The *HS* File : A New Dynamic Signature File Method for Efficient Information Retrieval

Jae Soo Yoo¹, Yoon-Joon Lee¹, Jae Woo Chang² and Myoung Ho Kim¹

¹ Dept. of Computer Science

Korea Advanced Institute of Science and Technology

373-1, Kusung-dong, Yuseong-gu, Taejeon, 305-701, Korea

² Dept. of Computer Engineering, Chonbuk National University Chonju,
Chonbuk 560-756, Korea

Abstract. Many works on the signature file methods have been made in the past, but they are mainly for static environments. However, many recent applications in practice require a dynamic information storage structure that effectively supports insertions, deletions and updates. Though there are a few signature file techniques for dynamic environments, they suffer from serious performance degradation when query signature weights are light. In this paper, we propose a new dynamic signature file organization, called the hierarchical signature(*HS*) file, that solves the problem of light query signature weights. We perform simulation experiments by using wide range of parameter values. We show through performance comparison based on experiments that the *HS* file improves performance significantly in both the retrieval time and the storage overhead over the other dynamic signature file methods proposed earlier.

Keywords : information retrieval, signature file, performance evaluation, dynamic environment

1 Introduction

Information retrieval and management have been a major field of computing for a long time. 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[2]. An approach widely advocated for efficient retrieval of unformatted data is to use the signature file method, which has been shown to be effective for textual data processing[5].

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[5]. 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. Although many

extraction methods for creating the signature from a document have been proposed, most of the signature file methods typically use superimposed coding. 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 main concern of this paper.

Many works on the storage structure of the signature file have been made in the past, but they are mainly for static environments[7, 8, 1, 9]. 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[10].

There are a few signature file techniques for dynamic environments. The *S-tree* proposed in [3] 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[10]. It also has much space overhead. The *Quick Filter* proposed in [10] uses partitioning principles based on linear hashing, which tends to cluster the signatures having the same suffixes(or prefixes) in the same page. The *Quick Filter* has the advantage that the more the number of bits set to '1' 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*.

In this paper, we propose a new dynamic signature file organization, called the hierarchical signature(*HS*) file. The proposed *HS* file overcomes the problem of light query signature weights. The *HS* file is a height balanced multiway tree that is a hierarchy of nodes containing signatures. It uses frame sliced approach[6] to leaf node construction to improve a filtering effect of the signature file. In order to verify the efficiency of the proposed *HS* file, we perform experiments for the *HS* file, *S-tree* and *Quick Filter*. The 10,000 and 100,000 documents with various types of parameters and queries are used. The results show that the *HS* file improves retrieval performance significantly over the *Quick Filter* and *S-tree*.

The remainder of this paper is organized as follows. In section 2, we presents a new dynamic signature file method and its characteristics. In section 3, we performs simulation experiments and shows that the proposed method achieves good performances over other dynamic signature file methods. Finally, conclusions are described in section 4.

2 The Hierarchical Signature(*HS*) File

In this section we describe a new dynamic signature file organization called the hierarchical signature(*HS*) file. It solves the problem of light query signature weights that is the major obstacle in the other methods proposed earlier. The *HS* file is also space-efficient.

The *HS* file is a height balanced multiway tree that is a hierarchy of nodes containing signatures. The *HS* file has two types of nodes, namely a leaf node and a nonleaf node. It uses frame sliced approach[6] to leaf node construction to improve a filtering effect of the signature file.

2.1 The Structure of the *HS* File

Frame-based Document Signature Construction Rule. A word-signature is a bit string of length k generated by a hash function that chooses, among k bit positions, m bit positions set to '1'. Then, each word-signature is divided into c substrings, each of which has the size $\frac{k}{c}$ that is equal to the frame size. These c substrings are allocated into c frames, among f frames, that are chosen by another hash function. The frame-signature is constructed by superimposing the parts of the word-signatures mapped into that frame. If some frames are not chosen by words in the document, all bits in those frame-signatures are set to '0'. Finally, the document-signature is a concatenation of all the frame-signatures.

