http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 8 (3): 275-283, 2009
ISSN 1812-5638

An Index Structure for Fast Query Retrieval in Object Oriented Data Bases Using Signature Weight Declustering

[1]I. Elizabeth Shanthi and [2]R. Nadarajan
[1]Department of Computer Science, Avinashilingam University for Women, Coimbatore, India
[2]Department of Mathematics and Computer Applications, PSG College of Technology, Coimbatore, India

Abstract: An important question in information retrieval is how to create a database index which can be searched efficiently for the data one seeks. One such technique called signature file based access method is preferred for its easy handling of insertion and update operations. Most of the proposed methods use either efficient search method or tree based intermediate data structure to filter data objects matching the query. Use of search techniques retrieves the objects by sequentially comparing the positions of 1s in it. Such methods take longer retrieval time. On the other hand tree based structures traverse multiple paths making comparison process tedious. This study describes a new indexing technique for object-oriented data bases using the dynamic balancing of B+ tree called SD (Signature Declustering) tree. The SD-tree represents all 1s in signatures in a compact manner that results in saving of insertion and searching time. Analytical experiments have been conducted by varying the signature length and the distribution of signature weight. The study clearly indicates the advantage of fast retrieval time in a way quite different from the other methods suggested in the past.

Key words: Signature file, information retrieval, OODB, indexing

INTRODUCTION

Information retrieval is the topic gaining more interest nowadays as the volume of data escalates in all computer-based applications. This has entrusted researchers to design more powerful techniques to generate and manipulate large amounts of data to derive useful information. Indexing plays a vital role in the fast recovery of required data from large databases.

Among the many indexing technique reported the signature file approach is preferred for its efficient evaluation of set-oriented queries and easy handling of insert and update operations. Initially applied on text data which is discussed by Faloutsos and Stavros (1984), Faloutsos (1985a), Larson (1984), Lee *et al.* (1995) and Zobel *et al.* (1998), it has now been used in other applications like office filing (Faloutsos and Stavros, 1987a), relational and object-oriented databases (Lee and Lee, 1992; Tousidou *et al.*, 2002; Yong *et al.*, 1994) and hypertext (Faloutsos *et al.*, 1990).

Signatures are hash coded abstractions of the original data. It is a binary pattern of predefined length with fixed number of 1s. The attributes' signatures are superimposed to form object's signature. To resolve a query, the query signature Sq is generated using the same hash function and compared with signatures in the signature file for 1s sequentially and many non-qualifying objects are immediately rejected. If all the 1s of Sq matches with that of the signature in the file it is called a drop. The signature file method guarantees that all qualifying objects will pass through the filtering mechanism; however some non-qualifying objects may also pass the signature test. The drop that actually matches the Sq is called an actual drop and drop that fail the test is called false drop. The next step in the query processing is the false drop resolution. To remove false drops each drop is accessed and checked individually. The number of false drops can be statistically controlled by careful design of the signature extraction method (Faloutsos and Stavros, 1987a, b) and by using long signatures (Faloutsos, 1985a).

Different approaches have been discussed by researchers to represent signature file in a way conducive for evaluating queries, such as sequential signature file in (Ishiwaka *et al.*, 1993), bit-slice signature file in (Ishiwaka *et al.*, 1993), multilevel signature file in (Chang and Hans Schek, 1989), compressed multi framed signature file in (Kocberber and Fazli, 1999), parallel signature file in (Ciaccia *et al.*, 1996), S-tree and its variants (Deppisch, 1986; Tousidou *et al.*, 2000), signature graph (Chen and Yibin, 2004) and signature tree (Chen, 2002, 2005; Chen and Yibin, 2006).

Corresponding Author: I. Elizabeth Shanthi, Department of Computer Science, Avinashilingam University for Women, Coimbatore, India

In the recently proposed signature tree (Chen and Yibin, 2006) signatures are stored in a binary tree like structure where tree nodes denote the signature bit positions and edges bit values. For signature insertion the existing tree is traced to find the appropriate position for the new signature. While searching the path is dictated by the node values of the tree from root rather than by the position of 1s in the query signature. This makes searching longer. On contrary in this study we focus on the advantages that could arise from storing only the set bits (1s) of signatures in a structure called SD-tree (Signature Declustering tree). A signature is assumed to be a cluster of 1s which is distributed over a set of leaf nodes in a B+ tree like structure. Further by clustering signatures having similar bit set in special nodes called signature nodes, any incoming query can readily be answered in a single access.

