

# Analysing Performance Characteristics of Dynamic Signature File Methods

Jae-Soo Yoo, Kil-Seong Choi, and Myoung-Ho Kim

## Abstract

With rapid increase of information requirements from various application areas, there has been much research on dynamic information storage structures that effectively support insertions, deletions and updates. In this paper we evaluate the performance of the existing dynamic signature file methods such as the *S-tree*, *Quick Filter* and *HS* file and provide guidelines for the most effective usage to a given operational environment. We derive analytic performance evaluation models of the storage structures based on retrieval time, storage overhead and insertion time. We also perform extensive experiments with various data distributions such as uniform, normal and exponential distributions. The relationships among various performance parameters are thoroughly investigated. We show through performance comparison based on analytic models and experiments that regardless of data distribution, the *HS* file significantly improves performance in both the retrieval time and the storage overhead over *S-tree* and *Quick Filter*.

## I. Introduction

Information retrieval and management have long been major fields of computing. This is evident from the rapid development and widespread use of database management systems, which are well suited for a variety of business applications. These applications typically deal with formatted data. However, there are many recent applications in which a large amount of data are unformatted, such as office information systems, geographical information systems, library systems, CAD/CAM systems and multimedia database systems[3, 4]. For example, a message in office information systems consists of a header and a body[2]. The header contains formatted data representing the important characteristics of the messages, e.g., sender, date, destination, etc. The body is composed of unformatted data such as series of words, graphics and images. An approach widely advocated for efficient retrieval of unformatted data such as text and image data is to use the signature file method, which has been shown to be effective for textual data processing[9].

The signature file is an abstraction of documents, which has been extensively studied as a storage structure for unformatted data such as texts or documents[9]. Since the size of the signature file is much smaller than that of a data file, the signature

file can effectively work as a filter that immediately discards most non-qualifying documents for a given query. Many studies on the storage structure of the signature file have been made in the past, but they are mainly used for static environments[11, 12, 1, 14]. Though there are certain applications having archival nature, i.e., insertions are less frequent and updates/deletions are seldom necessary, many applications in practice require a dynamic information storage structure[16].

There are a few signature file techniques for dynamic environments. The *S-tree* proposed in[5] groups similar document signatures in its terminal nodes, and then builds a B-tree-like index structure on top of them. Since, however, the filtering capability of *S-tree* heavily depends on the query signature weight, which is the number of bits set to '1' in the query signature, its performance degradation is quite significant for light query signature weights[16]. It also has much space overhead. The *Quick Filter* proposed in[16] uses partitioning principles based on linear hashing, which tends to cluster the signatures having the same suffixes (or prefixes) in the same page. However, it has the same problem of serious performance degradation for light query signature weights as that in the *S-tree*. The hierarchical signature(HS) file proposed in[15] is a height balanced multiway tree that is a hierarchy of nodes containing signatures. It uses a frame sliced approach[10] to leaf node construction to improve a filtering effect of the signature file.

In this paper we evaluate the performance of those dynamic signature file methods and address a guideline to choose the most efficient information access scheme for a variety of environments.

Manuscript received October 19, 1996; accepted June 16, 1997.

J. S. Yoo and K. S. Choi are with Dept. of Computer and Communication, Chungbuk National University, Chungbuk, Korea.

M. H. Kim is with Dept. of Computer Science, Korea Advanced Institute of Science Technology, Taejeon, Korea.

We first present an overview of the dynamic signature file methods and develop their analytic performance cost models. We also perform experiments for the *HS file*, *S-tree* and *Quick Filter*. The experiments are performed with various data distributions such as uniform, normal and exponential distributions. The 10,000 and 100,000 documents with various types of parameters and queries are used. We compare their performance based on those cost models and experiments. Finally, we provide guidelines for the most effective usage to a given operational environment.

The remainder of this paper is organized as follows. In section 2, we describe an overview of the signature file and dynamic signature file methods. In section 3, we develop analytic performance models for the dynamic signature file methods and compare their performances through those models. Section 4 performs experiments and shows that the experiments agree with analytic models. In section 5, we summarize the signature file implementation techniques with emphasis on their convenience for the multimedia applications processed under different conditions. Finally, conclusions are described in section 6.

## II. Signature Files

The main idea of the signature file access method is to derive properties of documents, called signatures, and then store sequentially them in a separate signature file. When processing a query, the signature file is scanned and most of nonqualifying documents are discarded.

Although many extraction methods for creating the signature from a document have been proposed, most of the signature file methods typically use superimposed coding. For example, suppose that a document is represented by three words, i.e. "Database", "Document" and "Retrieval". Figure 1 illustrates the construction of a document signature using superimposed coding when a signature length is twelve, and the number of bits that are set to '1' in a word signature is two. The method can deal with queries on parts of words, but it does not preserve the sequencing of information. The number of document descriptors is expected to be constant for different documents or to have only small deviation from constant value. Otherwise, document descriptors must be divided into logical blocks, and this requires more book-keeping for the signature comparison.

Document D = (Database, Document, Retrieval)

keywords	Word Signature
Database	0110 0000 0000
Document	0000 1000 0001
Retrieval	0001 0001 0000
Document Signature	0111 1001 0001

Fig. 1. Document Signature construction using Superimposed Coding.

