A Signature Access Method for the Starburst Database System

Walter W. Chang
Hans J. Schek[1]

IBM Almaden Research Center

Abstract

This paper describes a new signature generation method for constructing multi-level signature files to support both relational queries which contain multiple conjunctive (AND) predicates and generic document text queries. We describe the major problems with traditional multi-level signature files and then describe how to build multi-level signature files using a new composite method of parent signature generation. Performance of this signature generation scheme improves as more key fields are provided in the query. A combinatorial error problem common to all multi-level signature structures is identified and addressed. We show how a signature access method can provide query support for a large number of fields in a relation for which no index exists and can dramatically reduce the number of relation tuples that must be accessed during a normal scan. If one or more fields of the relation contain long field data such as text, the same signature mechanism can also provide query support for text search predicates.

1. Introduction

Signatures have been used extensively for information retrieval in electronic office and document filing systems and to some extent in database systems (Sacks-Davis87, Pfaltz80, Roberts79). Signatures are bit-vectors formed by a hash encoding of data objects such as long fields of text or relation tuples (records). Signatures are useful in filtering large quantities of data when queries are performed. In relational database systems, queries are traditionally answered by executing a plan generated by a query optimizer. The plan may require performing relatively expensive relation scans over a potentially large number of tuples, scanning one or more B-tree indexes which have been created and maintained over the key fields of the query, or by using some combination of the two.

In this paper we propose a signature access method within a relational database system to support conjunctive multi-field queries. If fields in a relation are used to store text data as externally defined abstract data types (Wilms88) we show how text search predicates can be supported by the same signature mechanism. We show that signature files can be incorporated into the Starburst extensible database architecture (Lindsay87) by extending an existing B-tree access method.

In Section 2 we present an overview, a definition of terminology, a discussion of single level signature files and the motivation for multi-level signature file structures. In Section 3 we describe the two main problems with multi-level signatures. Our proposal to solve these problems is presented in Section 4. In Section 5 we describe how an implementation of a multi-level signature file was achieved in the Starburst extensible DBMS. In Section 6. we discuss relevant query optimizer issues. In Section 7 we present concluding remarks.

2. A Summary of Signature Techniques

2.1. Signature Description and Terminology

Signatures compactly encode information about an object. For example, bits in the signature may represent field substrings of length k called $k-tuples$ (Harrison71). Examples of other signature encoding techniques are discussed in (Roberts79, Pfaltz80, and Deppisch86).

Data objects and query values are encoded using the same signature algorithm. When the bits of the query signature completely cover the signature bits of the data objects, the data object is a candidate that may satisfy the query. These data objects are then accessed and examined. Access of candidate objects that fail an exact match test are called *false drops*. False drops are due to hash collisions in the object signatures. Matches are called *hits*. Ideally, a query signature will reject all data signatures where the original data objects do not satisfy the specified query.

Signatures should be significantly smaller than the actual data object (10-20%), easy to compute, and provide a high degree of selectivity or data filtration. Prior work has shown that, optimal selectivity usually occurs when 50% of the bits in the signature pattern are set to 1s (Roberts79, Severance76). If too few or too many signature bits are set, the query signature cannot reject a large number of data object signatures and consequently, many additional objects must be accessed and examined. Under some conditions when duplicates are frequent, (Sacks-Davis83) and (Roberts79) show that values less than 50% are desirable.

[1]Current address: ETH Swiss Federal Institute of Technology, Switzerland.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the Fifteenth International
Conference on Very Large Data Bases

Amsterdam, 1989

All signature generation algorithms fall into one of two categories: superimposed or disjoint. Signature generation methods dependent on partitions within the data object (such as fields in a tuple) are called disjoint coding. For each partition or field of the object, different signature functions may be used. Each of these functions may yield signatures of different lengths. If one signature function is applied over an entire data object without regard to any internal object partitioning, the signature scheme is called superimposed coding. Superimposed coding schemes cannot directly preserve the relative ordering of attributes within an object.

The types of queries we want to support will determine whether disjoint or superimposed coding should be used. A Partial Match Query (PMQ) is defined to be a query with one or more predicates on which an exact string match is desired. Predicates are always connected conjunctively (by a Boolean AND operator.) An example of PMQ is shown below.

```
PMQ:   SELECT * FROM T1
         WHERE
          (COMPANY  = 'IBM') AND
          (DIVISION = 'Research');
```

When the match constraints are relaxed and substring matches are allowed, the queries are called Partial Partial Match Queries (PPMQ.) An example of PPMQ is shown below. Data and query values are regarded as strings, i.e. domains are handled as if they were of type CHAR and VARCHAR(). The "LIKE" operator used with the '%' wildcard symbol designate a substring match operation.

```
PPMQ:  SELECT * FROM T1
         WHERE
          (TITLE LIKE '%Engineer') AND
          (EMPLOYEE LIKE '%Walter');
```

PPMQs will always occur when text is a field type consisting of sentences or words. In this paper, we do not address using signatures to solve queries where the predicates may be connected by the logical OR operator or where predicates may be preceded by a logical NOT operator. These functions are not efficiently supported by signatures, although some discussion has addressed these issues (see Dadam83.)