A SUMMARY OF SIGNATURE FILE TECHNIQUES

A signature is a bit string formed from a given value. Compared to other index structures, signature file is more efficient in handling new insertions and queries on parts of words. Other advantages include its simple implementation and the ability to support a growing file. But it introduces information loss which can be minimized by carefully selecting the signature extraction method. Techniques for signature extraction such as Word Signature (WS) (Faloutsos and Stavros, 1984, 1987a; Faloutsos, 1985a, b) Superimposed Coding (SC) (Dervos *et al*., 1998; Faloutsos and Stavros, 1987a), Multilevel Superimposed Coding (Lee *et al*., 1995), Run Length Encoding (RL) (Faloutsos, 1985a, b; Faloutsos and Stavros, 1987a), Bit-block Compression (BC) (Faloutsos, 1985a, b), Variable Bit-block Compression (VBC) (Faloutsos, 1985b), (Faloutsos and Stavros, 1987a) have been discussed. This encoding schemes set a constant number say m, of 1s in the range [1..F], where, F is the length of the signature. The resulting binary pattern with m number of 1s and (F-m) number of 0s is called a word signature. The signature of a text block or object can be obtained by superimposing (logical OR operation) all its constituent signatures i.e., word signatures for text block and attributes' signatures for object. The set of all signatures form a signature file. An example of superimposed coding and a sample query evaluation is given below:

Information	0010 0100
Retrieval	0100 0001
Block signature	0110 0101

Sample queries

- Matching query

Key word = Information	0010 0100
Query descriptor	0010 0100
Block signature matches	(Actual drop)

- False Match query

Key word = Coding	0010 0001
Query descriptor	0010 0001
Block signature matches but keyword does not	(False drop)

- Non-matching query

Key word = Information	0010 0100
Key word = Science	0000 0110
Query descriptor	0010 0110
Block signature does not match	

Applications of signatures

- As an access method for documents

Database access methods useful for text retrieval such as full text scanning, inversion, clustering, multi attribute retrieval methods like hashing and signature files are discussed by Faloutsos and Stavros (1985) and Larson (1984). Here, the documents are stored sequentially in the text file. Signatures which are abstractions of the documents are stored in the signature file. The latter serves as a filter on retrieval. It helps in discarding a large number of non-qualifying documents. Signatures have been applied in areas rich in text documents like telephone directory (Roberts, 1979), office systems (Faloutsos and Stavros, 1987a), optical and magnetic disk access (Faloutsos and Raphael, 1988), data base management system and library automation.

- As an indexing method for large text file (Kent *et al*., 1990; Zobel *et al*., 1998)
- As an access method for formatted data (Roberts, 1979)

According to Faloutsos and Stavros (1987b) other environments include.

- To speed up searching in editor
- To compress a vocabulary
- For a spelling checking program
- In differential files

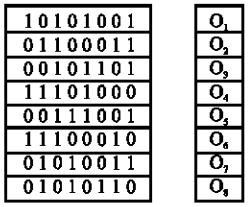

Fig. 1: A typical sequential signature file

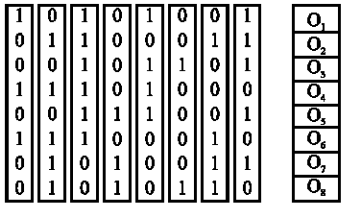

Fig. 2: A typical bit-sliced signature file

Sequential Signature File (SSF): The signatures are sequentially stored in a file called SSF as in Fig. 1. In single level signature methods every signature must be accessed and tested. Since signatures are abstractions of original data with smaller size, the method is faster than sequential scan of objects themselves. This method is easy to implement and requires low storage space and low update cost. The disadvantage is that more the number of objects exist, the more is the time spent on scanning signature file (Faloutsos and Raphael, 1988; Chen, 2005). Therefore, it is generally slow in retrieval (Fig. 1). To support faster access multilevel signature methods are suggested.

Bit-Sliced Signature File (BSSF): BSSF stores signatures in a column-wise manner as shown in Fig. 2. Thus, F bit-slice files one for each bit position of the set signatures is used. In retrieval only a part of the F bit-slice files have to be scanned and hence the search cost is lower than SSF. However, update cost is higher. This is because a new signature insertion requires about F disk accesses one for each Bit-slice file (Faloutsos and Raphael, 1988; Ishiwaka *et al*., 1993).

Compressed Bit-Sliced Signature File (CBSSF): By choosing a proper hashing function for signature extraction the value for m is forced to be one. F has to be increased to maintain the false drop probability. This creates a sparse matrix which is easy to compress (Faloutsos and Raphael, 1988; Kocberber and Fazli, 1999). A simple way to compress is to replace each one with its corresponding physical address.