For example, suppose that a document is represented by four words, say "Text", "Database", "Information" and "Retrieval". Fig.1 illustrates the construction of a document-signature when the length of document-signature is twelve and the number of frames is three. We assume that the number of bits which are set to '1' in a word-signature is two and the number of frames chosen by a hash function to construct a word-signature, c , is two.

Fig.1. Construction example of a document signature

Node Construction Rule. As mentioned before, the *HS* file has two types of nodes, namely a leaf node and a nonleaf node. A leaf node consists of as many blocks as the predetermined number of frames and one pointer block, while a nonleaf node consists of a single block. The index record in the nonleaf node, which consists of a signature and a pointer, is used to branch to the subtree that potentially contain the document satisfying the given query signature. The index record in the leaf node, which consists of f frame-signatures and a pointer,

points to the real document. Node configurations of the leaf nodes and nonleaf nodes are defined as follows:

- The leaf node $\{F_1, F_2, \dots, F_f, P\}$ is a set of f frame blocks and a pointer block such that
 - $F_i : \langle fs_{i1}, fs_{i2}, \dots, fs_{ik} \rangle$, where fs_{ik} is the k -th frame-signature in the i -th frame.
 - $P : \langle p_1, p_2, \dots, p_k \rangle$, where p_k is the pointer to a real document.
 - A signature $\langle fs_{1j}, fs_{2j}, \dots, fs_{fj} \rangle$ with a pointer p_j constitutes an index record in the leaf node.
- The nonleaf node $\{S_1, S_2, \dots, S_j\}$ is a set of index records, where S_j consists of a signature and a pointer. The signature in S_j is a concatenation of f frame-signatures, each of which is constructed by superimposing all the corresponding frame-signatures in the son node of S_j and the pointer in S_j points to its son node.

Fig.2 shows an example of the *HS* file, where the number of frames is two and the size of the frame-signature is four bits. In this example, document-signatures consist of two frame-signatures and are stored into the leaf node in such a way that each frame-signature is allocated into the corresponding frame. As shown in Fig.2, the leaf node is composed of two frame blocks and one pointer block. The pointers in the pointer block point to the real documents corresponding to document-signatures. A nonleaf node includes two index records, each of which consists of a signature and a pointer. The signatures in the nonleaf node are constructed by superimposing all the signatures in the son node. For example, the signature $\langle 0111, 1110 \rangle$ in N_6 comes from superimposing all the signatures in N_4 . The signature $\langle 1110, 0011 \rangle$ in N_5 comes from superimposing all the signatures in the frames F_{31} and F_{32} of N_3 .

Now we define the *HS* file of type (b_1, b_2, f) with the following properties, where b_1 and b_2 are the blocking factors of a leaf node and a nonleaf 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 nonleaf node is composed of only one block. Each nonleaf node has at most b_2 sons and signatures.
4. The signatures in the nonleaf nodes are constructed by superimposing the signatures contained in their son node.

According to this definition, the *HS* file of Fig.2 is of type $(8,2,2)$.

2.2 Operations on the *HS* File

Retrieval. To demonstrate the retrieval process in the *HS* file, we consider a light weight query signature $S(Q) = \langle 1000, 0000 \rangle$ for the *HS* file in Fig.2. The

Fig. 2. Example of The *HS* file

second signature of the root node leads us to node N5, since the first bit position of its first frame signature contains '1'. Since the signature in N5 satisfies the query signature, the leaf node N3 is accessed.

If we arrive at the leaf node, in order to determine the access order of frames we first divide the query signature into the number of frames, f , with the same size as the frame-signature. After obtaining the f query frame-signatures, we access the corresponding frames in such a order that the weight of the query frame-signatures are high. Therefore, the first frame F_{31} in N3 is accessed first since the first query frame-signature $\langle 1000 \rangle$ has more '1's than the second one.

The reason is that if we first access the frame corresponding to the heaviest query frame-signature, the filtering effect is greatly improved. If a frame has only one frame-signature that satisfy the query frame-signature, we can avoid accessing the remaining frames. As a result, the number of block accesses is reduced. We need not access the second frame in node N3 since the second frame-signature of $S(Q)$ does not contain '1' at all. This is because the query frame-signature does not filter non-qualifying documents.