An important characteristic of both PMQ and PPMQ is the concept of query weight. The query weight strongly influences the selectivity of a query signature. Let s_Q designate the signature of the query expression Q. The query weight is defined as the Hamming Weight (HW) of s_Q, i.e. $QueryWeight(s_Q) = HW(signature(Q))$ or simply the number of bits set to 1 in the query signature. As more bits are set in the query signature, the signature becomes more selective in filtering data object signatures. This property is true regardless of whether one or more predicates are entered and independent of the type (PMQ or PPMQ). Thus, queries with higher weights will cause fewer data objects to be accessed and will have fewer false drops since all bits of the query signature are less likely to be covered by bits in the object signature.

2.2. Single Level Signature Files

The simplest structure for a signature file is a single-level organization. Tuple signatures and Tuple-IDs (TIDs) are stored in a file in fixed sized units of storage called *pages*. A signature is computed for a query expression and then compared sequentially against all tuple signatures. Candidate tuples are retrieved using the TID and then examined. Single-level signature organizations may be suitable for applications where the number of data objects is small (e.g., less than 5K.) An example of a signature file and sample query are shown in figure 1. In this example a single predicate of (EMPLOYEE LIKE 'Chang') would have a signature of 01000010 and would qualify the first and fourth tuples but would reject all other tuples shown. After accessing and inspecting the first and fourth tuples, only the first would match the query.

Early in our work we conducted a series of single-level signature file experiments using records from internal IBM corporate telephone directory files. The results are summarized in the table in figure 2. These results and those of (Deppisch86) show that signature selectivity is very high as a function of the query weight. For query weights greater than 3, over 98% of the objects were immediately rejected by the signature. Other experiments, all with similar behavior convinced us that signatures would be a promising, simple, and generic method to provide coverage over several or all fields of each database record.

In the table in figure 2, our test relation required 270 pages and with no pages in memory took 45 seconds to scan on an IBM RT/PC workstation. A regular index over the LASTNAME field required 58 pages while a one-level signature file required 61 pages. This data indicated that access time was reduced by a factor of 2 to over a factor of 10. Combined signature/relation page IOs can be reduced by a factor of 2.5 when compared to a pure relation scan which would require all tuples on all 270 relation pages to be accessed. While this is a major improvement, a normal index still gives better search performance over signatures. However, one index file is required for each different key-column, while one signature file provides coverage for all columns of the relation. This

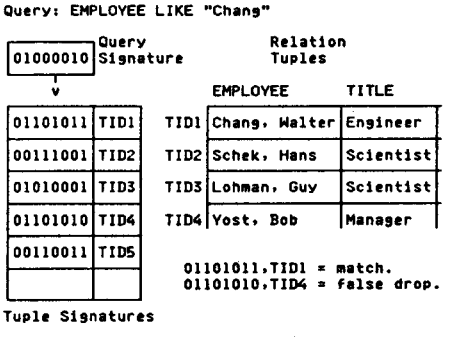

Figure 1: A Single Level Signature File.

has a significant impact on updates.

Relation contains 10000 tuples, Pages are 4KB.

Query Weight	Tuples Rejected by S1	Tuples Accessed by S1	False Drops	Matches	Time (Sec.)
1	7513.1	2486.8	2343.6	143.3	27.9
2	8051.9	1948.2	1503.7	444.5	21.4
3	9201.9	798.1	519.0	279.2	11.8
4	9860.8	139.2	136.4	2.8	4.8
5	9956.9	43.1	40.7	2.5	3.9
6	9988.6	11.4	10.3	1.1	3.4
7	9993.6	6.3	5.3	1.0	3.3
8	9998.4	1.6	0.6	1.0	3.0
9	9998.8	1.2	0.2	1.0	3.2
10	9999.0	1.0	0.0	1.0	3.1

Figure 2: Performance of A Single Level Signature File

To compare the search and update effort, we estimated the number of page accesses needed when a signature was used and when an index was used. We assumed each record contained F total fields and that a query specified f fields. For the signature access path, all 61 signature pages must be inspected during a query, independent of f. In case of the index, we take every value of the query and follow the path from the root to the leaf page where we fetch the tuple addresses (TIDs.) If each index contains l levels, we need $f \times l$ index page accesses.

Assume the number of different items to be put into an index is 10000 and the average number of different items within one record is 10. We utilize a B-tree for the items with 3 levels under these assumptions. Further we assume that the address list or TID-list of any item fits into one page. For many practical queries, $f=3$ or $f=4$ and therefore 9 to 12 index pages are accessed during a search as opposed to 61 in the signature case.

For insertion or deletion using the signature method one page access is needed. For indexes, we need at least as many (leaf) page accesses as we have values in the record, in our example $F=10$. If any of our index pages split, non-leaf pages must also be accessed. Thus, if the relation is fully indexed, each insert will require us to touch typically $F \times l$ (30 pages in our example). This discussion shows that signatures behave well in a dynamic environment and require substantially less update activity than full or even partial inversion schemes. We will show in the following discussion that multi-level signatures, if applied carefully, can meet this requirement.

2.3. Multi-Level Signature Organizations

In the previous example, if N = the number of tuples, a single-level signature file requires N comparisons between the query signature and the tuple signatures. The complexity is linear, and if N is large, we want to avoid the sequential signature scan, even if a single signature test is efficient.