A retrieval procedure using a signature file is as follows. Words in a query are first hashed to form a query signature by the same way used for the document signature. If every bit that is set in the query signature is also set in the document signature, then the document signature becomes a potential match. Superimposed coding can result in a false match because a document signature can qualify a query signature, though the document itself does not satisfy the query. A signature extraction method that reduces the number of false matches is also important, but is not the main concern of this paper.

A few works have been made to enhance the basic form of the signature file for the dynamic environments. They include the *S-tree*, *Quick Filter* and *HS file*, which are described below.

### 1. S-tree

The S-tree is a dynamic tree organization of signatures[5]. An S-tree groups similar document signatures in its terminal nodes and then builds a B-tree-like index structure on top of them. Even though the deletion requires some extra effort, the S-tree works well and always remains balanced. The filtering capability of an S-tree heavily depends on the query signature weight, which is the number of bits set to '1' in the query signature. Thus, while the S-tree achieves very good performance for the heavy query weights, its performance degradation is quite significant for light query signature weights[16]. It also has much space overhead.

### 2. Quick Filter

The *Quick Filter* uses partitioning principles based on linear hashing for organizing and searching for the dynamic data file[16]. This method tends to cluster the document signatures having the same suffixes (or prefixes) in the same page.

The *Quick Filter* partitions a set of document signatures into  $n$  pages, where  $2^{h-1} < n \leq 2^h$  for some integer  $h$ . The document signatures with the same  $h$ -bit suffix are stored into the same primary page or its overflow page. When a document signature  $S$  is stored in the *Quick Filter*, the associated hash split function  $g(S, h, n)$  is as follows:

$$g(S, h, n) = \begin{cases} \sum_{i=0}^{h-1} b_s - 2^i, & \text{if } \sum_{i=0}^{h-1} b_s - 2^i < n \\ \sum_{i=0}^{h-2} b_s - 2^i, & \text{otherwise} \end{cases}$$

Here,  $b_i$  denotes  $i$ -th bit value of the signature and  $s$  denotes signature size in bits. The hash function  $g$  takes the  $h$  or  $(h-1)$  bit suffix of  $S$  and interprets it as a non-negative integer. The value of  $g$  is always smaller than  $n$  and is used to determine an addressable page where the signature  $S$  is to be stored. The initial condition is  $h = 0$ ,  $n = 1$  and we define  $g(S, 0, 1) = 0$ .

Since the *Quick Filter* is constructed based on linear hashing, the important characteristic of this organization is that all signa-

tures in a page have the same suffix (or prefix) corresponding to the level of hashing  $h$ . The *Quick Filter* has the advantage that the more the number of bits set to '1' in the query signature is, the less the number of blocks accessed is. However, it has the same problem of serious performance degradation for light query signature weights as that in the S-tree.

### 3. HS File

The HS file is a height balanced multiway tree that is a hierarchy of nodes containing signatures[15]. The HS file has two types of nodes, namely a leaf node and a non-leaf node. It uses the frame sliced approach[10] to leaf node construction to improve a filtering effect of the signature file.

The HS file of type  $(b_1, b_2, f)$  has the following properties, where  $b_1$  and  $b_2$  are the blocking factors of a leaf node and a non-leaf node respectively, and  $f$  is the number of frames in the leaf node:

1. Each path from the root to any leaf node has the same length.
2. A leaf node consists of  $f$  blocks and one pointer block. Each leaf node has at most  $b_1$  document signatures that are stored into  $f$  frames, and  $b_1$  pointers to the corresponding documents.
3. A non-leaf node is composed of only one block. Each non-leaf node has at most  $b_2$  children and signature.
4. The signatures in the non-leaf nodes are constructed by superimposing the signatures contained in their child node.

## III. Analytic Cost Model

In this section we develop analytic performance models of our HS file[15], the *quick Filter*[16] and the S-tree[5]. The existing dynamic signature file methods have not constructed the analytic cost models to compare their performance in various environments. As a result, we is to derive the analytic performance models, which strongly substantiate our performance results and enable the analytic cost models to be used for various types of environments that are difficult to be constructed for actual performance experiments. The analytic performance models are derived based on the characteristics of the methods presented in the [15, 16, 5]. Table 1 shows the notations and descriptions of the input and design parameters. Now we will examine the performance measures for given input and design parameters. The measures we are interested in are listed below:

- $R_{HS}, R_{QF}, R_{ST}$  : Number of disk block accesses on retrieval for the HS file, the Quick Filter and the S-tree, respectively.
- $O_{HS}, O_{QF}, O_{ST}$  : Additional disk space(pages)for the HS file, the *Quick Filter* and the S-tree, respectively.

- $I_{HS}, I_{QF}, I_{ST}$  : Number of disk block accesses on the insertion of one document for the HS file, the *Quick Filter* and the S-tree, respectively.

