index for All Levels with Replication (ALWR)

This scheme is the same as the ANWR except that each element type has a separate inverted list per index term. A similar scheme has also been described in [23] without analysis. The inverted list for the example is as follows:

```
invert-list(person) = {{A},{B,C},{D,E}}
invert-list(female) = {{A},{B},{D,E}}
invert-list(man) = {{A},{C},{ }}
invert-list(girl) = {{A},{B},{D}}
invert-list(woman) = {{A},{B},{E}}
```

The inverted list for "person" has three sub-lists for three element types. This scheme has also considerable storage overhead in the inverted list.

4.1.3 Inverted Index for Leaf Nodes Only (LNON)

In this scheme, we include element identifiers in the inverted list for leaf nodes only as shown in Figure 4.

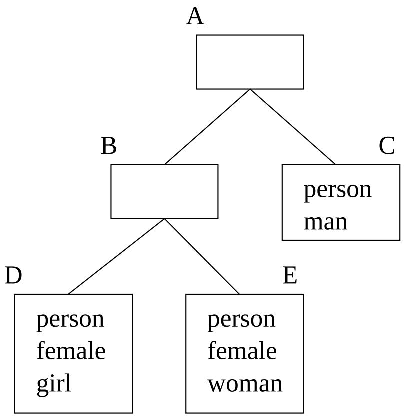

Figure 4: Indices for Leaf Nodes Only

The inverted list for the example is as follows:

```
invert-list(person) = {C,D,E}
invert-list(female) = {D,E}
invert-list(man) = {C}
invert-list(girl) = {D}
invert-list(woman) = {E}
```

The storage requirement of this scheme is much less than that of the ANWR. Even though we do not maintain the UID's of the internal nodes in the document tree, the UID's of the ancestor nodes of a node can be calculated by the *parent* function. Thus, we can access any document element using the inverted list.

4.1.4 Inverted Index for All Nodes without Replication (ANOR)

We propose a novel method to build the inverted list which saves more space. Figure 5 shows how to build an index for structured documents. In this scheme, we use the fact that the child nodes of a node can have some index terms in common.

By using this fact, we can construct the inverted list for the example as follows:

```
inverted-list(person) = {A}
inverted-list(female) = {B}
inverted-list(man) = {C}
inverted-list(girl) = {D}
inverted-list(woman) = {E}
```

In this scheme, the index of a node is

INDEX[node] ∪ INDEX[ancestors] ∪ INDEX[descendents].

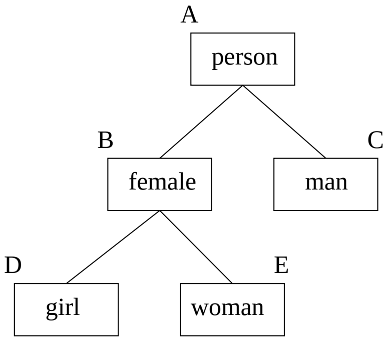

Figure 5: Indices without Replication

Using the ANOR inverted list, we can access any element in the database using the *parent* and *child* function.

4.1.5 Inverted Index for Root Node Only (RNON)

We can consider an extreme case which has been used in traditional information retrieval systems. In this scheme, we include only the element identifier of the root in the inverted list, that is, the document identifier only, as shown in Figure 6.

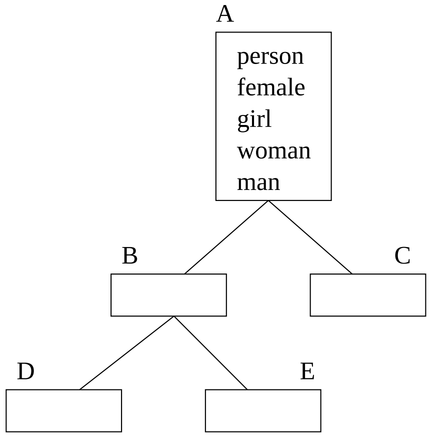

Figure 6: Indices for Root Node Only

The inverted list for the example is given below.

```
inverted-list(person) = {A}
inverted-list(female) = {A}
inverted-list(man) = {A}
inverted-list(girl) = {A}
inverted-list(woman) = {A}
```

However, this approach is inappropriate for structure query processing. That is, to process structure queries, we have to either access all related documents as a whole or maintain other data structures to identify satisfying element identifiers.

