Performance Evaluation of Dynamic Signature File Methods

Jae Soo Yoo

Dept. of Computer Science & Statistics
Mokpo National University
Muan-gun, Chonnam, 534-729, South Korea

M. H. Kim, Y. J. Lee & B. M. IM

Dept. of Computer Science
KAIST, 373-1 Kusung-dong
Yusung-gu Taejon 305-701 South Korea

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.

1 Introduction

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[3]. 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[5, 1, 7]. 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[9].

There are a few signature file techniques for dynamic environments. such as the *S-tree*[2], the *Quick Filter*[9] and the *HS File*[8]. 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. We first present 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.

2 Signature Files

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.

2.1 S-tree

The *S-tree* is a dynamic tree organization of signatures[2]. 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[9]. It also has much space overhead.

2.2 Quick Filter

The *Quick Filter* uses partitioning principles based on linear hashing for organizing and searching for the dynamic data file[9]. This method tends to cluster the document signatures having the same suffixes (or prefixes) in the same page.

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

2.3 HS File

The *HS* file is a height balanced multiway tree that is a hierarchy of nodes containing signatures[8]. The *HS* file has two types of nodes, namely a leaf node and a non-leaf node. It uses the frame sliced approach[4] 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 sons and signatures.
4. The signatures in the non-leaf nodes are constructed by superimposing the signatures contained in their son node.

3 Analytic Cost Model

In this section we develop analytic performance models of our *HS* file[8], the *Quick Filter*[9] and the *S-tree*[2]. the notations and descriptions of the input and design parameters are same with [8]. 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.

3.1 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. This is computed as $\sum_{d=1}^{h-2}(\prod_{i=1}^{d}\beta(i)p(w*m,i))+1$. $\prod_{i=1}^{d}(\beta(i)p(w*m,i))$ is the average number of matched signatures in the height $h-1$. The $p(w*m,i)$ is also computed as $(1-(1-\frac{k}{s})^{\lambda(i)})^{w*m}$ 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, ..., h-2, respectively.
- R_{leaf} : Number of accessed disk blocks when accessing leaf nodes for a given query. This is computed as $\prod_{i=1}^{h-1}(\beta(i)p(w*m,i))(f*(1-(1-\frac{1}{f})^{w})*c)$. Here, $f*(1-(1-\frac{1}{f})^{w})$ is the average number of distinct frames selected.
- R_e : Expected number of bits set in the l-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 l-bit suffix of the query signature and can be written as $\frac{\binom{s-l}{W(Q)-j}\binom{l}{j}}{\binom{s}{W(Q)}}$.
- R_{node} : Number of accessed disk blocks when accessing nodes except root node for a given query in the S-tree. This is computed as $\sum_{d=1}^{h-1}(\prod_{i=1}^{d}\beta(i)p(w*m,i))\ \prod_{i=1}^{d}(\beta(i)p(w*m,i))$ 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:

- $R_{HS} = R_{non-leaf} + R_{leaf}$
- $R_{QF} = \frac{n}{2^{R_e}}$
- $R_{ST} = R_{node} + 1$

3.2 Storage Overhead

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

- O_{leaf} : Additional storage space for maintaining the leaf nodes in the *HS* file. This is computed as $\lceil\frac{N}{b_1*lf}(1+f)\rceil$.
- $O_{snon-leaf}$: Additional storage space for maintaining the superior non-leaf nodes of leaf nodes in the *HS* file. This is computed as $\lceil\frac{B_0}{b_2*lf)}\rceil$. Here, the B_0 is $\lceil\frac{N}{b_1*lf}\rceil$.
- $O_{non-leaf}$: Additional storage space for maintaining the non-leaf nodes except superior non-leaf nodes in the *HS* file. This is computed as $\sum_{i=1}^{h-2}\lceil\frac{B_i}{b_2*lf}\rceil$. Here, the B_i is $\lceil\frac{B_{i-1}}{b_2*lf}\rceil$ for each i=1, 2, ..., h-2.

- O_{Sleaf} : Additional storage space for maintaining the leaf nodes in the $S-tree$. This is computed as $\lceil \frac{N}{b_2 * lf} \rceil$.