One method of avoiding the linear scan is to cluster signatures together and create a group signature by superimposing or bit-ORing the individual signatures. A query signature is first compared to the group signature and if the query signature does not completely cover the group signature, the entire group can be discarded without further inspection. If the test with the group signature is positive, the query signature is tested against the individual signatures of the group. Shown below in figure 3 is a partial diagram of a multi-level signature structure.

Different multi-level signature organizations have already been investigated (Roberts79, Pfaltz80, Deppisch86, Sacks-Davis87). In this paper, B-tree terminology will be used. A "leaf signature" is stored at the lowest level, a "parent signature" is stored at higher levels.

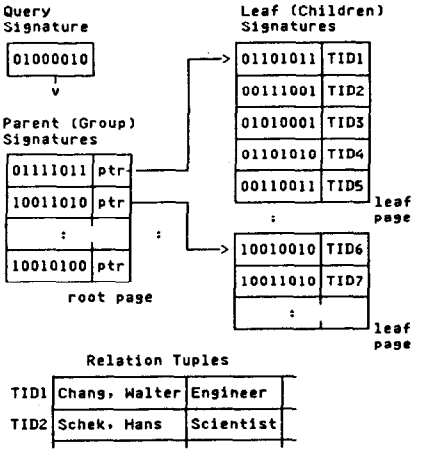

Figure 3: Multi-level signature file.

Each tuple signature and its TID is stored as a pair (SIG_i, TID_i) on leaf pages. The set of Signature-TID pairs on a leaf page forms a group. The parent signature of each group is denoted by s_P. Query signatures that reject the parent signature s_P reject all leaf signatures which belong in that group. In figure 3, since the bits of the query signature completely cover the bits of the first parent signature but not those of the second, the first leaf page would be searched but the second page would not. Bit-slice multi-level signature organizations exist but are not considered due to their significantly higher disk update costs (Sack-Davis87.) We now turn to the difficulties involved with multi-level signature organizations.

3. Problems with Multi-level Signatures

There are two major problems associated with multi-level signatures. One is the density of set bits in the parent signatures. The second problem is the rate of "combinatorial errors" caused by a large class of queries. These are key problems because both cause query signatures to unnecessarily qualify more object signatures thereby increasing the false drop rate. While the first problem has received some treatment, e.g., (Deppisch86), the second problem has received little attention until recently (Sacks-Davis87.)

Problem 1: Density of Set-1's in Parent Signatures

As more signatures are added into a group, the group signature saturates to an all 1's bit vector. For a signature s, the density of s is defined as $HW(s) \div m$ where m is the length of the signature in bits. This saturation effect unfortunately happens quickly for even a small number of signatures when leaf signatures have the "optimal" density of 0.5. Let N_L be the number of signatures/leaf page, α_L be the probability that the i'th bit of a leaf signature bit is 1, and α_P be the probability that the i'th bit of a parent signature bit is 1. The probability that the i'th bit in the parent is set to 1 is obtained by computing first the probability that the i'th parent bit is 0. For this to be true, each of the i'th bits of all N_L signatures in the leaf cluster must be 0. This yields the well-known formula:

(1) $$\alpha_P = 1 - (1 - \alpha_L)^{N_L} \text{ , also}$$

(2) $$N_L = \left\lceil \frac{\ln(1 - \alpha_P)}{\ln(1 - \alpha_L)} \right\rceil$$

(3) $$\alpha_L = 1 - e^{\frac{1}{N_L}\ln(1-\alpha_P)}$$

From equation (1), for optimal object signatures with $\alpha_L = 0.5$, $\alpha_P = 0.938$ if the clustersize N_L is only 4. Experiments using 4-byte and 6-byte signatures confirmed these expectations. More advanced clustering techniques have been proposed and tested successfully by (Deppisch86) using S-Trees. Another solution to solve this problem is presented in (Sacks-Davis83) and uses large "segment" or block descriptors which are stored in a bit-slice organization. However, even if the parent level saturation problem is addressed, a second problem still remains.

Problem 2: The Combinatorial Error

The second serious problem is the combinatorial error. It appears in two seemingly different classes of signature applications: (1) conjunctive multi-predicate database queries consisting of PMQ and PPMQ, and (2) text search queries.

We consider the conjunctive multi-predicate case first. Assume we have N records with F fields each. Assume that we have already computed good selective signatures s_i for each of the N individual records. Such signatures either are obtained by a concatenation of the individual field signatures s_{ij} as in formula (4), or by superimposing the individual signature patterns for the $j'th$ field values of a record, as in formula (5).