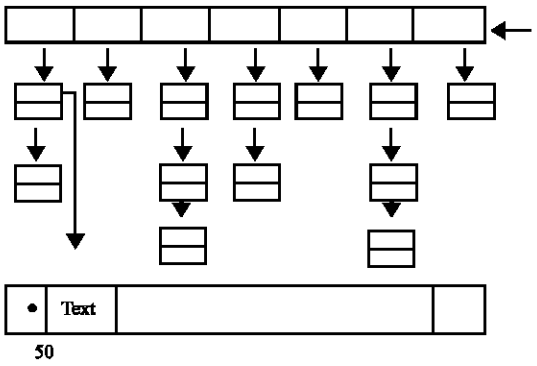

Fig. 3: A typical compressed bit-sliced signature file organization

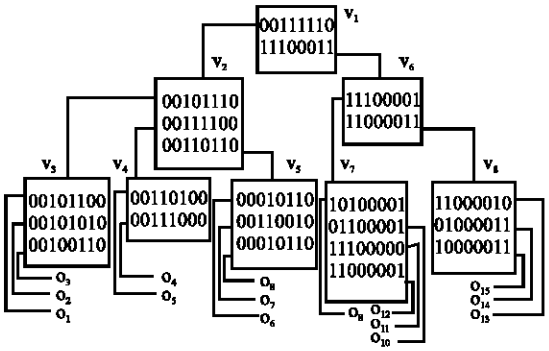

Fig. 4: A typical S-tree organization

Here, the hash table has a list of pointers pointing to the heads of linked list (Faloutsos and Raphael, 1988). For example assume that the word text has its first bit set to 1 and it appears at the 50th byte of text file then searching the first bucket list, the position of the word text is found. Although, this approach gives some space saving, the number of false drops will definitely be increased due to sparse signature files.

S-tree: S-tree (Deppisch, 1986) is a B+ tree like structure (Bayer and Unterauer, 1977; Comer, 1979) with leaf nodes containing a set of signatures with their Object Identifiers (OIDs). A typical compressed bit-sliced signature file organization shown in Fig. 3.

The internal nodes are formed by superimposing the lower level nodes as in Fig. 4. For example to retrieve a query signature Sq = 11000000, the S-tree is searched top down. From root node then v_1 is compared and searching moves to v_6. In the next level both v_7 and v_8 match and finally the searching ends up with signatures o_{11}, o_{12} in v_7 and o_{13} in v_8.

The advantage is simple tree searching way of obtaining signatures rather than searching the whole

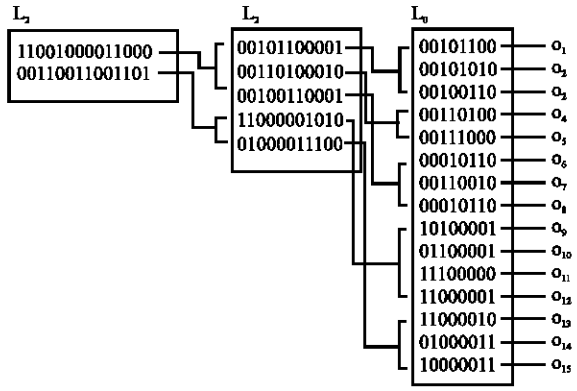

Fig. 5: A typical multilevel signature file organization

signature file. The disadvantage is that due to superimposing internal nodes in the upper level tend to have more weight which ultimately decreases selectivity.

The S-tree has been further improved by Tousidou *et al.* (2000, 2002). In Tousidou *et al.* (2000) a number of new split methods namely linear split, quadratic split, cubic split and hierarchical clustering for S-tree is proposed to improve query response time. In (Tousidou *et al.*, 2002) a new hybrid scheme combining linear hashing, S-tree and parametric weighted filter is used to evaluate subset-superset queries.

Multilevel signature file: The structure is similar to S-tree. However, a signature at non-leaf node is formed by superimposed coding from all text blocks indexed by the sub tree of which the signature is the root. Figure 5 shows multilevel signature file for the set of signature values shown in Fig. 4. For more details refer (Lee *et al.*, 1995). Though this method improves selectivity in an internal node, it requires more space. An improved method for multilevel signature file is discussed in (Chang and Hans Schek, 1989).

Signature graph: The signature file is organized as a trie like structure (Chen and Yibin, 2004, 2006). However, the path visited in the graph to find a signature that matches a given query signature corresponds to a signature identifier which is not a continuous piece of bits differentiating the signature graph from trie. Though signatures are represented compactly, the search path length is not same for all queries. In other words the graph is not balanced. In worst case, it degrades to a signature file. Figure 6 shows the signature file and the corresponding signature graph.