**Table 1.** Input / Design Parameters.

Parameters	Description
N	Total number of documents
P	Size of disk page(block) in bits
D	Number of words per one document
Q	Query signature
s	Document signature size in bits
k	Average document signature weight
f	Number of frames in the HS file
m	Number of bits set to '1' by one word
n	Number of addressable pages of the signature file
w	Number of words in the query
c	Number of distinct frames selected by each word when constructing a document signature
t	Size of pointer in bytes
h	Height of the HS file
sh	Height of the S-tree
l	Hashing level of the <i>Quick Filter</i>
$b_1$	Blocking factor of a leaf node in the HS file
$b_2$	Blocking factor of a non-leaf node in the HS file, a node in the S-tree and a page in the <i>Quick Filter</i>
lf	Average page(node) load factor of HS file, <i>QuickFilter</i> , S-tree
$W(Q)$	Query weight, i.e., the number of ones in the query signature
$\beta(i)$	Expected utilization of the level $i$ in the HS file and S-tree
$\lambda(i)$	Cardinality in the level $i$
$p(x,i)$	Probability that $x$ prespecified positions contain ones in the level $i$

### 1. Retrieval Time

To estimate the retrieval performance we assume that a query with  $w$  words should be processed and the document signatures follow uniform distribution. For the analysis, we make use of the following measures for searching the signature file as well as retrieving qualifying documents.

- $R_{non-leaf}$  Number of accessed disk blocks when accessing non-leaf nodes for a given query in the HS file. This is computed as  $\sum_{d=1}^{h-2} (\prod_{i=1}^d \beta(i) p(\omega * m, i)) + 1$ .  $\prod_{i=1}^d \beta(i) p(\omega * m, i)$  is the average number of matched signatures in the height  $d$ . The  $p(\omega * m, i)$  is also computed as  $(1 - (1 - \frac{k}{S})^{i(d)})^{\omega * m}$  and the  $\lambda(i)$  and  $\beta(i)$  are calculated as  $\frac{N}{\prod_{j=1}^i \beta(j)}$  and  $\frac{N}{(b_2 * lf)^{i-1}}$  for  $i = 1, 2, \dots, h-2$ , respectively.
- $R_{leaf}$  Number of accessed disk blocks when accessing leaf nodes for a given query in the HS file. This is computed as  $\prod_{i=1}^{h-1} (\beta(i) p(\omega * m, i)) (f * (1 - (1 - \frac{1}{f})^\omega) * c)$ . Here,  $f * (1 - (1 - \frac{1}{f})^\omega)$  is the average number of distinct frames selected.
- $R_c$  Expected number of bits set in the 1-bit suffix of the query signature.

This is computed as  $\sum_{j=1}^{\min(L, W(Q))} (j * P(j))$ .

Here,  $P(j)$  is the probability that  $j$  bits are set in the 1-bit suffix of the query signature and can be written as

$$\frac{\binom{s-1}{W(Q)-j} \binom{1}{j}}{\binom{s}{W(Q)}}$$

$R_{\text{node}}$  Number of accessed disk blocks when accessing nodes except root node for a given query in the S-tree.

This is computed as  $\sum_{d=1}^{sh-1} \left( \prod_{i=1}^d \beta(i) p(\omega * m, i) \right)$ .

$\prod_{i=1}^d (\beta(i) p(\omega * m, i))$  is the number of matched signatures at depth  $d$ .

According to these measures, we can calculate the retrieval times of the HS file, *Quick Filter* and S-tree as follows:

$$R_{\text{HS}} = R_{\text{non-leaf}} + R_{\text{leaf}}$$

$$R_{\text{QF}} = \frac{n}{2^{Re}}$$

$$R_{\text{ST}} = R_{\text{node}} + 1$$

## 2. Storage Overhead

In addition, we make use of the following measures for estimation the storage overhead needed to maintain the signature file.

$O_{\text{leaf}}$  Additional storage space for maintaining the leaf nodes in the HS file.

This is computed as  $\left[ \frac{N}{b_1 * lf} (1 + f) \right]$ .

$O_{\text{Snon-leaf}}$  Additional storage space for maintaining the superior non-leaf nodes of leaf nodes in the HS file. This is computed as  $\left[ \frac{B_o}{b_2 * lf} \right]$ . here, the  $B_o$  is  $\left[ \frac{N}{b_1 * lf} \right]$ .

$O_{\text{non-leaf}}$  Additional storage space for maintaining the non-leaf nodes except superior non-leaf nodes in the HS file.

This is computed as  $\sum_{i=1}^{h-2} \left[ \frac{B_i}{b_2 * lf} \right]$ .

Here, the  $B_i$  is  $\left[ \frac{B_{i-1}}{b_2 * lf} \right]$  for each  $i = 1, 2, \dots, h-2$ .

$O_{\text{Sleaf}}$  Additional storage space for maintaining the leaf nodes in the S-tree.

This is computed as  $\left[ \frac{N}{b_2 * lf} \right]$ .

$O_{\text{Snon-leaf}}$  Additional storage space for maintaining the non-leaf nodes in the S-tree.

This is computed as  $\sum_{i=0}^{sh-2} \left[ \frac{S_i}{b_2 * lf} \right]$ .

Here, the  $S_o$  is  $\left[ \frac{N}{b_2 * lf} \right]$  and the  $S_i$  is  $\left[ \frac{S_{i-1}}{b_2 * lf} \right]$  for each  $i = 1, 2, \dots, sh-2$ .

According to these measures, we can calculate the storage overheads of the HS file, *Quick Filter* and S-tree as follows :

$$O_{\text{HS}} = O_{\text{leaf}} + O_{\text{Snon-leaf}} + O_{\text{non-leaf}}$$

$$O_{\text{QF}} = \frac{N}{b_2 * lf}$$

$$O_{\text{ST}} = O_{\text{Sleaf}} + O_{\text{Snon-leaf}}$$

## 3. Insertion Time

We can calculate the insertion times of the HS file, *Quick Filter* and S-tree as follows :

$I_{\text{HS}} = 2(h+f) - 1 + \alpha_1$ , where  $h$  is estimated as  $\lceil \log_{\frac{b_1+b_2}{2}} N \rceil + 1$ , and  $\alpha_1$  is the average number of accessed blocks to reorganize the file when an overflow occurs. The  $\alpha_1$  is estimated as  $\left\lceil \frac{h+f-1}{b_1} \right\rceil$ .