4.2 Signature File

The signature file is a useful indexing technique in information retrieval systems because its storage utilization is better than that of the inverted index [11] [19]. In order to support structure queries efficiently, we propose modified signature file structures.

4.2.1 Signature File for All Nodes (SFAN)

In this scheme, we maintain signatures of all elements in the document tree. However, it requires considerable space to maintain element signatures. Moreover, it is inefficient as the entire signature file must be accessed to process a query.

4.3 Signature File for All Levels (SFAL)

In this scheme, all element signatures of the document tree are maintained. However, unlike the SFAN, each element type is assigned a separate signature file in order to reduce the time needed to scan the signature file. Even though this scheme can access document elements very fast, it requires considerable space.

4.3.1 Signature File for Leaf Nodes Only (SFLN)

To reduce the space for the signature file, we maintain signatures of the leaf nodes only and do not maintain signatures of the internal nodes. This scheme is similar to the LNON inverted indexing scheme. The signature of the internal node can be calculated by ORing the signatures of its leaf nodes.

4.4 Signature File for Root Node Only (SFRN)

This scheme is the one used in traditional information retrieval systems. In this scheme, only the root node of the document tree has a signature and the other nodes do not have signatures. However, it is difficult to support structure queries in this scheme.

4.5 Signature File for Selected Levels (SFSL)

In order to reduce the time required to perform the bitwise OR operation in the SFLN, we can maintain signatures at some selected levels of the document tree. The SFLN and SFRN are the extreme cases of this scheme. As an example of this scheme, we can combine the SFLN and SFRN schemes, that is, we can maintain signatures for the root node and leaf nodes. We can also maintain three levels of signatures, for example, for the root node, leaf nodes, and middle level nodes.

5 Cost Comparison

We analyze the storage requirements and disk access times of the ANWR, ALWR, LNON, and ANOR inverted indexing schemes. The inverted index model which will be used for the analysis is shown in Figure 7.

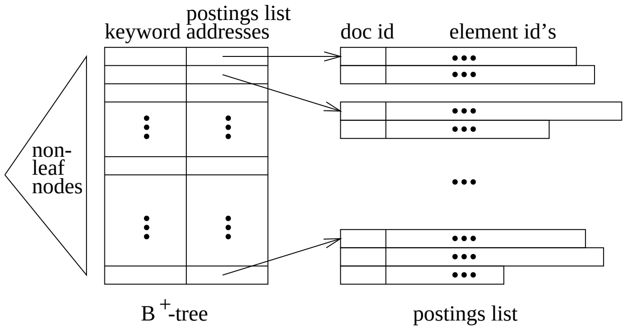

Figure 7: Inverted Index Structure for Structured Documents

The symbols of Table 2 are used in the space and time expressions representing the storage requirements and disk access times respectively.

Table 2: Symbols and Definitions

symbol	definition
d	degree of B^+-tree
h	height of document tree
k	degree of document tree
l	height of B^+-tree
m	average number of keywords in a node
n_{doc}	total number of documents in database
n_{invert_list}	number of inverted lists per index term
n_{key}	total number of keywords in database
p	rate of promoted keywords from children to parent node
s_{block}	block size in bytes
s_{doc_id}	document identifier size in bytes
s_{elem_id}	element identifier size in bytes
s_{entry}	table entry size in bytes
s_{key}	average keyword size in bytes
s_{ptr}	pointer size in bytes
s_{set}	set size of document variable
t_{random}	random disk block access time
t_{seq}	sequential disk block access time
u	rate of unique index terms in children nodes

5.1 Storage Requirement

The storage requirement of an index structure is the sum of those required for the B^+-tree and postings list.

$$S_{index} = S_{btree} + S_{posting}. \tag{3}$$

The B^+-tree has the leaf and internal nodes.