Signature tree: The signature tree is a binary tree like structure with nodes representing the bit positions and

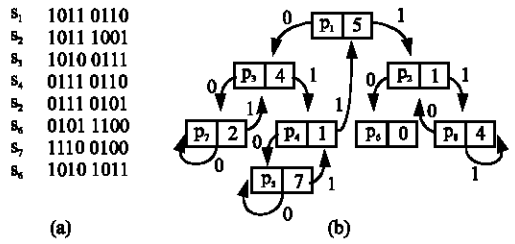

Fig. 6: A typical signature file and signature graph

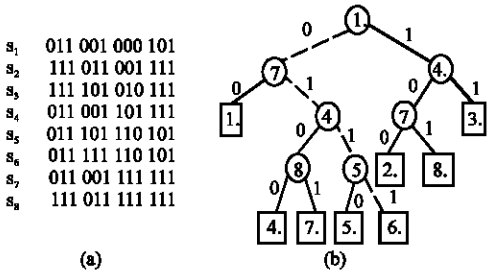

Fig. 7: A typical signature file and signature tree

left and right sub tree leading to binary values 0 and 1, respectively. Each signature is identified from the root by checking the bit positions dictated by the nodes. Nevertheless for a query signature the tree is searched top to bottom according to the bit positions dictated by the nodes rather than the 1s in query signature. Also, for a match with bit 1 searching follows the right sub tree and for 0s both left and right sub trees. That is for a balanced signature tree more than one path is traversed. Figure 7 indicates a signature file and its corresponding signature tree. The thick lines in Fig. 7 shows the signature identifier corresponds to s_3.

THE STRUCTURE OF SD-TREE

Here, we describe the structure of SD-tree. There are three types of nodes:

- Internal nodes
- Leaf nodes
- Signature nodes

The internal nodes and leaf nodes are somewhat similar to the internal nodes and leaf nodes of B+ trees, respectively. The internal nodes form the upper tree and leaf nodes at last but one level. The signature nodes are at the bottom level of the SD-tree. We will now explain the structure of the nodes in detail. To make discussion simple, we assume the tree of order 3 for a signature file with 8 block signatures of length 12.

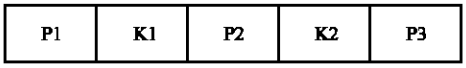

Fig. 8: A typical structure of an internal node

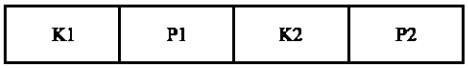

Fig. 9: A typical leaf node entry

Structure of interual node: An internal node of SD-tee is shown in Fig. 8. Like B+ tree in an internal node pointers and keys alternate each other. For a tree of order 3 the internal node has two keys K1 and K2 and three pointers P1, P2, P3. These pointers are tree pointers pointing to the nodes at the lower level. While searching the tree left pointer is followed for values less than or equal to the node value, else right pointer is followed for values greater than the node value as in B+ tree.

Structure of a leaf node: The leaf nodes appear in the last but one level of the SD-tree. Like B+ tree all the key values appear in ascending order of their values. But unlike B+ tree in SD-tree each value is followed by a signature chain instead of data pointer and hence there is no sequential pointers between the leaf nodes. This is shown in Fig. 9. Pointers P1 and P2 point to the corresponding signature nodes for K1, K2.

Structure of signature node: The structure of a signature node is shown in Fig. 10. The signature node for Ki has 2^{i-1} binary combinations denoting the possible prefixes. When a signature Su with 1 in the ith position is to be inserted the intermediate prefix formed so far is compared with the binary combinations in the signature node at Ki and u is inserted in the list.

Overall structure of SD-tree: Consider the partially filled SD-tree shown in Fig. 11a for discussion. This tree of order 3 has been constructed for signature length (F) 12.

The root node has the value 6. In the next level internal node has the value 2 in the left sub tree and finally it leads to the leaf node with values 1 and 2. Consider the insertion of signatures from signature file Fig. 11b. After the tree construction for the given signature length is over, the first signature S_1 is read. The first set bit of S_1 is at position 2. Hence, leaf node with value 2 is accessed and its signature node followed. The possible prefixes possible at signature node 2 are 0 and 1. S_1 is therefore added simply in the prefix list 0 if it does not exist; or the prefix is created and signature value inserted as 0-1. For S_2 with set bit positions at 1 and 3