$I_{\text{QF}} = 2 + \alpha_2$ , where  $\alpha_2$  is the average number of overflow pages accessed when inserting one document and is estimated as  $\left\lceil \frac{1}{b_2} \right\rceil$ .

$I_{\text{ST}} = 2sh + \alpha_3$ , where  $sh$  is at most  $\lceil \log_k N \rceil - 1$ , and  $\alpha_3$  is the average number of accessed nodes to reorganize the file when an overflow occurs. The  $\alpha_3$  is estimated as  $\left\lceil \frac{sh}{b_2} \right\rceil$ .

## 4. Performance Comparison

We compare the performance of the HS file with the S-tree and the *Quick Filter* using the developed analytic cost models. The values of each input and design parameter are presented in Table 2 and are based on [14].

**Table 2.** The values for parameters.

methods parameters	HS File	Quick Filter	S-tree
N	10,000	10,000	10,000
	100,000	100,000	100,000
P	1 Kbytes	1 Kbytes	1 Kbytes
	2 Kbytes	2 Kbytes	2 Kbytes
D	20	20	20
f	2, 4, 8, 16	NA	NA
m	16	16	16
w	2 ~ 10	2 ~ 10	2 ~ 10
t	4	4	4
s	512 bits	512 bits	512 bits
k	240 bits	240 bits	240 bits
b1	256	NA	NA
b2	16	16	16
W(Q)	32-160	32-160	32-160
lf	0.68	0.59	0.53

We investigate the retrieval performance and storage requirement of dynamic signature file methods in the following three cases.

**CASE 1** A data file consists of 10,000 documents and the page size is 1K bytes.

**CASE 2** A data file consists of 10,000 documents and the

page size is 2K bytes.

**CASE 3** A data file consists of 100,000 documents and the page size is 1K bytes.

Figure 2 (a), (b), (c) show the retrieval performances of the dynamic signature file methods on the CASE 1, CASE 2, CASE 3, respectively. The theoretical results show that the HS file achieves about 240% and 300% performance gains on retrieval over the *Quick Filter* and the S-tree on the average.

Table 3 illustrates the storage overheads of the dynamic signature file methods on the three cases. The storage overheads of the HS file, *Quick Filter* and S-tree are about 10.1%, 11.5% and 18.7% on the average. The HS file uses the least storage space, while S-tree is worst.

**Table 3.** Storage overhead of dynamic signature file methods.

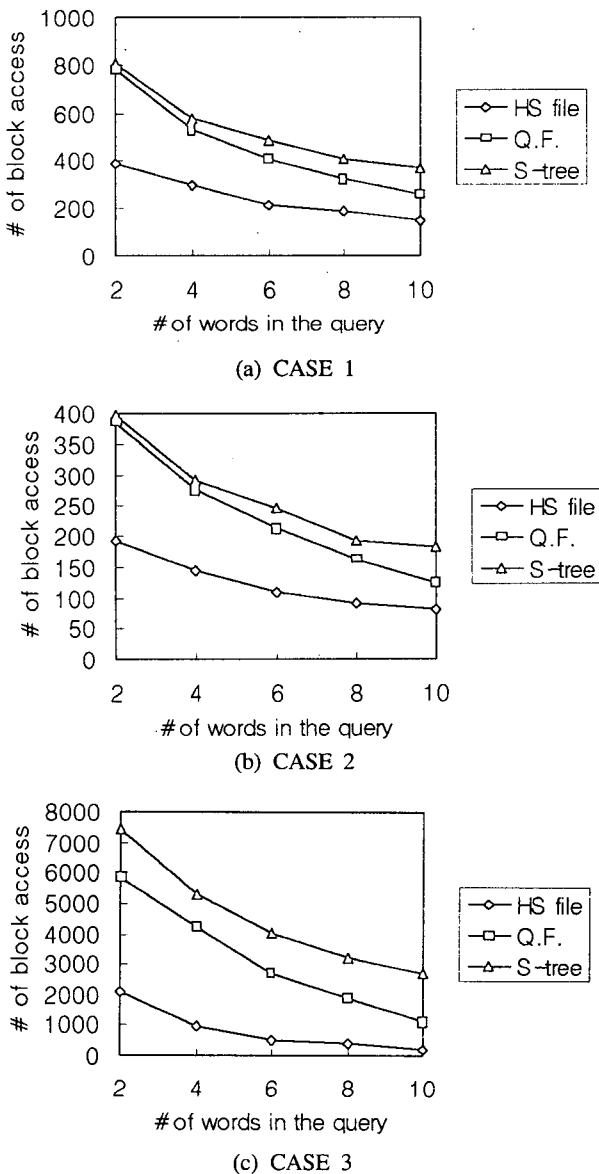
Method \ Experiment	CASE 1	CASE 2	CASE 3
S-tree	1,845	1,870	19,240
Quick Filter	1,151	1,163	11,430
HS File	983	1,031	10,215