$$S_{btree} = S_{leaf} + S_{internal}. \tag{4}$$

Since the total number of keywords in the database is n_{key}, the number of disk blocks required for the leaf nodes is

$$S_{leaf} = \left\lceil \frac{(s_{key} + n_{invert\_list} * s_{ptr}) * n_{key}}{s_{block}} \right\rceil. \tag{5}$$

If we assume that the height of the B^+-tree is l and the degree (fan-out) is d, then the number of disk blocks for the internal nodes is

$$\begin{aligned} S_{internal} &= \sum_{i=0}^{l-2} d^i \\ &= \frac{d^{l-1} - 1}{d - 1}. \end{aligned} \tag{6}$$

The degree d can be calculated as follows:

$$d = \left\lfloor \frac{s_{block}}{s_{key} + s_{ptr}} \right\rfloor. \tag{7}$$

The storage requirement for the postings list for the database is

$$S_{posting} = \left\lceil \frac{n_{key} * S_{id}}{s_{block}} \right\rceil, \tag{8}$$

where

$$S_{id} = N_{doc\_per\_key} * s_{doc\_id} + N_{post\_per\_key} * s_{elem\_id}.$$

The number of postings per keyword is

$$N_{post\_per\_key} = \left\lceil \frac{N_{post\_per\_doc} * n_{doc}}{n_{key}} \right\rceil. \tag{9}$$

Now we calculate $N_{doc\_per\_key}$, the average number of documents per keyword. Here u is the rate of unique index terms among the child nodes which will be indexed in the parent node. The range of u is

$$\frac{1}{k} \le u \le 1.$$

If u is close to $\frac{1}{k}$, the document is very dense, that is, the elements of the document are closely related to each other and have very similar index terms. When u is large, the document is sparse, that is, the elements of the document are not closely related.

Assume that the average number of index terms of a leaf node is m. Then the number of index terms of a node at level (h - 1) is

$$k * m * u.$$

The number of index terms of a node at level i is

$$k^{h-i} * m * u^{h-i}.$$

Since the number of nodes at level i is k^{i-1}, the total number of index terms at level i is

$$\begin{aligned} & k^{i-1} * k^{h-i} * m * u^{h-i} \\ &= k^{h-1} * m * u^{h-i}. \end{aligned}$$

Since the number of index terms of a root node is

$$k^{h-1} * m * u^{h-1},$$

the number of documents per keyword is

$$N_{doc\_per\_key} = \left\lceil \frac{k^{h-1} * m * u^{h-1} * n_{doc}}{n_{key}} \right\rceil. \tag{10}$$

Now we calculate $N_{post\_per\_doc}$, the number of index terms per document, for each inverted indexing scheme.

ANWR:
The total number of index terms per document is

$$\begin{aligned} N_{post\_per\_doc} &= \sum_{i=1}^{h} k^{h-1} * m * u^{h-i} \\ &= k^{h-1} * m * \sum_{j=0}^{h-1} u^j \\ &= k^{h-1} * m + k^{h-1} * m * \sum_{j=1}^{h-1} u^j \\ &= k^{h-1} * m + k^{h-1} * m * \frac{u * (1 - u^{h-1})}{1 - u}. \end{aligned} \tag{11}$$

ALWR:
For the ALWR, we have to maintain as many inverted lists as the height of the document tree. The total number of index terms per document for the inverted list of level i is the same as the number of index terms at level i. Thus,

$$N_{post\_per\_doc} = k^{h-1} * m * u^{h-i}. \tag{12}$$

LNON:
The total number of index terms per document is

$$N_{post\_per\_doc} = k^{h-1} * m \quad (13)$$

since the number of leaf nodes in the tree is k^{h-1}.