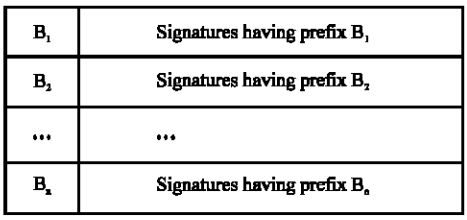

Fig. 10: A typical signature node entry

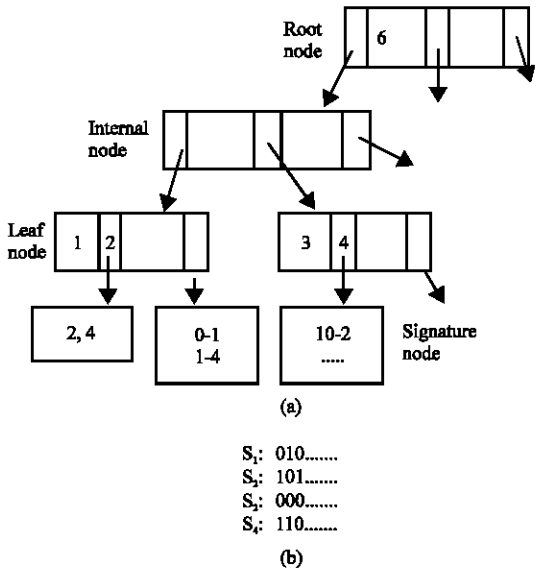

Fig. 11: Overall structure of SD-tree

signature value 2 is inserted at signature node 1 with no prefix and signature node 3 with prefix 10.

The storing of binary prefixes helps in faster query signature processing. For example consider Sq = 1000 1001. To find all matching signatures of Sq using signature tree, it is traversed from root and node values are compared with the bit positions of Sq. If the bit value is 0, both left and right edges are marked; else if the bit value is 1 right edge is marked and process continues until leaf nodes are reached which are the matching signatures of Sq. Using signature tree multiple paths are traversed to find the list of matching signatures.

Consider the same Sq search using SD-tree. The last occurrence of 1 in Sq is at position 8. The search algorithm hence forms the binary prefix as 1000 100 using the position of 1s. A single tree access to signature node 8 will read all the matching signatures from the list with prefix 1000 100. Hence, regardless of the bit pattern of Sq, SD-tree brings out all matching signatures in a single tree access.

TREE SEARCH AND UPDATE

Here, the algorithms shows for signature insert, delete and search operations on SD-tree.

Insertion: The algorithm for signature insertion is outlined here:

```
Insert (Su) //u is the signature No.
Input : The signature to insert Su.

1. Let i1, i2, ….. in be the positions of 1 in Su.
2. Move i2 to in to queue Q;
 B = NULL.//prefix bit pattern is empty
3. If (i1 = 1) then begin
      Access leaf node i1;
 insert u in the signature node;
      B = strcat ( B, '1');//indicate set bit
 end
 else begin
 for k = 1 to i1 – 1 do
 B = strcat ( B, '0');
 Access leaf node i1;
 insert u in the signature node @ prefix B;
 B = strcat ( B, '1');
 end
 f = i1; // store current bit position in f
4. While Q not empty do
 begin
 read x from Q;
 if (x = f+1) then
 begin
 Access leaf node x;
 Write u @ prefix B;
 end
 else begin
 for k = f +1 to x-1 do
 B = strcat ( B, '0');
 Access leaf node x;
 Write u @ prefix B
 end
 B = strcat (B, '1');
 f = x;
 end. // until queue is empty
```

Searching: The following algorithm outlines the steps to search for signatures matching a given query signature Sq.

```
Search(Sq)
Input: The (query) signature to search.
Output: The list of signatures matching the given signature.

Let i1, i2, ….. in be the positions of 1 in Sq.
Move i2 to in-1 to queue Q. B = NULL;
If (i1 = 1) then B = strcat ( B, '1');
 else begin
 for k = 1 to i1 _1 do
 B = strcat (B, '0');
 B = strcat (B, '1');
 end;
 f = i1; // Store the current bit position
4. While Q not empty do
 begin
 read x from Q;
 If (x = f+1) then B = strcat (B, '1')
 else begin
 for k = f+1 to x-1
 B = strcat (B, '0');
 B = strcat (B, '1');
 end;
 end. // until Q is empty
Access leaf node in.
Search for B.
Read and output the list of signatures.
```

Deletion: The algorithm to delete a signature from SD-tree is described below:

```
Delete (Su)
Input : Su, the signature to delete.

1. Let i1, i2, ….. in be the positions of 1 in Su.
2. For each ik (1 ≤k≤n) form prefix B as in Insert (Su).
3. Access the leaf node and follow the signature node;
4. Access prefix B and search for u.
5. If present, delete it.
6. Repeat steps 2 through 5 for all iks.
```

RESULTS AND DISCUSSION

To validate the proposed structure we implement SD-tree using Mat lab and run simulation experiments on different datasets. The parameters considered in the experiments are Signature length (F), number of signatures in the signature file (n), Signature weight (m) and distribution of signature weight (d). The experiments were carried out in a standalone system with Intel Pentium IV processor. The main memory size is 512 MB and 40 GB hard disk capacity.