Since we assumed that document signatures follow uniform distribution, probabilities that overflow occurs in the HS file, *Quick Filter* and S-tree are  $\frac{1}{b_1}$ ,  $\frac{1}{b_2}$  and  $\frac{1}{b_2}$ , respectively. Therefore, the  $a_1$ ,  $a_2$  and  $a_3$  are directly proportional to  $\frac{1}{b_1}$ ,  $\frac{1}{b_2}$  and  $\frac{1}{b_2}$ , respectively. When the database consists of 100,000 documents and the number of frames is 8, the average number of blocks accessed in order to insert one document in the HS file, *Quick Filter* and S-tree is about 11, 2 and 8, respectively. That is, the *Quick Filter* achieves the best insertion performance, while the HS file is the worst. However, the difference of insertion performance between the HS file and the *Quick Filter* is very small over that of retrieval performance between them. As a result, since in the information retrieval applications, retrievals occur much more frequently than insertions, we can see that the HS file is significantly better than other dynamic signature file methods, in terms of whole system performance.

However, it is very difficult to design the analytic cost models correctly estimating such portions in terms of the retrieval performance because the number of false matches and the number of block accesses to process the given multikey query may vary on the real environment. As we also can not correctly estimate the average page load factor of each method and the heights of HS file and S-tree, the analytic models on storage overhead and insertion performance are not considered to be correct. Therefore, to verify the analytic cost models, we actually implement them and run the experiments in section 4.

### IV. Experiments

In this section, to compare the performance of dynamic signature file methods and investigate the characteristics of the HS file, we actually implement them and perform extensive experiments with various data distributions: uniform, normal and exponential. Three basic distributions were used over the range of  $[-2^{31}, 2^{31} - 1]$ : 1) a uniform distribution, 2) a normal distribution  $N(0, \sigma)$ , where  $\sigma = 1/3 \times 2^{31}$  and 3) an exponential distribution  $1/\theta \times e^{-(x+2^{31})/\theta}$ , where  $\theta = 1/4 \times 2^{32}$ . The experiments are also performed for various sizes of databases and various performance parameters as shown in Table 2. We use 100 sample queries to evaluate the characteristics of the HS file and the performances of the dynamic signature file methods. We discuss the performance comparison of the dynamic signature file methods with



**Fig. 2.** Analytic retrieval performance.

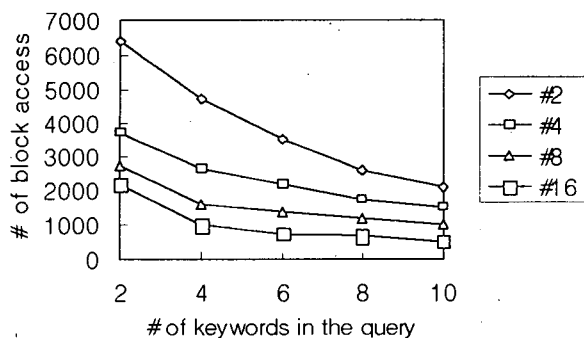


Fig. 3. Retrieval performance of the HS file according to the number of frames.

various numbers of frames and with various types of queries. For convenience, we discuss the performance comparison of the dynamic signature file methods when 100,000 documents with various numbers of frames and with various types of queries are used and the page size is 1 Kbytes. This is because experimental results of the remaining cases used in the analytic cost models are very similar to those of this case.

First, we investigate how much frame-based document signature construction affects the retrieval performance. When the number of frames chosen for word signature is sixteen, we found that the retrieval performance of frame-based document signature construction is about 20% better than the conventional document signature extraction method. Second, we investigate the retrieval performance of the HS file according to the number of frames in the leaf node when the number of documents is 100,000. We can see through Figure 3 that the larger the number of frames is, the better the retrieval performance is. When the size of a document signature is 512 bits and the size of a pointer is 4 bytes, the HS file has at most sixteen frames in the leaf node. The reason is that when the HS file has only one pointer block in the leaf node, the size of a frame signature must not be less than that of a pointer. As a result, the retrieval performance of the HS file using sixteen frames is about 3.3 times better than that of the HS file using two frames.

According as the number of frames in the leaf nodes is increased, the storage space that the HS file uses is shown in Figure 4. We can see through the figure that when constructing HS file, the more the number of frames in the leaf nodes, the lower the storage space that it occupies. This is because according as the number of frames in the leaf node is increased, the occurrence rate of overflow in the HS file is decreased, and thus the number of its internal nodes and the height of HS files is reduced.

Figure 5 shows experimental results on retrieval of each dynamic signature file method when data follows uniform, normal and exponential distributions. The number of frames in the leaf node of the HS file is sixteen. In the figure, symbols U, N and E represent uniform, normal and exponential distributions, respectively. Figure 5(a) shows that the HS file achieves greatly

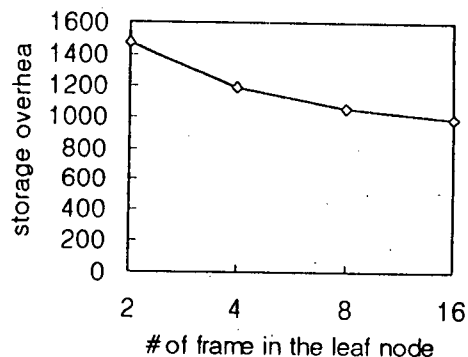


Fig. 4. Storage overhead of the HS file according to the number of frames.

similar retrieval performance independently of data distributions. However, we can see through Figure 5(b) and (c) that the *Quick Filter* and S-tree are somewhat dependent on the data distributions. The retrieval performance of each method, when data follows uniform distribution, is better than that of each method when data follows skewed distributions such as normal and exponential distributions.