(4) $$s_i = s_{i1} \mid s_{i2} \mid \ldots \mid s_{iF}$$

(5) $$s_i = s_{i1} \cup s_{i2} \cup \ldots \cup s_{iF}$$

For the following discussion, we may even assume that the s_i are perfect, i.e., there will be no false drops for whatever PMQ is posed. In reality, this can be achieved by the use of long signatures. We will now consider the group signature s_P formed by superimposing all single signatures s_i. Thus,

$$s_P = s_1 \cup s_2 \cup \ldots \cup s_{N_L}$$

Due to superimposing, any s_P signature represents not only the given N records but also all records which may be obtained by any Cartesian Product of field value combinations from the given field values of the N records. This causes our second problem. As an example, consider the simplest case of two records which contain only two fields. Here R_i designates a record with two fields and V_{jk} designates the $k'th$ value from the $j'th$ field.

$$R_1 = (V_{11}, V_{21}) , \qquad R_2 = (V_{12}, V_{22})$$

The group signature here would represent not only R_1 and R_2, but also records with combinations of these fields:

$$R_3 = (V_{11}, V_{22}) , \qquad R_4 = (V_{12}, V_{21})$$

With F fields, the group signature represents $(\mu_1 \times \mu_2 \times \ldots \times \mu_F)$ records where μ_j is the number of distinct V_{jk}'s or cardinality of the $j'th$ field. This product has a drastic influence on the "combinatorial error" which occurs when a query requests a record by specifying f fields. In the simple example above, by specifying two fields, four queries are possible, but only two of these queries will find matches. The other two queries will be satisfied by the group signature s_P but will be false drops since the corresponding tuples are not in the database. More generally, when field values are unique within the N records, the probability P_{Match} for finding a matching tuple given a query in which all F fields are constrained by a predicate, where the $j'th$ predicate consists of any value in field j, is given by:

(6) $$P_{Match} = \frac{N}{\prod_{j=1}^{F} \mu_j}$$

Therefore, the risk P_{Cerror} of having a combinatorial error is given by the following:

(7) $$P_{Cerror} = 1 - P_{Match}$$

This gives the probability for a combinatorial error when each of the f field values $V_1, V_2, \ldots, V_f$ from the query occur individually in at least one of the N records. More precisely, we assume that for each predicate for which the query values are V_{jk}, $k = 1, 2, \ldots, \mu_j$, there exists a record in the group which has value V_{jk} in the $j'th$ field.

Let $q_1 \ldots q_f$ represent the list of f fields for which predicates are specified in the query. If μ_{q_j} is the cardinality of a field in the query and is less than N, duplicate values will be present in this field. The total number of distinct records, considering only the $q_1 \ldots q_f$ fields, is represented by $N_{q_1 \ldots q_f}$. This value is a function of the cardinality and distribution of V_{q_j}'s. By generalizing (6) and (7), the match and combinatorial error probability for a specific query with f predicates is now given by:

$$(8) \qquad P_{Match} = \frac{N_{q_1 \ldots q_f}}{\prod_{j=1}^{f} \mu_{q_i}}$$

$$(9) \qquad P_{Cerror} = 1 - P_{Match}$$

Since the query contains values from the $N_{q_1 \ldots q_f}$ records, the query signature will necessarily cover the group signature. Thus, formula (9) gives an estimate for the probability that the group does not contain a matching record due to the combinatorial error. From our experiments, even for small values of N and f, this error is considerable. Consider a simple example in which field values are unique, i.e., $\mu_j = N$. Equations (8) and (9) reduce to the two special cases:

$$P_{Match} = \frac{1}{N^{f-1}}, \qquad P_{Cerror} = 1 - \frac{1}{N^{f-1}}$$

Suppose, for example, that $N = 10$, $f = 2$, and let $\mu_1 = \mu_2 = N$. Since there are no duplicates, $N_{q_1 \ldots q_f}$ is the total number of distinct records when two fields are considered and, is just N. For a query Q, the probability that the group of 10 records does not contain a match is already 0.9. As f increases, the combinatorial error also increases. If $f = 3$ and $\mu_3 = N$, the probability that we do not have a match is already 0.99.

We now turn to the case of text. Substantial work has been done in order to find good signatures for text based on statistics and information theory (Schek78, Christodoulakis84, Faloutsos85a,85b,87). Since text is a set or sequence of words and a query is also a set or sequence of words, there is still a considerable combinatorial error problem. Consider the following text consisting of two phrases as an example:

```
t = <<database index extension>,
    <operating system kernel>>
```

Assume that a signature s has been assigned to t and that a query asks for all documents with the phrase < database kernal >. The signatures would indicate a match even though the query did not match the actual text. Text signatures built on a word basis represent not only the original text phrase. They also represent every phrase which could be formed by any combination of the words in the original phrase. Thus, we have to deal with the same combinatorial problem.

The probability for this error increases as more phrases in the text are clustered to compute s and as more words are specified in the query. Formula (9), shows this to be a problem with group signatures with only a few phrases and a two- or three-word query.

While various aspects of the 1's bit saturation problem have been addressed in the literature, the combinatorial error problem remains. (Schek78) has examined estimating frequent pairs or triples of words and more recently, (Sacks-Davis87) has proposed multi-level signatures using larger parent signatures and setting bits based on common word-pairs to address one aspect of this combinatorial problem.

4. A Solution to the Combinatorial Problem

In this section, we describe a general approach to building second level signatures that exhibit a significantly reduced false drop sensitivity due to the combinatorial error.

The Combinatorial Signature

We propose a new signature function that encodes combinations of values within a record rather than single values from a record. This is equivalent to computing a new signature based on a combination of multiple bits in the original object signature. First, a leaf signature S1 with m_1 bits is computed for each record. Techniques for finding the length and hash function for S1 are well-known (see Faloutsos87).