ANOR:
For the analysis of the ANOR, we introduce a parameter p which represents the probability that an index term is promoted from the child nodes to their parent node. The range of p is

$$0 \leq p \leq 1.$$

Then, the number of index terms of a leaf is

$$m - m * p.$$

The number of index terms of a node at level i is

$$m * p^{h-i} - m * p^{h-i+1}.$$

The total number of index terms at level i is

$$k^{i-1} * m * p^{h-i} - k^{i-1} * m * p^{h-i+1}$$

because the number of nodes at level i is k^{i-1}. And the number of index terms of the root node is

$$m * p^{h-1}.$$

Thus, the total number of index terms of a document is

$$N_{post\_per\_doc} = \sum_{i=2}^{h} (k^{i-1} * m * p^{h-i} - k^{i-1} * m * p^{h-i+1})$$

$$+ m * p^{h-1}$$

$$= k^{h-1} * m - m * \sum_{i=1}^{h-1} p^{i} * \left(k^{h-i} - k^{h-i-1}\right). \quad (14)$$

5.2 Disk Access Time

The disk access time required to retrieve the postings list of a keyword is the sum of the B^{+}-tree access time and postings list access time.

$$T_{search} = T_{btree} + T_{posting}. \quad (15)$$

The disk access time of the B^{+}-tree is

$$T_{btree} = l * t_{random}. \quad (16)$$

The disk access time of the postings list of a keyword is

$$T_{posting} = t_{random} + (N_{block\_per\_key} - 1) * t_{seq}, \quad (17)$$

where the number of disk blocks of the postings list per keyword is

$$N_{block\_per\_key} = \left\lceil \frac{S_{posting}}{n_{key}} \right\rceil. \quad (18)$$

5.3 Analytical Results

We compare the storage requirements of the inverted indexing schemes by using the parameters as shown in Table 3. The disk access times are based on the Conner CP30200 model [8]. We assume that the database has 100,000 documents with 50,000 keywords.

Table 3: Parameter Settings

symbol	value
d	64
h	5
k	5
l	4
m	10
n_{doc}	100,000
n_{invert_list}	5 (ALWR), 1 (others)
n_{key}	50,000
p	varied
s_{block}	1024
s_{doc_id}	4
s_{elem_id}	2
s_{entry}	4
s_{key}	12
s_{ptr}	4
s_{set}	50
t_{random}	19 msec
t_{seq}	0.7 msec
u	0.6

Figure 8 illustrates the storage requirements of the ANWR, ALWR, LNON, and ANOR, when u is 0.6. It shows that the ANOR has the best performance while the ALWR has the worst. It shows that the index size of the ANOR decreases linearly as the value of p increases.

Figure 9 illustrates the average disk access times of the four schemes, when u is 0.6 and any arbitrary keyword index is accessed. It shows that the ANOR has the best performance when p is larger than 0.1. We also have obtained similar results with other values of u.

5.4 Experimental Results

Experiments have been performed on a SUN SPARC station with a local Conner CP30200 disk. We have ex-