Figure 6 illustrates the retrieval performance of dynamic signature file methods when data follows uniform distribution and the number of frames in the leaf node of the HS file is 16. We can see through Figure 6 that the HS file is much more efficient than the other dynamic signature file methods, independent of the number of words in the query. From the experimental results, we showed that the HS file achieved about 180~360% and about 200~400% performance gains on retrieval over *Quick Filter* and S-tree, on the average. This is because the HS file uses frame-based signature extraction method and a unique retrieval process[15].

When the number of documents is 100,000 and the number of frames is 16, the storage overheads of the HS file, *Quick Filter* and S-tree are about 9.8%, 10.3% and 21.2%, respectively. As a result, the storage overhead of the HS file is much less than that of S-tree, while it is similar to that of the *Quick Filter*.

When the number of frames in the HS file is 16, the average number of blocks accessed in order to insert one document in the HS file, *Quick Filter* and S-tree is about 20, 4 and 10 on the average, respectively. We found that the *Quick Filter* achieves the best insertion performance, while the HS file achieves the worst. The reason is that the *Quick Filter* is constructed based on the linear hashing and the HS file uses the frame-sliced approach to the leaf node. In the information retrieval applications where retrievals occur much more frequently than insertions, however, such difference of insertion performance among dynamic signature file methods can be ignored.

In order to verify the correctness of the analytic model, the error rates on retrieval time, storage overhead and insertion time are computed as follows, where E and T indicate a theoretical

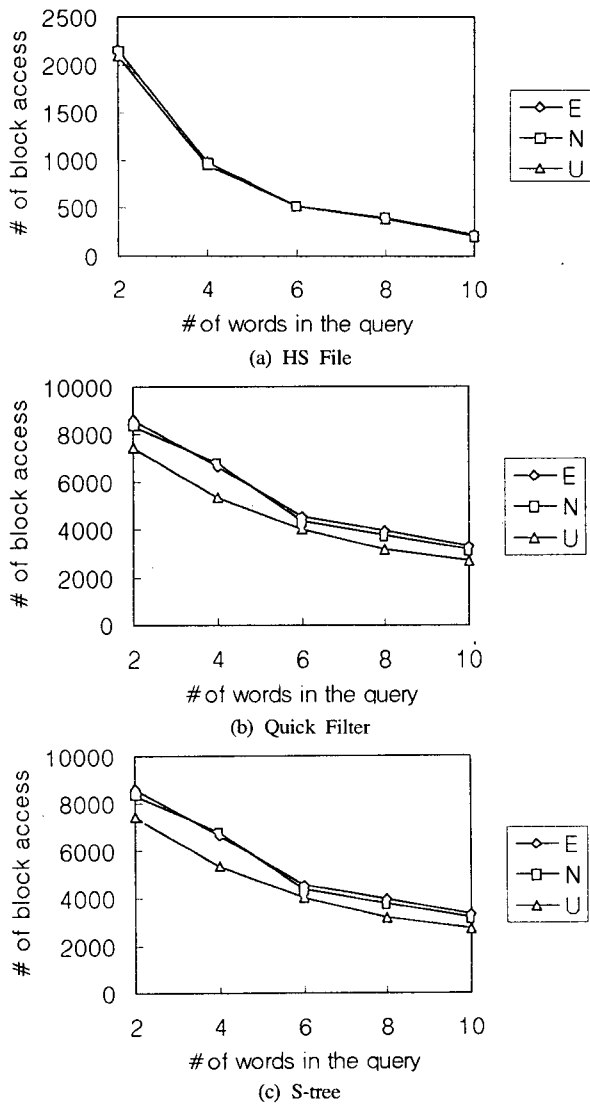


Fig. 5. Retrieval performance of dynamic signature file methods according to data distributions.

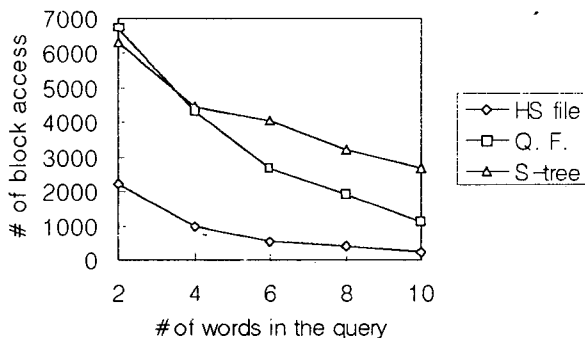


Fig. 6. Experimental retrieval performance of dynamic signature file methods.

100,000 documents respectively.

$$\text{Error Rate} = [\text{Max}(T,E) - \text{Min}(T,E)] / \text{Max}(T,E)$$

The error rates on storage overhead of HS file and Quick Filter are very small. However the difference between analytic model and experimental result of S-tree(2.5%) is more than that of HS file(0.3%) and Quick Filter(1.2%). The error rates on insertion are very small. As a result, the conclusion from the experiment is that the analytic and experimental results agree well.