In our new technique, for each S1 leaf signature a larger *combinatorial* signature CS1 of m_{CS1} bits is computed. CS1 is formed by testing a complete set or subset of all r pairs of bit positions in S1 and setting one of the m_{CS1} bit positions in CS1 to 1 if 1's appear at both bit positions indicated in S1. This could be extended to bit triplets, quadruplets, etc. For now, we consider only bit-tuples consisting of bit pairs.

To form a group (parent) signature for a set of N_L leaf signatures, all of the CS1's are superimposed (bit-OR'ed). Assume signatures are computed for records R_1 and R_2 and that each field value V_{jk} sets one S1 signature bit when it is hashed. When a combinatorial signature CS1 is computed by selecting pairs of bits in S1, each CS1 bit indicates the presence of a combination of record attributes. If a query is posed which specifies combinations of values (such as (V_{11}, V_{22}) or (V_{12}, V_{21})), a conventional S2 signature formed by bit-ORing leaf-level S1 signatures will not reject S1 signature groups. This query is said to contain a *combinatorial error*. However, an CS2 signature formed by bit-ORing CS1 signatures will indicate the presence of combinations of values and have a high probability of rejecting lower level leaf signature groups. An example of how the new combinatorial signature CS1 works is shown in figure 4.

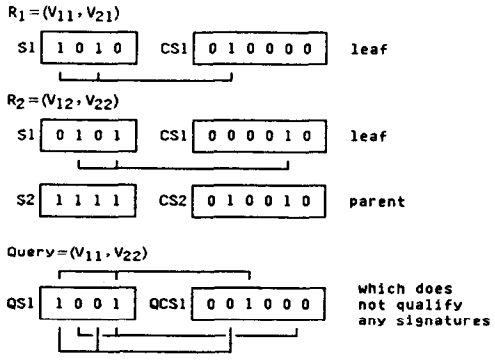

Figure 4: Rejection of query with combinatorial error

When a new object signature is incrementally added into the group covered by an CS2 signature, we compute the new object's CS1 signature from its S1 signature and then bit-OR the CS1 signature into the parent CS2.

The basic strategy for generating the combinatorial signature CS1 is to encode a full or partial subset of all combinations of S1 bit pairs. To compute CS1, we first identify a set of unique bit position pairs in S1. Each of these S1 bit pairs is bit-ANDed and each result is mapped to a specific bit position in CS1. When this is done systematically, the m_1 bits of S1 will form $m_1 - 1$ bit partitions in CS1 of size $(m_1 - 1)$, $(m_1 - 2)$,..., 1. For example: Bit 1 of CS1 is set by bit-ANDing bits 1 and 2 of S1. Bit 2 of CS1 is set by bit-ANDing bits 1 and 3 of S1. This is shown below in Figure 5.

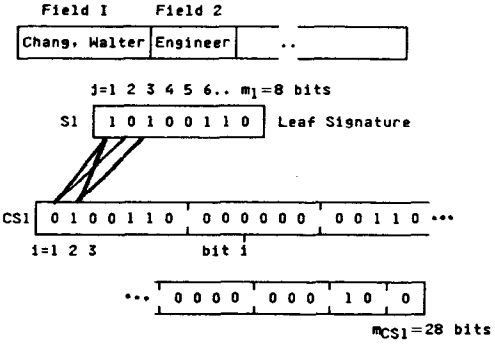

Figure 5: Computation of the combinatorial signature.

Using this method, we can determine the value of m_{CSI} for the required length of CS1. CS1 is partitioned into a set of bit fields of length $(m_1 - 1)$, $(m_1 - 2)$, ..., $(m_1 - (m_1 - 1))$. To find the total number of bits required for CS1, we sum the number of bits of all of the partitions:

$$(10) \qquad m_{CSI} = ((m_1 - 1)m_1) - \sum_{i=1}^{m_1 - 1} i, \quad \text{which gives:}$$

$$(11) \qquad m_{CSI} = m_1 \frac{(m_1 - 1)}{2}$$

Beyond a combinatorial signature, we can also use a second parent signature LS2 which is obtained by longer leaf signatures LS1 with m_{LSI} bits using other known methods (Sacks-Davis87). These longer LS1 signatures and the new combinatorial CS1 signature associated with S1 do not have to be stored in the leaf levels of the signature file since they are used only to build higher level signatures. In this sense, LS1 and CS1 signatures are "virtual".

Whether bit position pairs in S1 are selected at random or in some systematic way, for each CS1 bit, there is a small probability that selected S1 bit positions represent k-tuples from the same field. If this is the case, our new CS1 signature will not provide protection against combinatorial errors. Fortunately, this probability is low. If each record consists of F fields and we assume the signature encoding of the i'th field sets b_i signature bits in S1, the probability that a bit in CS1 was set by u S1 bits from the same record field i is given by:

$$P_{Same} = \left(\frac{b_i}{m_1}\right)^u$$

where $u = 2$ for bit pairs. For example, if a record consists of $F = 8$ fields and yields a signature S1 with $m_1 = 64$ total bits, and an average field requires $b_i = 8$ signature bits, $P_{same} = 0.016$. We expect this amount to be quite tolerable.

Thus, the key parameters in applying this technique are the selection of the CS1 generating algorithm, and an appropriate S1 signature. The selected density of S1 is crucial because it will determine the density in CS1 and the number of CS1 signatures which may be clustered to form CS2. We now address how signature parameters can be determined.