Time complexity: Like in other signature applications we use the response time as the performance measure (Kocberber and Fazli, 1999). To validate the SD-tree the time complexity of insert, search and delete algorithms

have been analytically compared with signature tree (Chen and Yibin, 2006). The time complexity of the insert algorithm is the sum of time taken to construct the B+ tree of order p and the time taken for inserting. Since B+ tree is constructed to insert values in ascending order of 1, 2, F where, F is the length of the signature for a given dataset. Compared to the use of B+ tree as index structure for large datasets the value F is minimum and hence the time taken for SD-tree construction. So, the time complexity for insertion is bounded by O (n.m) only, for s signature file with n signatures, where, m is the No. of 1s in the given signature. The same for the signature tree generation algorithm is bounded by O (n.F), F being the signature length. This saves insertion time in SD-tree substantially. Deletion follows similar steps as insertion and so the time complexity is same for both. Another desirable characteristic of SD-tree is that as the signature length (F) increases by varying the tree order (p), the height of the SD-tree (h) can be kept small to promote faster search. For a balanced signature tree the height of the tree is bounded by O ($\log_2 n$), n is the number of leaf nodes. The same for dynamically balanced SD-tree is given by O ($\log_p (F/p-1)$). This implies shorter the tree, faster is the traversal.

Similarly, the cost of searching a balanced signature tree will be O ($\lambda.\log_2 n$) on the average, where, λ represents the number of paths traversed. In SD-tree search time for a query with the last set bit in ith position is the sum of time taken to access the leaf node (T_{li}) and signature node search time (T_{si}). This is given by:

$$T_s = T_{li} + T_{si}$$

where, T_{li} is constant for all leaf nodes for a dynamic balanced structure like SD-tree and T_{si} increases as the value of i increases. Hence, the search time is bounded by O ($T_{li} + 2^{i-1}$). Obviously, compared to the λ-value, Tli is very small which is another benefit of SD-tree.

SD-tree maintenance and space overhead: SD-tree maintenance is quite simple that the tree is not subject to extensive node split or merges. This is because once the tree is constructed for a given signature length in an application it never changes. And so the insertions and deletions do not affect the node values or the height of the tree. Operations are reflected only in the signature node. To analyze the query response time the signature weight distribution was fixed as 100, 50, 40, 30 and 20%. The length of the signature was varied from 12, 10, 8 and 6 and results noted. The values are shown in Fig. 12-17.

It is obvious that the query search time for SD-tree is lesser than signature insertion time. For swd of 50, 40, 30 and 20% the lower and upper bytes of signatures were considered separately and values noted. The results are shown in Fig. 13-16, respectively. All the outputs clearly indicate that the time taken for signature insertion and query response is slightly higher in upper levels. Nevertheless the query response time is lesser than signature insertion time. As the structure complexity increases in signature nodes in upper levels the swd was analyzed for both ends separately.

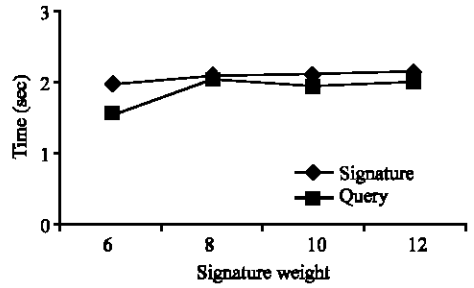

Fig. 12: The signature insertion time and query response time with the 100% signature weight distribution (swd)