## V. Discussion

In this section, we summarize the results in a guideline that provides information for selecting a signature file structure for dynamic applications. To do this, we first present criteria that should allow users to express specific characteristics of their applications. We extend criteria for storage structures by Tiberio [13]. Secondly, we provide guidelines for the most effective usage of a signature file to a given operational environment.

### 1. Application Characteristics and Guideline Criteria

Many dynamic applications such as office information systems, geographical information systems, design applications(CAD/CAM), advanced information retrieval systems and so on require the processing of the documents with multimedia data(e.g. text, image, voice, etc.). The multimedia data documents are regarded as the data representation of complex entities. For example, a document may consist of attributes(e.g., author, title, publisher, date), text(e.g., message, report, program source), image(e.g., digitized photograph, pie chart) or voice(e.g., voice annotation).

Multimedia documents are typically large and can be considered as a consecutive area of bytes with variable length from the data storage point of view. They are identified by a unique document identifier. Currently, we consider only documents with two types of data, text and image among multimedia data types. The text query expresses the fact that, in a typical query on text data, only a small set of words is used as a search pattern. Since the total number of words in the natural language text is high, the selectivity of individual words is also high. Text queries with only one word are rare and those with many words that result in a query weight higher than 50% of the maximum query signature weight, cannot be considered at all.

The specification of an image data query reflects the fact that the number of documents that can be recognized by image analyzers is much smaller than the number of distinct words in a natural language text. Obviously, selectivity of the image data descriptors is much lower and the specification of a reasonable image query leads to query signatures with higher weights than for text queries.

In order to get a storage structure that is best suitable for

result and an experimental result, respectively. The error rates on retrieval time are 0.5~7% and 0.5~15% in case of 10,000 and

specific environment, we may classify the methods according to their suitability degree for each criteria. To do this, we have chosen the following levels: A = excellent, B = good, C = fair, D = requires a little effort, E = requires much effort, F = cannot use at all.

Now the criteria for signature file structures are presented based on Tiberio's criteria[13]. The dynamic storage structures may have the following four criteria that influence their performance: (1) query signature weight, (2) the size of a data file, (3) storage overhead, (4) support of applications.

As we have seen in previous section, query signature weight is also related to the type of multimedia data. Low-weight queries are typical for text data of natural language, while heavy-weight queries are typical for image data such as technical drawing, photograph and so on. Since the number of distinct words in the text data is very large, the selectivity of one word is very high and usually few words are enough to form a selective query.

On the other hand, the number of distinct descriptors in image documents is low. The reason is that good general image analyzers have not been yet developed. Most of image analyzers depend on applications and are capable of recognizing only specialized sets of components classified as a descriptor. In order to find an image document, we should give a query with many descriptors because, in general, the selectivity of those descriptors is very low. As a result, the query signature weights for image documents is very heavy. It should be noted that the query signature weight does not depend much on the number of descriptors in the documents. The total of distinct descriptors is much more important. We consider the following three criteria regarding query signature weight: SS1 = low, SS2 = medium, SS3 = high.

Since the performance of the storage structures depends on the size of a document file in the signature files, we consider the following three criteria: SS4 = small, SS5 = medium, SS6 = large. The last criterion is SS7 = space requirements. This will make it possible to consider the storage structures according to their convenience in terms of space overhead.

## 2. Selection Guidelines for Dynamic Applications

Until now, we have investigated and analyzed the performance and characteristics of dynamic signature file organizations. Based on these researches, we address a guideline to choose the most appropriate signature file structure for a variety of applications.

Since all the information is very general and even vague, a numeric form is less suitable. therefore, we evaluate various access methods using grades described in the previous subsection. The main contribution of this work is the performance comparison framework that can be used to select a good implementation configuration for a specific problem. To illustrate the process of decision-making, we consider two different files of multimedia data. One is office text data files and the other is a constantly

growing image file with few updates.

Table 4 shows the degrees of suitability of dynamic signature file methods according to the criteria of storage structures. The SSF is a simple file of fixed-length signatures[16]. It is called single-level signature file or signature file method with bit-string representation. Table 4 is based on the results of performance analysis of dynamic signature file methods based on analytic cost models and experiments. For the application such as the office text file, HS file is much better than any other dynamic signature file method. The reason is that a office text file is characterized by low weight query signatures. For the applications such as image files, the situation is different because of large data files and higher query signature weights that we can expect for image queries. For such conditions, we recommend that *quick filter* or HS file is used. However, even for high weight queries, HS file is more efficient than *quick filter*.

**Table 4.** Degrees of convenience of dynamic storage structures.

Dynamic Storage Structures	Criteria						
	Query weights			Size of data file			SR
	SS1	SS2	SS3	SS4	SS5	SS6	SS7
SSF	D	D	D	C	D	E	A
S-tree	F	D	B	C	D	D	E
Quick Filter	E	D	B	B	B	A	B
HS	B	A	A	C	A	A	B

## VI. Conclusions

We have evaluated the performance of dynamic signature file methods in terms of retrieval time, storage overhead and insertion time. We have first developed analytic cost models and evaluated the space-time performance of these methods in the various environments. Then, we have carried out extensive performance experiments with various data distributions such as uniform, normal and exponential distributions and a wide range of parameter values. We have found that experiments closely agree with the analytic cost models, which strongly substantiate our performance results and enable the analytic cost models to be used for various types of environments that are difficult to be constructed for actual performance experiments. Through analytic cost models and various experiments, we have shown that the HS file has improved performance significantly in both the retrieval time and the storage overhead over the methods proposed earlier.