Determining Signature Densities

For each key to be stored in the index, we compute signatures S1, LS1, and CS1 from our data object, a record. S1 is a simple encoding using the algorithm given by (Harrison71) where we set k = 3 and use k-tuples within fields of data objects. Leading and trailing blanks in the fields can be ignored. Our first goal will be to design signature functions such that parent signatures attain the desired density when they are formed by superimposing the N_L lower level signatures.

This can be achieved by using a general heuristic for determining the lengths, density, and hash functions for the signatures in our file. In Figure 6, only S1 signatures are stored in leaf pages, while longer LS2 and CS2 signatures are stored only at parent levels. When a group of virtual LS1 leaf signatures are superimposed, a parent signature LS2 of the same length is formed. CS2 is formed in a like way. The higher level signatures LS3 and CS3 are formed by superimposing groups of LS2 and CS2 signatures, respectively. We next discuss the LS1 virtual signature.

Determining Signature Lengths

To select the length m_{LSI} for a virtual leaf signature LS1, an estimate is made of the number of unique k-tuples contained in a record, since this determines the likely number of 1's set in the signature. Since a k-tuple is simply a substring sequence of k bytes, as k increases, the number of distinct k-tuples within the record decreases while for smaller k, k-tuples are more likely to be repeated. To simplify our design, we set k to 3 and assume a nearly uniform distribution of k-tuples within each record. For example, if the Harrison algorithm is used, an estimate is made for AVG_{kt}, the average number of unique k-tuples per record using equation (12). Equation (13) gives the leaf signature density α_{LSI}, given the target parent density α_{LS2} and leaf clustersize N_L derived from (3).

$$(12) \qquad AVG_{kt} = \left[\frac{Len(Record) - (k-d)}{d} \right]$$

$$(13) \qquad m_{LSI} = \frac{AVG_{kt}}{\alpha_{LSI}}$$

The determination of design parameters for signature CS1 is trickier. Since CS2 is formed by superimposing CS1's and CS1 is derived from S1, we work backwards. We first need the density α_{CSI} for CS1 given N_L and a selected parent target density of α_{CS2} (typically .5 or less.) Given the density that must occur in a parent signature CS2, α_{CSI} for a virtual leaf combinatorial signature CS1 can be directly determined by using equation (3). Since bits of CS1 are set by bit-ANDing pairs of bits in S1, the density α_1 for S1 can be derived from the product:

$$(14) \qquad \alpha_{CSI} = \alpha_1^2.$$

To determine the length m_1 of S1, we again use an estimate of the number of unique k-tuples present in each record and divide by α_1. By altering the k-tuple size and distance d between k-tuples, a high degree of flexibility is possible in deciding m_1. When $k = 2$ and $d = 1$, we have the simplest case of Harrison's algorithm. If k and d are made variable and k-tuples become equivalent to entire words, for large m_1 the signature generalizes into a Bloom filter. The final step is to determine the length m_{CSI} of signature CS1. This is done by using equation (11).

5. Incorporating Multi-Level Signatures in Starburst B-Trees

We now describe a generic organization for multi-level signature files using B-tree structures. Other hierarchical organizations are possible, but are not discussed here. (Korth82), (Prabhakar83), and others have attempted to integrate signatures into B-trees. We now generalize these efforts. In our approach, one or more signature fields are appended to leaf and parent key entries within the B-tree index. Signature entries in interior pages filter entire pages of lower level signatures while leaf signatures filter specific data objects during predicate evaluation.

For all non-leaf pages of a B-tree, page entries consist of ordered < Dividing-key, Pointer > pairs used to determine which child page must be accessed next. To incorporate signatures, for each non-leaf entry we also store one or more parent signatures which represent the aggregation of all lower level signatures in the child pages. We selected two different signature functions for each parent entry and designate these by LS2 and CS2.

The motivation for multiple signatures at non-leaf levels is as follows: If different group signatures are used, there is a lower chance that a group containing all false drops will be qualified by all the signatures generated from a query. The signature functions must be sufficiently different in the way object attributes are captured. Examples of these signature functions are those for which the k-tuple size is increased or the distance d between k-tuples is increased (Harrison71).

In figure 6 we show a partially constructed signature file with the constituent signature elements. The functions key(R_i), tid(R_i) indicate extraction of the key fields and TID of data record R_i. S1 indicates a short signature for each data record, LS1 indicates the longer "virtual" signature of the same object, and CS1 is the combinatorial signature derived from S1. ParKey indicates a regular B-Tree Parent Key used to navigate to the leaf pages.

At the leaf pages of the B-tree, entries now consist of the record key fields, the short S1 signatures, and TID. For each table record R_i, an LS1 and CS1 is computed and used only to form the appropriate LS2's and CS2's at the parent levels. LS1 and CS1 signatures are shown on the right of the leaf pages in figure 6 and are used to compute parent signatures by a bit-ORing operation.

Higher Signature Levels

To form higher signatures, we can repeat the idea of superimposing lower level parent signatures until the root level is reached. One method is to simply Bit-OR lower level parent signatures to form higher level parent signatures. By using properly designed leaf, "virtual", and combinatorial signatures, saturation can be delayed significantly as we move to higher levels of the index. Alternative hashing methods are possible by varying the Harrison parameters (k, m, d) or by using a class of universal hashing functions at the different levels of the index (Carter80).