Maintenance of the *HS* File. To insert the first document signature, the *HS* file first creates a root node with a single block and a leaf node. The leaf node consists of as many blocks as the number of frames and one pointer block. The document signature is stored into the leaf node in such a way that each frame signature is allocated into the corresponding frame. Finally, the entry $(fs_1, fs_2,$

..., fs_f , ptr) is inserted into the root node. Here the frame signature fs_f is obtained by superimposing f -th frame signatures in the leaf node and ptr points to the leaf node.

When inserting new document signatures, we first find an appropriate leaf node to store the document signatures. For example, when we insert the document signature $S_{16} = \langle 0100, 1010 \rangle$ of document D_{16} into the *HS* file of Fig.2, we access root node N_6 and compare the first bit of first frame-signature of S_{16} with those of first frame-signatures in N_6 . Since the first signature $\langle 0111, 1110 \rangle$ is qualified, we access its son node N_4 . We again compare the first bit of the first frame-signature of S_{16} with those of first frame-signatures in N_4 . Because the first bits of two signatures in N_4 satisfy the first bit of the first frame-signature of S_{16} , we compare the first bit of the second frame-signature of S_{16} with those of second frame-signatures in N_4 and find the matched second signature $\langle 0011, 1110 \rangle$. As a result, the first frame-signature $\langle 0100 \rangle$ and second frame-signature $\langle 1010 \rangle$ of S_{16} are inserted into F_{21} and F_{22} in the leaf node N_2 respectively and the pointer to the document D_{16} is inserted into the pointer block. The signature $\langle 0011, 1110 \rangle$ of N_4 is changed into $\langle 0111, 1110 \rangle$ by superimposing S_{16} to itself. We repeat this process until the root node is reached. If an overflow occurs, it is solved by splitting algorithm. We call it frame bitwise bipartition strategy.

Fig.3 illustrates on the domain space how the overflow is solved in the leaf node, when we construct the *HS* file of Fig.2. In the *HS* file, the number of frames determines the dimension on the domain space and one leaf node corresponds to one region of the domain space. In this example, since the number of frames is two, we consider two dimensional domain space, where x-axis and y-axis represent first frame and second one respectively. Since the blocking factor of the leaf node is eight, the overflow occurs when inserting ninth document-signature.

In order to solve the overflow, the *HS* file first bipartitions the domain space based on x-axis and obtains Fig.3(b). As shown in Fig.3(b), the left region contains the document-signatures that the first bit of frame signatures in the first frame is '0' and the right region includes the document signatures that the first bit of frame signatures in the first frame is '1'. We call this splitting strategy frame bitwise bipartition one. Fig.3(c) shows the situation that twelve documents have been inserted without overflow since nine document signatures were stored. The overflow occurs again when D_{13} is inserted into the left region that already contains eight document signatures. We split the left region based on y-axis to overcome the overflow and then get Fig.3(d). Finally, after inserting fifteen documents, we can obtain Fig.3(e) that shows the situation of leaf node level in Fig.3. The overflow in the nonleaf node is also processed by the frame bitwise bipartition strategy.

2.3 Advantages of the *HS* File

The *HS* file reduces the number of block accesses for a document retrieval through the unique signature extraction method and retrieval process described

Fig. 3. Leaf node construction on the domain space

in 3.1.1 and 3.2.1. That is, when the *HS* file processes light weight query signatures that significantly degrade the performance of the *S-tree* and the *Quick Filter*, it accesses only a few blocks by the unique signature extraction method and retrieval process over them. For example, if the number of frames chosen by a word in extreme case is one, in the leaf node level only one random disk access is required for a single word query and thus at most n frames are scanned for n word queries. Since light weight query signatures also have many query frame-signatures with all bits set to '0', we need not access the corresponding frame blocks in the leaf nodes.

When the leaf node is constructed based on the node construction rule, the *HS* file can reduce the number of splits and save the storage space. This is because the leaf node of the *HS* file includes much more document signatures than the leaf node of the *S-tree* and the page of the *Quick Filter*. We can see through operations on the *HS* file that the *HS* file is relevant to applications which require a dynamic information storage structure.

