Key-Based Partitioned Bit-Sliced Signature File

Pavel Zezula*
IEI-CNR Pisa
Italy
zezula@iei.pi.cnr.it

Paolo Ciaccia and Paolo Tiberio
DEIS - CIOC-CNR - University of Bologna
Italy
{pciaccia,ptiberio}@deis.unibo.it

Abstract

A new signature file organization is proposed as a combination of two orthogonal partitioning strategies, the key-based and the bit-sliced, respectively. The design results from theoretical analysis of these elementary approaches in which performance is analytically studied respecting a simplified abstract storage structure model. The new organization is able to achieve very high search performance for queries containing arbitrary number of query terms – bit-sliced (key-based) organization is good only for queries containing few (many) terms, quite bad performance is obtained in the other cases. Update performance is also discussed and a generalization of the method, able to adjust the trade-off between the search efficiency and the maintenance costs, is put forward for consideration. The proposal is also compared with similar approaches in the field of signature files.

1 Introduction

An understanding of the advantages and disadvantages of storage structures is an important skill to know for everybody trying to develop or apply computer software aiming at processing large collections of data. The reason is that there is not a single *all-purpose* storage structure, and a unique situation requires application of a specific technique. The choice of a proper structure is particularly difficult when building general purpose systems (i.e., systems able to process data in many diverse ways), thus discovering new, more general, and performance stable storage structures still remains as an important research challenge. One way of approaching this problem is to combine two or more existing strategies into a unique technique that would share advantages of its constituents.

Another fundamental concept of the information storage technology suggests to convert unstructured, complex, or too bulky information into a form that is more readily processible by computers. Retrieval is then done on the new (typically much shorter and better structured) file, and in this way, improvements in performance are obtained. Typical storage structure representatives of this kind are known as the *Inverted File* [1] and the *Signature File Access Method* [4, 12, 13]. In the latter approach, data objects are converted into bit vectors, *signatures*, and a collection of such vectors, called *signature file*, is used for retrieval.

*On leave from the Technical University, Brno, Czech Republic.

Early signature file implementations, based on *sequential* organization, could not compete in retrieval performance with inverted files; this was mainly true for large files [9]. Consequently, many new techniques have been developed for the implementation of signature files [4, 13], thus by applying the proper technique for a task at hand, required performance can usually be achieved.

The number of suggested signature file implementations is really impressive – surveys of the topic can be found in [4, 13]. Their performance significantly depends on the number of conjunctive terms in a query. Usually, organizations working very well for queries with few terms (i.e., *low weight queries*) are not suitable for queries containing many terms (i.e., *high weight queries*). This is also true in the opposite direction, thus organizations suitable for executing queries of high weights do not perform well for low weight queries. Typical representatives of signature file organizations possessing these properties, are the *bit-sliced* [10] organization – files organized as bit slices achieve high performance for low weight queries – and the *key-based partitioned* approach [5, 17], which is convenient to use when high weight queries are processed.

In this paper, we propose to combine these two approaches in order to obtain a signature file organization that is appropriate for any kind of query. We start in Section 2 by formally defining partitioning schemes of a signature file and proceed by specifying different query evaluation strategies in Section 3. In Section 4, we define an abstract storage system and derive formulas able to determine retrieval costs of three well known storage organizations. The proposal and performance evaluation of a new signature file organization is presented in Section 5.

2 Background

Even though more than one signature extraction method have been proposed in literature, *superimposed coding*, see e.g. [4], is by far the most frequent way of producing signatures – due to its simple matrix structure, a file of superimposed signatures offers a lot of possibilities for storage organization. Accordingly, whenever we talk in this article about a signature, we mean a signature produced by the superimposed coding method.

A *signature file* $\mathbf{S}$ is a set of n *signatures*, where $\mathbf{S}_i$ $(i = 1, \ldots, n)$ is a binary string of f bits representing the information content of *data object* O_i. Each object is supposed to contain T indexable *terms*, such as *attribute values* or *keywords*, and signature $\mathbf{S}_i$ is produced by *superimposing* (i.e., inclusive OR-ing) T *term signatures* of O_i. The signature of a term is simply obtained by hashing the term onto an f-bit zero valued vector so that bits in v, not necessarily distinct, positions are set to "1".

Optimum signatures (i.e., signatures balancing the trade-off between the search efficiency and the space occupancy) [4], have 50% of bits with value "1"and 50% of bits with value "0". The number of bits in signature $\mathbf{S}_i$ is called the signature *weight* and is designated as $w(\mathbf{S}_i)$. That means that the optimum signatures have $w(\mathbf{S}_i) = f/2$, on the average.

In a formal way, a file of superimposed signatures, $\mathbf{S}$, can be seen as a matrix of bits

$$\mathbf{S} = \left\| \begin{array}{cccc} s_{1,1}, & s_{1,2}, & \ldots & s_{1,f} \\ s_{2,1}, & s_{2,2}, & \ldots & s_{2,f} \\ \vdots & \vdots & \vdots & \vdots \\ s_{n,1}, & s_{n,2}, & \ldots & s_{n,f} \end{array} \right\|$$

where $s_{i,j} \in \{0,1\}$, for $i \in \{1,2,\ldots,n\}$ and $j \in \{1,2,\ldots,f\}$, are bits constituting a file of n signatures, where each signature has size f bits.

There are several ways how a matrix can be split into its parts. Simple horizontal (i.e., the *bit-string*) and vertical (i.e., the *bit-sliced*) splitting strategies can be defined as follows. In this work we do not consider other partitioning strategies, such as the horizontal splitting method described in [11].