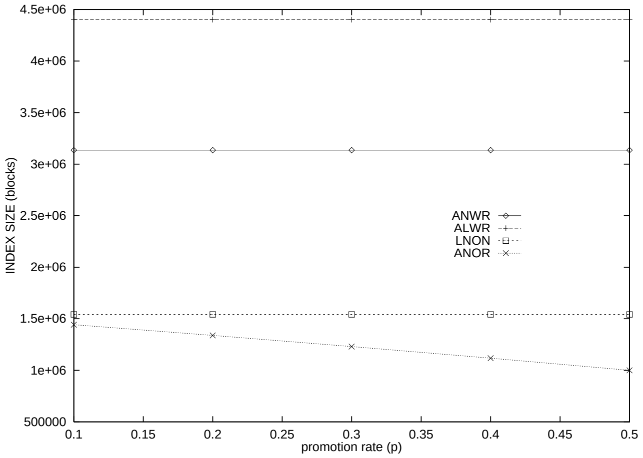

Figure 8: Index Space Requirement

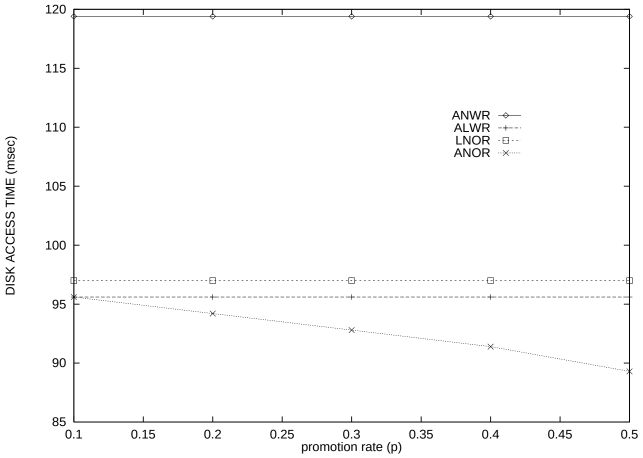

Figure 9: Average Disk Access Time

perimented in a single user environment after building a partial inverted index in a 100 Mbyte disk space. We have assumed the database contains 100,000 documents with 50,000 keywords and u is 0.6. Figure 10 shows the results of the experiment obtained by accessing the index for a given keyword 100 times each and averaging the access times. The results are similar to the analytical results. The reason that the experimental results are somewhat faster is that we have experimented in a single user environment with a relatively small disk space and we have not considered other aspects such as buffer management in the analysis.

6 Conclusions and Future Work

It is important to provide powerful structure query facilities in a structured document management system. To support structure query processing and provide fast element access, we have proposed new inverted index and signature file schemes which reduce the index

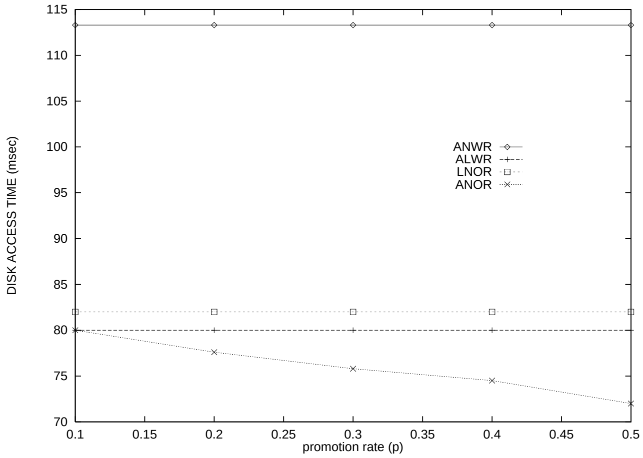

Figure 10: Experimental Disk Access Time

space considerably compared to the previous schemes. Our scheme exploits the hierarchical document structure and and uses the fact that index terms are inherited between hierarchically related elements. We have evaluated the storage requirements and disk access times of the inverted index schemes. Among the inverted indexing schemes, the ANOR (inverted index without replication) has shown the best performance in the storage requirement. It also has the least disk access time when the elements of a document have some common keywords. It means that the ANOR can be the best choice since the difference of the storage requirements between the ANOR and other schemes is great. By using this index structure, we can access document elements very fast with much less index space.

Recently, much attention has been focused on video databases [22] [26]. Video documents also have hierarchical structures as with text documents. By exploiting this structure using structure queries, users can obtain greater benefits than by using only content queries. Our indexing schemes can also be applied to structured video document management.

In this paper, we have presented signature file schemes which can be used for structured documents. We are currently evaluating the storage requirements and disk access times of them.

References

[1] T. Arnold-Moore, M. Fuller, B. Lowe, J. Thom, and R. Wilkinson, "The ELF Data Model and SGQL Query Language for Structured Document Databases," CITRI TR 94-13, Collaborative Information Technology Research Institute, Melbourne, Australia, 1994.

[2] P. B. Berra, Y. K. Lee, and K. Yoon, "Multimedia Database Design for Heterogeneous Distributed Database Systems: 2nd Interim Report," The New York State Center for Advanced Technology in Computer Applications and Software Engineering, Syracuse University, June 1995.

[3] G. E. Blake, M. P. Consens, P. Kilpelainen, P. -A. Larson, T. Snider, and F. W. Tompa, "Text/Relational Database Management Systems: Harmonizing SQL and SGML," *Proceedings of the International Conference on Applications of Databases,* pp. 267-280, Vadstena, Sweden, June 1994.