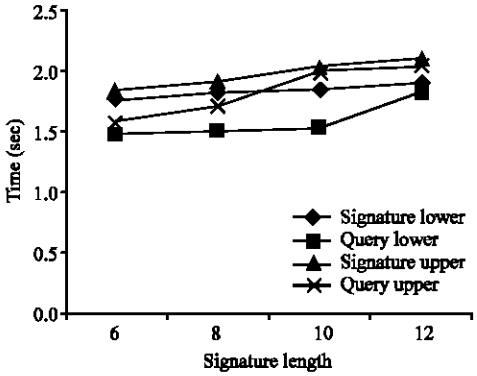

Fig. 13: The upper and lower bytes of 50% signature weight distribution

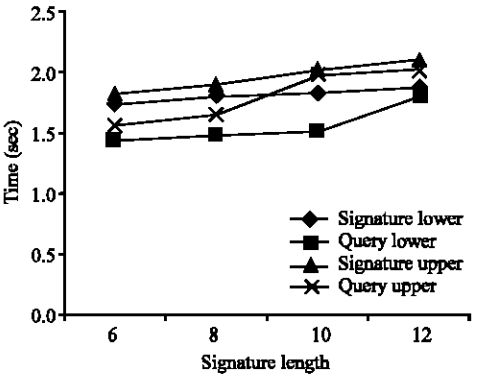

Fig. 14: The upper and lower bytes of 40% signature weight distribution

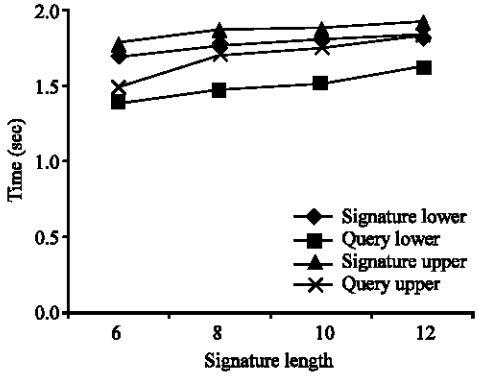

Fig. 15: The upper and lower bytes of 30% signature weight distribution

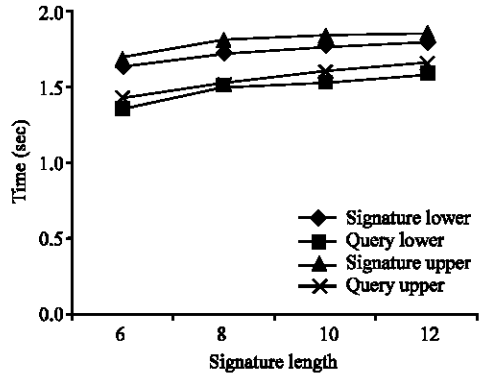

Fig. 16: The upper and lower bytes of 20% signature weight distribution

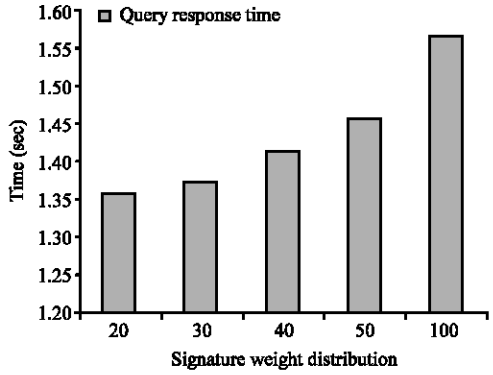

Fig. 17: Query response time for F = 12

Figure 17 shows the query response time of various swds for signature length 12. It can be seen that as the swd increases the time taken to form the binary prefix also increases. In other words, the query response time is directly proportional to the signature weight.

Figure 18 shows the space overhead of SD-tree. The structure complexity of signature nodes increases from

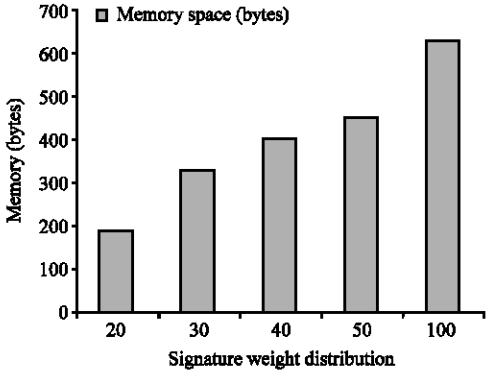

Fig. 18: Space overhead in SD-tree for F = 12

lower to upper nodes in SD-tree. This increases the memory space occupied by signature nodes.

The size of binary prefix bit pattern used in signature nodes grow from lower level to upper level nodes. For signature length 12 and swds 20, 30, 40, 50 and 100% the number of bytes occupied by signatures is plotted in Fig. 18. If b is the byte size and F is the signature length then the space complexity is bounded by O (b. F).

CONCLUSION AND RESEARCH DIRECTIONS

This study presents a novel way to represent signatures in a B+ tree like structure called SD-tree and analyzed the performance for signature insertion and query response time. By varying the signature length and distribution of 1s in the signature the query response time was noted and results plotted. Analytical results prove the superiority of SD-tree over signature tree with respect to insertion time and search time. The space overhead in SD-tree may be higher due to the presence of binary prefixes in higher order signature nodes, but the flexibility provided by the SD-tree outweighs all besides simple maintenance and faster query retrieval time.

The study is proposed to extend in the following directions. The simulated experiments are to be run and verified on a real Object Oriented Data Base system. Another direction is when the signature weight is more than 50%, use 0s so that No. of signature nodes accessed for insertion and search is optimal. Also the structure can be modified to support point and range queries in Object Oriented Data Base system.

REFERENCES

Bayer, R. and K. Unterauer, 1977. Prefix B-trees. ACM Trans. Database Syst., 2: 11-26.

Chang, W.W. and J. Hans Schek, 1989. A signature access method for the starburst database system. Proceedings of the 15th International Conference On Very Large Data Bases, August 22-25, San Francisco, CA, USA., pp: 145-153.