3 Performance Evaluation

In this section, to estimate the performance of dynamic signature file methods, we actually implement them and perform simulation experiments. Table 1 shows the notations and descriptions of the input and design parameters used for performance evaluation. The values of each input and design parameter are presented in Table 2. The simulation experiments are performed for various sizes of databases and various performance parameters. We use 100 sample queries to evaluate the characteristics of the *HS* file and the performances of the dynamic signature file methods. To save the space, we discuss the performance comparison of the dynamic signature file methods only 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 remaining cases are very similar to those of this case.

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 <i>HS</i> file
m	Number of bits set to '1' by one word
w	Number of words in the query
t	Size of pointer in bytes
l	Hashing level of <i>Quick Filter</i>
b_1	Blocking factor of a leaf node in the <i>HS</i> file
b_2	Blocking factor of a non-leaf node in the <i>HS</i> file, a node in the S-tree and a page in the query signature
W(Q)	Query weight, i.e., the number of ones in the query signature

Table 1. Input/Design Parameters

First, we investigate whether frame-based document signature construction affects the retrieval performance or not. 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 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 the experiments that the more the number of frames is, the better the retrieval performance is. In this environment, the maximum number of frames that the leaf node of the *HS* file can have is sixteen since it depends on the size of document signature and the size of 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.

Fig.4 shows an experimental result on retrieval when the number of documents is 100,000 and the number of frames in the leaf node of the *HS* file is 16. It is shown that the *HS* file is much more efficient than the other dynamic signature file methods independently of the number of words in the query. From the experimental result, 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 an unique retrieval process.

When the number of documents is 100,000 and the number of frames is 16,

(NA:Not Applicable)

Methods Parameters	HS File	Quick Filter	S-tree
N	10,000	10,000	10,000
P	100,000	100,000	100,000
	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

Table 2. The values for parameters

Fig. 4. Retrieval performance of dynamic signature file methods

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 *Quick Filter*. We also found that the *Quick Filter* achieves the best insertion performance, while *HS* file is the worst. The reason is that the *Quick Filter* is constructed based on the linear hashing and *HS* file uses the frame-sliced approach to the leaf node. However, the difference of insertion performance among them is not important since it is much smaller than the difference of retrieval performance, and insertion operation occurs much less frequently than retrieval operation in the field of information retrieval.

4 Conclusions

We have proposed a new dynamic signature file method for efficient information retrieval, called the *HS* file. The *HS* file is a height balanced multiway tree that is a hierarchy of nodes containing signatures. It uses frame sliced approach to leaf node construction to improve a filtering effect of the signature file. To solve the overflow of nodes, the *HS* file utilizes a special splitting algorithm, called the frame bitwise bipartition strategy. The *HS* file has solved problems of light query signature weights that were major obstacles in the other methods proposed earlier.

We have compared the performance of the *HS* file with the *Quick Filter* and the *S-tree* in terms of retrieval time, storage overhead and insertion time. For this, we have carried out extensive performance experiments with wide range of parameter values and evaluated the space-time performance of these methods. Through the 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.

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. S. Christodoulakis and C. Faloutsos, "Design Considerations for a Message File Server," *IEEE Trans. on Soft. Eng.*, Vol. SE-10, No. 2, Mar. 1984, pp. 201-210.
3. U. Deppisch, "S-tree: A Dynamic Balanced Signature Index for Office Retrieval," *ACM SIGIR*, 1986, pp. 77-87.
4. C. Faloutsos and S. Christodoulakis, "Description and Performance File Method for Office Filing," *ACM TOIS*, Vol. 5, No. 3, 1987, pp. 237-257.
5. 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.
6. 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.
7. C. S. Roberts, "Partial Match Retrieval via the Method of the Superimposed Codes," *Proc. IEEE* 67, Dec. 1979, pp. 1624-1642.
8. R. Sacks-Davis and K. Ramamohanarao, "Multikey Access Methods based on Superimposed Coding Techniques," *ACM TODS*, Vol. 12, No. 4, Dec. 1987, pp. 655-696.
9. 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.
10. P. Zezula, F. Rabitti and P. Tiberio, "Dynamic Partitioning of Signature Files," *ACM TOIS*, Vol. 9, No. 4, Oct. 1991, pp. 336-369.