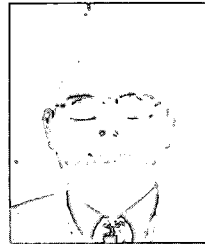
We have also summarized the dynamic signature file methods with emphasis on their suitability for dynamic applications processed under different conditions. The criteria for dynamic storage structures were presented for guidelines that can be used to select effective implementations for specific applications. Based on the criteria, we have provided a dynamic storage structure that can



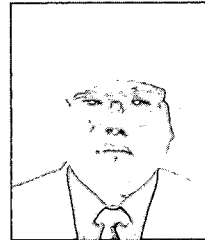
outperform others for the most effective usage to a given operational environment.

## References

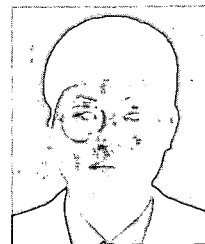
- [1] J. W. Chang, J. H. Lee and Y. J. Lee, "Multikey Access Methods Based on Term Discrimination and signature Clustering," ACM SIGIR, pp. 176-185, 1989.
- [2] S. Christodoulakis and C. Faloutsos, "Design Considerations for a Message File Server," IEEE Trans. on Software Engineering, Vol. SE-10, No. 2, pp. 201-210, Mar. 1984.
- [3] S. Christodoulakis et al., "Development of a Multimedia information system for an Office Environment," VLDB '84, pp. 261-271, 1984.
- [4] S. Christodoulakis, "Issues in the Architecture of a Document Archiver using Optical Disk Technology," ACM SIGMOD, pp. 34-50, May 1985.
- [5] U. Deppisch, "S-tree: A Dynamic Balanced Signature Index for Office Retrieval," ACM SIGIR, pp. 77-87, 1986.
- [6] J. R. Files and H. D. Huskey, "An Information Retrieval System Based on Superimposed Coding," Proc. of the Fall Joint Computer Conf., pp. 423-432, 1969.
- [7] C. Faloutsos, "Access methods for Text," Computing Surveys, Vol. 17, No. 1, pp. 49-74, 1985.
- [8] C. Faloutsos and S. Christodoulakis, "Description and Performance File Method for Office Filing," ACM Trans. on Information Systems, Vol. 5, No. 3, pp. 237-257, 1987.
- [9] C. Faloutsos, "Signature-based Text Retrieval Methods : A Survey," IEEE Computer Society Technical Committee on Data Engineering, Vol. 13, No. 1, pp. 25-32, Mar. 1990.
- [10] Z. Lin and C. Faloutsos, "Frame-Sliced Signature Files," IEEE Trans. on Knowledge and Data Engineering, Vol. 4, No. 3, pp. 281-289, Jun. 1992.
- [11] C. S. Roberts, "Partial Match Retrieval via the Method of the Superimposed Codes," Proc. IEEE 67, pp. 1624-1642, Dec. 1979.
- [12] R. Sacks-Davis and K. Ramamohanarao, "Multikey Access Methods based on Superimposed Coding Techniques," ACM Trans. on Database Systems, Vol. 12, No. 4, pp. 655-696, Dec. 1987.
- [13] P. Tiberio and P. Zezula, "Selecting Signature Files for Specific Applications," 5th Annual European Computer Conference, pp. 718-725, May 1991.
- [14] J. S. Yoo, J. W. Chang, Y-J Lee and M. H. Kim, "Performance Evaluation of Signature-Based Access Mechanisms for Efficient Information Retrieval," IEICE Trans. on Information and Systems, Vol. E76-D, No. 2, pp. 179-183, Feb. 1993.
- [15] J. S. Yoo, Y-J Lee, J. W. Chang and M. H. Kim, "The HS File : A New Dynamic Signature File Method for Efficient Information Retrieval," 5th International Conference, DEXA '94, Athens Greece, pp. 571-580, Sep. 1994.
- [16] P. Zezula, F. Rabitti and P. Tiberio, "Dynamic Partitioning of Signature Files," ACM Trans. on Information Systems, Vol. 9, No. 4, pp. 336-369, Oct. 1991.



**Jae-Soo Yoo** received the B.S. degree in computer engineering from Chonbuk National University in 1989, and the M.S. and Ph. D. degrees in computer science from the Korea Advanced Institute of Science and Technology(KAIST), in 1991 and 1995. From March 1995 to August 1996, he was a member of the faculty of the computer science at the Mokpo National University. He has been as assistant professor of computer and communication engineering at the Chungbuk National University since 1996. His research interests include database systems, multimedia database, information retrieval and distributed object computing.



**Kil-Seong Choi** received the B.S. degree in the department of computer science from Taejon Industrial University in 1988, and the M.S. degree in the department of computer science from Suwon University in 1992. Since 1996, he has been studying database systems at the department of computer and communication engineering in Chungbuk National University for Ph. D. degree. His research interests include Multimedia and Imaging Database Systems, and Information Retrieval.



**Myoung-Ho Kim** received the B.S. and M.S. degrees in computer engineering from Seoul National University, Seoul, Korea in 1982 and 1984, respectively, and the Ph. D. degree in computer science from Michigan State University, East Lansing in 1989. Since 1989 he has been with the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Taejon where he currently is an associate professor. His research interests include database systems, distributed and parallel database, data mining, multidatabase, information retrieval and real-time database.