Chen, Y., 2002. Signature files and signature trees. Inform. Process. Lett., 82: 213-221.

Chen, Y. and C. Yibin, 2004. Signature file hierarchies and signature graphs: A new index method for object-oriented databases. Proceedings of the 2004 ACM Symposium on Applied Computing, March 14-17, New York, USA., pp: 724-728.

Chen, Y., 2005. On the signature trees and balanced signature trees. Proceedings of the 21st International Conference on Data Engineering, April 05-08, Washington, DC, USA., pp: 742-753.

Chen, Y. and C. Yibin, 2006. On the signature tree construction and analysis. IEEE Trans. Knowledge Data Eng., 18: 1207-1224.

Ciaccia, P., T. Paolo and Z. Pavel, 1996. Declustering of key-based partitioned signature files. ACM Trans. Database Syst., 21: 295-338.

Comer, D., 1979. The ubiquitos B-tree. Comput. Surveys, 11: 121-137.

Deppisch, U., 1986. S-tree: A dynamic balanced signature index for office retrieval. Proceedings of the 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1986 New York, USA., pp: 77-87.

Dervos, D., Y. Manolopoulous and P. Linardis, 1998. Comparison of signature file methods with superimposed coding. J. Inform. Process., 65: 101-106.

Faloutsos, C. and C. Stavros, 1984. Signature files: An access method for documents and its analytical performance evaluation. ACM Trans Office Inform. Syst., 2: 267-288.

Faloutsos, C. and C. Stavros, 1985. Design of a signature file method that accounts for non-uniform occurrence and query frequencies. Proceedings of the 11th international conference on Very Large Data Bases Vol. 11, August 21-23, Stockholm, Sweden, pp: 165-170.

Faloutsos, C., 1985a. Signature files: Design and performance comparison of some signature extraction methods. Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, 1985 New York, USA., pp: 63-82.

Faloutsos, C., 1985b. Access methods for text. ACM Comput. Surveys, 17: 49-74.

Faloutsos, C. and C. Stavros, 1987a. Description and performance analysis of signature file methods for office filing. ACM Trans. Office Inform. Syst., 5: 237-257.

Faloutsos, C. and C. Stavros, 1987b. Optimal signature extraction and information loss. ACM Trans. Database Syst., 12: 395-428.

Faloutsos, C. and C. Raphael, 1988. Fast text access methods for optical and large magnetic disks: Designs and performance comparison. Proceedings of the 14th International Conference on Very Large Databases, August 29 September 01, San Francisco, CA, USA., PP: 280-293.

Faloutsos, C., L. Raymond, P. Catherine and S. Ben, 1990. Incorporating string search in a hypertext system: User interface and signature file design issues, Hypermedia, 2: 183-200.

Ishiwaka, Y., K. Hiroyuki and O. Nobuo, 1993. Evaluation of signature files as set access facilities in OODBs. Proc. ACM SIGMOD Record, 22: 247-256.

Kent, A., R. Sacks-Davis and K. Ramamohanarao, 1990. A signature file scheme based on multiple organizations for indexing very large text databases. J. Am. Soc. Inform. Sci., 41: 508-534.

Kocberber, S. and C. Fazli, 1999. Compressed multi-framed signature files: An index structure for fast information retrieval. Proceedings of the 1999 ACM symposium on Applied computing, February 28 March 02, New York, USA., pp: 221-226.

Larson, P.A., 1984. A method for speeding up text retrieval. ACM SIGMIS, Database Winter, 15: 19-23.

Lee, W.C. and D.L. Lee, 1992. Signature file methods for indexing object-oriented database systems. Proceeding 2nd International Computer Science Conference, 1992 Hong Kong, pp: 616-622.

Lee, D.L., K. Young Man and P. Gaurav, 1995. Efficient signature file methods for text retrieval. IEEE Trans. Knowledge Data Eng. TKDE, 7: 423-435.

Roberts, C.S., 1979. Partial-match retrieval via the method of superimposed codes. Proc. IEEE, 67: 1624-1642.

Tousidou, E., N. Alex and M. Yannis, 2000. Improved methods for signature-tree construction. Comput. J., 43: 301-314.

Tousidou, E., B. Panayiotis and M. Yannis, 2002. Signature-based structures for objects with set-valued attributes. Inform. Syst., 27: 93-121.

Yong, H., L. Sukho and K. Hyoung-Joo, 1994. Applying signatures for forward traversal query processing in object-oriented databases. Proceeding 10th International Conference Data Engineering, Feb. 14-18, Houston, TX, USA., pp: 518-525.

Zobel, J., M. Alistair and R. Kotagiri, 1998. Inverted files versus signature files for text indexing. ACM Trans. Database Syst., 23: 453-490.