- $O_{Snon-leaf}$: Additional storage space for maintaining the non-leaf nodes in the $S-tree$. This is computed as $\sum_{i=0}^{sh-2} \lceil \frac{S_i}{b_2 * lf} \rceil$. Here, the S_0 is $\lceil \frac{N}{b_2 * lf} \rceil$ and the S_i is $\lceil \frac{S_{i-1}}{b_2 * lf} \rceil$ for each i=1, 2, ..., sh-2.

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

- $O_{HS} = O_{leaf} + O_{snon-leaf} + O_{non-leaf}$

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

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

3.3 Insertion Time

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

$I_{HS} = 2(h+f) - 1 + \alpha_1$, where h is estimated as $\lceil \log_{\frac{b_1+b_2}{2}} N \rceil + 1$, and α_1 is the average number of accessed blocks to reorganize the file when an overflow occurs. The α_1 is estimated as $\lfloor \frac{h+f-1}{b_1} \rfloor$.

$I_{QF} = 2 + \alpha_2$, where α_2 is the average number of overflow pages accessed when inserting one document and is estimated as $\lceil \frac{1}{b_2} \rceil$.

$I_{ST} = 2sh + \alpha_3$, where sh is at most $\lceil \log_k N \rceil - 1$, and α_3 is the average number of accessed nodes to reorganize the file when an overflow occurs. The α_3 is estimated as $\lceil \frac{sh}{b_2} \rceil$.

3.4 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 [7].

(NA:Not Applicable)

Methods / Paramters	HS File	Quick Filter	S-tree
N	10,000 100,000	10,000 100,000	10,000 100,000
P	1 Kbytes 2 Kbytes	1 Kbytes 2 Kbytes	1 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

Table 1 : The values for parameeters

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.

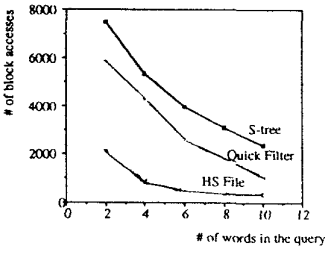

Figure 1 : Analytic retrieval performance

To save the space, Figure 1 show the retrieval performances of the dynamic signature file methods on the CASE 3. 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.

Since we assumed that document signatures follow unform 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 α_1, α_2 and α_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.

4 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

Method \ Experiment	CASE 1	CASE 2	CASE 3
S-tree	1845	1870	19240
Quick Filter	1151	1163	11430
HS File	983	1031	10215

Table 2 : Storage overhead

range of $[-2^{31}, 2^{31}-1]$: 1) a uniform distribution, 2) a normal distribution N(0,σ), where $\sigma = 1/3x2^{31}$ and 3) an exponential distribution $1/\theta x e^{(x+2^{31})/\theta}$, where $\theta = 1/4x2^{32}$. The experiments are also performed for various sizes of databases and various performance parameters.

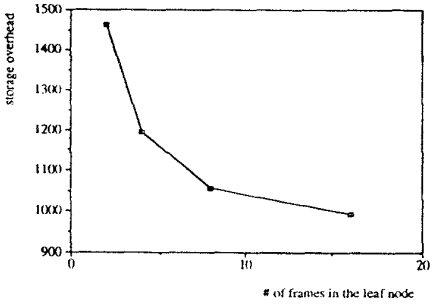

Figure 2. Storage overhead of the *HS* file

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 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. 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 experiment 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 the experiment, 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 2. We can see through the figure that when constructing *HS* file, the more the number of frames in the leaf nodes, the higher 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* file is reduced.

Figure 3 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 3(a) shows that the *HS* file achieves greatly similar retrieval performance independently of data distributions. However, we can see through Figure 3(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.