Many of the concurrency problems and recovery issues which apply to index managers will also apply to our new B-tree signature files as well. The basic multi-record locking algorithms developed for the Starburst index component are extended for signature indexes, and are similar to the algorithms described in (Bayer77, Mohan89). Due to space constraints, we defer presenting the implementation details of the al-

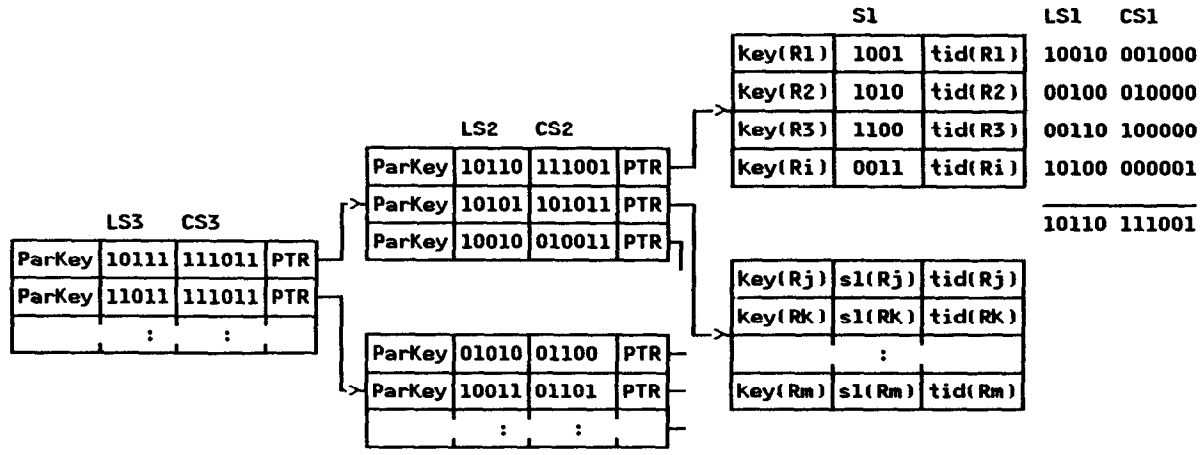

Figure 6: Multi-Level Signature Index incorporating S1, LS1, and CS1

gorithms for signature insert, delete, fetch, and scan operations as well as index locking and recovery to a future research report. We now address how the query optimizer would use a signature access method.

6. Starburst Query Optimizer Issues

For the query optimizer to decide whether or not to use an existing signature, it must know which relation columns are covered by the signature and then estimate the cardinality of field values, selectivity of a given query, and cost of using the signature file. To the optimizer, the signature access method or *attachment* has the same functional interface as other attachments in Starburst for tuple insert, delete, fetch, and scan (see Stonebraker80 and Lindsay87). Thus, the signature attachment will appear to the Starburst rule-based query optimizer as a scan LOLEPOP (Low-Level Plan Operator), described in (Haas88 and Lohman88).

Since the signature attachment will return an answer superset rather than a precise tuple set like a conventional index, a FILTER operator must be applied to eliminate tuples which may be false drops. The estimated cost of using a signature is based upon column cardinality, which is estimated by using a model of the probability any tuple will be accessed (selectivity), and a probabilistic model of page accesses.

The signature attachment could also be generalized for use in plans to solve text search predicates. If Harrison's k-tuples are set to be entire keywords in a document or a long field (Lehman89), than each $k-tuple$ represents a text keyword and precisely one bit per keyword will be set in signatures S1 and LS1. A signature index over text stored in a long field would be classified as an *exogenous* database attachment (Schwarz86) since the index may be bound to a long field, rather than a base table.

Our experimental results show that for conjunctive multi-predicate queries where f is large, signatures compete well with indexes. For queries with weights corresponding to two or more predicates (i.e., when the query weight exceeded 9), measurements showed that over 97% of the relation tuples were filtered and less than 20% of the signature file was accessed. Compared to relation scans, total page accesses (measured by page fixes) were reduced by over 92%. These preliminary tests indicate that signature files are a promising method for greatly accelerating relation scan performance for queries when normal indexes are not available.

7. Conclusions

We have identified an undesirable, naturally occurring false drop phenomena inherent to all multi-level signature structures which we have called the combinatorial error. We have presented a new combinatorial signature generation method which significantly reduces the negative effect of the combinatorial error. We have shown how to generically build multi-level signature structures from single level signature files by extending generic B-tree indexes such as the Starburst index component. Signature files can serve as a powerful extension access method to support not only database queries usually handled by relation or index scans, but also text applications.

Acknowledgments

We would like to express our thanks to the members of the Starburst project who provided us with many comments and suggestions during our work. We would also like to thank Uwe Deppisch at the Technical University of Darmstadt for his S-Tree experiments using our test data. Finally, we would like to thank Laura Haas, Guy Lohman, Pat Selinger, Bill Cody, and Bob Yost for helping to review this paper.