[4] D. M. Choy, F. Barbic, R. H. Gueting, D. Ruland, and R. Zicari, "Document Management and Handling," *Proceedings of the IEEE '87 Office Automation Symposium,* pp. 241-246, 1987.

[5] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl, "From Structured Documents to Novel Query Facilities," *Proceedings of the 1994 ACM SIGMOD Conference,* pp. 313-324, Minneapolis, Minnesota, May 1994.

[6] V. Christophides and A. Rizk, "Querying Structured Documents with Hypertext Links using OODBMS," *Proceedings of the European Conference on Hypermedia Technology (ECHT '94),* pp. 186-197, Edinburgh, UK, September 1994.

[7] C. Clifton and H. Garcia-Molina, "Indexing a Hypertext Database," *Proceedings of the 16th International Conference on Very Large Data Bases,* pp. 36-49, Brisbane, Australia, 1990.

[8] Conner Periperials, Inc., "CP-30200 Specification Summary," 1995.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms,* The MIT Press, 1990.

[10] C. Faloutsos, "Access Methods for Text," *ACM Computing Surveys,* vol. 17, no. 1, pp. 49-74, March 1985.

[11] C. Faloutsos, "Signature Files," In W. B. Frankes and R. Baeza-Yates, Ed., *Information Retrieval: Data Structures and Algorithms,* pp. 44-65, Prentice Hall, 1992.

[12] W. B. Frankes and R. Baeza-Yates, Ed., *Information Retrieval: Data Structures and Algorithms,* Prentice Hall, 1992.

[13] C. F. Goldfarb, *The SGML Handbook,* Oxford University Press, Oxford, UK, 1990.

[14] D. Harman, E. Fox, R. Baeza-Yates, and W. Lee, "Inverted Files," In W. B. Frankes and R. Baeza-Yates, Ed., *Information Retrieval: Data Structures and Algorithms,* pp. 28-43, Prentice Hall, 1992.

[15] *ISO 8879. Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML),* International Organization for Standardization, 1986.

[16] W. E. Kimber, "HyTime and SGML: Understanding the HyTime HYQ Query Language," Available via anonymous ftp at ftp.ifi.uio.no /pub/SGML/HyTime, August 1993.

[17] Y. K. Lee, S. -J. Yoo, K. Yoon, and P. B. Berra, "Structured Document Management and Handling," Technical Report 9506, CASE Center, Syracuse University, June 1995.

[18] Y. K. Lee, S. -J. Yoo, K. Yoon, and P. B. Berra, "Querying Structured Hyperdocuments," *Proceedings of the 29th Hawaii International Conference on System Sciences,* Maui, Hawaii, January 1996.

[19] Z. Lin and C. Faloutsos, pp. 28-43, "Frame-Sliced Signature Files," *IEEE Transactions on Knowledge and Data Engineering,* vol. 4, no. 3, pp. 281-289, June 1992.

[20] I. A. Macleod, "Storage and Retrieval of Structured Documents," *Information Processing and Management,* vol. 26, no. 2, pp. 197-208, 1990.

[21] I. A. Macleod, "A Query Language for Retrieving Information from Hierarchic Text Structures," *The Computer Journal,* vol. 34, no. 3, pp. 254-264, 1991.

[22] E. Oomoto and K. Tanaka, "OVID: Design and Implementation of a Video-Object Database System," *IEEE Transactions on Knowledge and Data Engineering,* vol. 5, no. 4, pp. 629-643, August 1993.

[23] R. Sacks-Davis, T. Arnold-Moore, and J. Zobel, "Database Systems for Structured Documents," *Proceedings of the International Symposium on Advanced Database Technologies and Their Integration (ADTI '94),* Nara, Japan, October 1994.

[24] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval,* McGraw-Hill, 1983.

[25] G. Salton, *Automatic Text Processing,* Addison-Wesley, 1989.

[26] R. Weiss, A. Duba, and D. K. Gifford, "Composition and Search with a Video Algebra," *IEEE Multimedia,* pp. 12-25, Spring 1995.