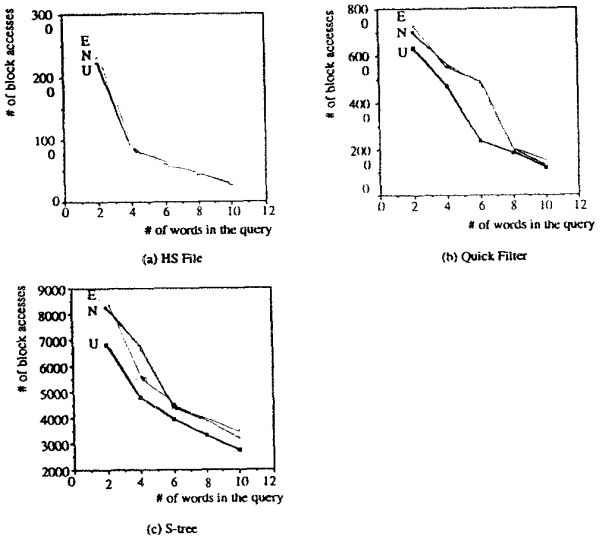

Figure 3. Retrieval performance

Figure 4 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 4 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 $\sim$ 360% and about 200 $\sim$ 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[8].

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 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 rate is computed as follows, where E and T indicate a theoretical result and an experimental result, respectively. The error rates on retrieval are $0.5 \sim 7\%$ and $0.5 \sim 15\%$ in case of 10,000 and 100,000 documents respectively.

Error Rate=$[Max(T,E) - Min(T,E)]/Max(T,E)$

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* is more than that of *HS* file and *Quick Filter*. 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.

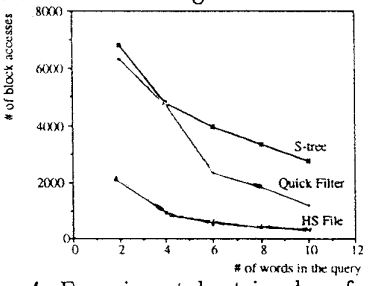

Figure 4. Experimental retrieval performance

5 Discussion

In this section, we summarize the results in a guide 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. They extend criteria for storage structures by Tiberio. Secondly, we provide guidelines for the most effective usage to a given operational environment.

5.1 Selection Criteria

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.

The specification of a 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 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[6]. 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. One of criteria that we have necessarily to consider is SS7 = support of applications such as range queries, partial match queries and synonym handling. The last criterion is SS8 = space requirements. This will make it possible to consider the storage structures according to their convenience in terms of space overhead.

5.2 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 signauture file methods according to the criteria of storage structures. The SSF is a simple file of fixed-length signatures[9]. 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 this 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. The applications such as image files are 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* and *HS file* are used. However, even for high weight queries, *HS* file is more efficient than *quick filter*.

Dynamic Storage Structures	Criteria							
	Query weights			Size of data file			APP	SR
	SS1	SS2	SS3	SS4	SS5	SS6	SS7	SS8
SSF	D	D	D	C	D	E	F	A
S-tree	F	D	B	C	D	D	F	E
Quick Filter	E	D	B	B	B	A	F	B
HS	B	A	A	A	A	A	F	B

Table 4 : Degrees of convenience

6 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. They should allow users to express specific characteristics of their 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*, 1989, pp. 176-185.

[2] U. Deppisch, "S-tree: A Dynamic Balanced Signature Index for Office Retrieval," *ACM SIGIR*, 1986, pp. 77-87.

[3] C. Faloutsos, "Signature-based Text Retrieval Methods : A Survey," *IEEE Computer Society Technical Committee on Data Engineering*, Vol. 13, No. 1, Mar. 1990, pp. 25-32.

[4] Z. Lin and C. Faloutsos, "Frame-Sliced Signature Files," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 4, No. 3, Jun. 1992, pp. 281-289.

[5] C. S. Roberts, "Partial Match Retrieval via the Method of the Superimposed Codes," *Proc. IEEE 67*, Dec. 1979, pp. 1624-1642.

[6] P. Tiberio and P. Zezula, "Selecting Signature Files for Specific Applications," *5th Annual European Computer Conference*, May 1991, pp. 718-725.

[7] 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, Feb. 1993, pp. 179-183.

[8] 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, September, 1994, pp. 571-580.

[9] P. Zezula, F. Rabitti and P. Tiberio, "Dynamic Partitioning of Signature Files," *ACM Trans. on Information Systems*, Vol. 9, No. 4, Oct. 1991, pp. 336-369.