REFERENCES

[Bayer77] R. Bayer, M. Schkolnick, *Concurrency of Operations on B-Trees,* Acta Informatica 9, 1-21, (1977) (Springer-Verlag, 1977).

[Carter80] L. J. Carter, M. N. Wegman, *Universal Classes of Hash Functions,* IBM Research Report RC 6687 (#28796), IBM Thomas J. Watson Research Center (1977).

[Christodoulakis84] S. Christodoulakis, C. Faloutsos, *Design Considerations for a Message File Server,* IEEE Transact. on Software Engineering, Vol. SE-10, No. 2. (March 1984).

[Dadam83] P. Dadam, P. Pistor, H. J. Schek,. *A Predicate Oriented Locking Approach For Integrated Information Systems,* Information Processing 83: Proceedings of the IFIP 9th World Computer Congress (Paris, France, September 1983).

[Deppisch86] U. Deppisch, *S-Tree: A Dynamic Balanced Signature Index for Office Retrieval,* Proceedings of the 1986 ACM Conference "Research and Development in Informational Retrieval" (Pisa, Italy, September 1986).

[Faloutsos85a] C. Faloutsos, *Signature Files: Design and Performance Comparison of some Signature Extraction Methods,* Proceedings of SIGMOD, pp. 63-82. (1985).

[Faloutsos85b] C. Faloutsos, *Design of a Signature File Method that Accounts for Non-Uniform Occurence and Query Frequencies,* Proceedings of VLDB, pp. 165-170. (1985).

[Faloutsos87] C. Faloutsos, S. Christodoulakis, *Optimal Signature Extraction and Information Loss,* ACM Transactions on Database Systems, Vol.12, No.3 Sept. 87. (1987).

[Haas88] L. M. Haas, W. F. Cody, S. Finkelstein, J. C. Freytag, G. Lapis, B. Lindsay, G. Lohman, K. Ono, H. Pirahesh, *An Extensible Processor for an Extended Relational Query Language,* IBM Research Report RJ 6182, IBM Almaden Research Center, San Jose, CA. (1988).

[Harrison71] M. C. Harrison, *Implementation of the Substring Test by Hashing,* Communications of the ACM Vol 14, No. 21. (December 1971).

[Korth82] R. P. King, H. F. Korth, B. E. Willner, *Design of a Document Filing and Retrieval Service,* IBM Research Report RC 9696 (#42815) 11-18-82, Thomas J. Watson Research Center (1982).

[Lehman89] T. Lehman, B. Lindsay, *The Starburst Long-Field Manager,* 15th Proceedings of VLDB, 1989.

[Lindsay 87] B. Lindsay, J. McPherson, H. Pirahesh, *A Data Management Extension Architecture,* Proceedings of Association for Computing Machinery Special Interest Group on Management of Data, 1987 Annual Conference, May 27-29 (1987).

[Lohman88] G. Lohman, *Grammar-like Functional Rules for Representing Query Optimization Alternatives,* Proceedings of SIGMOD 1988.

[Mohan89] C. Mohan, F. Levine, *Aries-IX: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging,* IBM Research Report (In preparation), IBM Almaden Research Center (1989).

[Pfaltz80] J. L. Pfaltz, W. J. Berman, E. M. Cagley, *Partial-Match Retrieval Using Indexed Descriptor Files,* Communications of the ACM Vol. 23, No. 9 (September 1980).

[Prabhakar83] T. V. Prabhakar, H. V. Sahasrabuddhe, *Signature Trees - A Data Structure for Index Organization,* International Conference on Systems, Man, and Cybernetics Proceedings 1983 vol. 2 (1983).

[Roberts79] C. S. Roberts, *Partial-Match Retrieval via the Method of Superimposed Codes,* Proceedings of the IEEE Vol. 67, No. 12 (December 1979).

[Sacks-Davis83] R. Sacks-Davis, K. Ramamohanarao, *A Two Level Superimposed Coding Scheme For Partial Match Retrieval,* Information Systems Vol. 8, No. 4 (1983).

[Sacks-Davis87] R. Sacks-Davis, A. Kent, K. Ramamohanarao, *Multikey Access Methods Based on Superimposed Coding Techniques,* ACM Transactions on Database Systems Vol. 12, No. 4, December 1987 (1987).

[Schek78] H. J. Schek, *The Reference String Access Method and Partial Match Retrieval,* Proc. Information System Methodology 1978, Vol. 65., G. Bracchi, P. C. Lockemann, also see: IBM Scientific Center Report TR 77.12.008 (1978).

[Schwarz86] P. Schwarz, W. Chang, et. al., *Extensibility in the Starburst Database System,* IBM Research Report RJ 5311 (#54671) 09-23-86 (1986).

[Severance76] Severance, D. G., Lohman, G. M., *Differential Files: Their Application to the Maintenance of Large Databases,* ACM Transactions in Database Systems Vol. 1, No. 3 (September 1976)

[Stonebraker80] Stonebraker, M, *Retrospection on a Database System,* ACM Transactions in Database Systems Vol. 5, No. 2 (June 1980)

[Wilms88] P. Wilms, P. Schwarz, H. Schek, L. Haas, *Incorporating Data Types in an Extensible Database Architecture,* 3rd International Conference on Data and Knowledge Bases, June 28-30, 1988. Jerusalem, Israel (1988).