Bit-string splitting:

$$\mathbf{S} = \left\| \begin{matrix} \mathbf{S}_1 \\ \mathbf{S}_2 \\ \vdots \\ \mathbf{S}_n \end{matrix} \right\| \quad where \quad \begin{matrix} \mathbf{S}_1 & = & [s_{1,1}, & s_{1,2}, & \ldots & s_{1,f}] \\ \mathbf{S}_2 & = & [s_{2,1}, & s_{2,2}, & \ldots & s_{2,f}] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{S}_n & = & [s_{n,1}, & s_{n,2}, & \ldots & s_{n,f}] \end{matrix}$$

and $\mathbf{S}_i$ is the i-th signature of the file $\mathbf{S}$. In the following, whenever a confusion might occur, we call $\mathbf{S}_i$ an *object signature* since it represents the content of the i-th data object.

Bit-sliced splitting:

$$\mathbf{S} = \|\mathbf{S}^1, \mathbf{S}^2, \ldots, \mathbf{S}^f\| \quad where \quad \begin{matrix} \mathbf{S}^1 & = & [s_{1,1}, & s_{2,1}, & \ldots & s_{n,1}]^T \\ \mathbf{S}^2 & = & [s_{1,2}, & s_{2,2}, & \ldots & s_{n,2}]^T \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{S}^f & = & [s_{1,f}, & s_{2,f}, & \ldots & s_{n,f}]^T \end{matrix}$$

and $\mathbf{S}^j$ is the j-th bit-slice (i.e., the j-th column) of the file $\mathbf{S}$.

Frame-sliced splitting:

A more complicated way of splitting a signature file suggests grouping bit slices into separable units to form the *frame-sliced* structure defined as follows. Let $x > 0$, called the *frame-slice size*, be the number of consecutive bit-slices forming a *frame-slice*, where $x \cdot y = f$ for a specific $y \in \{1,2,\ldots,f\}$. The precise meaning of frame-sliced splitting can be stated as

$$\mathbf{S} = \|\mathbf{F}^1, \mathbf{F}^2, \ldots, \mathbf{F}^{f/x}\| \quad where \quad \begin{matrix} \mathbf{F}^1 & = & [\mathbf{S}^1, & \mathbf{S}^2, & \ldots & \mathbf{S}^x] \\ \mathbf{F}^2 & = & [\mathbf{S}^{x+1}, & \mathbf{S}^{x+2}, & \ldots & \mathbf{S}^{2x}] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{F}^{f/x} & = & [\mathbf{S}^{f-x+1}, & \mathbf{S}^{f-x+2}, & \ldots & \mathbf{S}^f] \end{matrix}$$

Observe that our definition does not correspond to the Frame-Sliced Signature File organization from [7], which uses a different method for generating signatures and processing queries.

Key-based splitting:

By considering parts of signatures as *keys*, signature $\mathbf{S}_i$ can be split into its *key* part, $\mathbf{K}_i$, and the *body*, $\mathbf{S}_i^{(b)}$, containing the rest of the signature. Provided keys are defined as k-bit signature prefixes, we can view splitting of the signature $\mathbf{S}_i$ as

$$\mathbf{S}_i = \mathbf{K}_i, \mathbf{S}_i^{(b)}$$

where $\mathbf{K}_i = [s_{i,1}, s_{i,2}, \ldots, s_{i,k}]$ and $\mathbf{S}_i^{(b)} = [s_{i,k+1}, s_{i,k+2}, \ldots, s_{i,f}]$ for all i and $k \in \{1,2,\ldots,f\}$.

3 Querying in Signature Files

Signature file queries, defined as conjunctions of terms, are transformed into a signature form as well. A query signature, $\mathbf{Q}$, is then a row binary vector $\mathbf{Q} = [q_1, q_2, \ldots, q_f]$, where $q_j \in \{0, 1\}$ for $j = 1, 2, \ldots, f$, and the first k bits form the query signature key, $\mathbf{K_Q}$. Obviously, the query signature weight, $w(\mathbf{Q})$, should not exceed the weight of object signatures, i.e. $w(\mathbf{S}_i) \geq w(\mathbf{Q})$, and can become quite small, e.g. 8 or 10, compared to $w(\mathbf{S}_i)$, which is typically counted in hundreds.

The task of a signature file query evaluation is to find all signatures in $\mathbf{S}$ which *qualify* for the query signature $\mathbf{Q}$. Such set of signatures, designated as the *query response*, $R_\mathbf{Q}$, can be defined in several different manners.

Response for query signature Q:

$$
\begin{aligned}
R_\mathbf{Q} &= \{\mathbf{S}_i | (\mathbf{S}_i \; AND \; \mathbf{Q}) = \mathbf{Q}\} && (1)\\
&= \{\mathbf{S}_i | \forall j : q_j = 1 \Rightarrow (s_{i,j} = 1)\} && (2)\\
&\subseteq \{\mathbf{S}_i | (\mathbf{K}_i \; AND \; \mathbf{K_Q}) = \mathbf{K_Q}\} && (3)
\end{aligned}
$$

Notice that AND in Equations 1 and 3 stands for the *bitwise* logical operator.

Equations 1, 2, and 3 suggest three ways to identify $R_\mathbf{Q}$. The first of them, Equation 1, uses as a test the traditional *inclusion condition* in which a query signature is supposed to be included in a qualifying object signature. In other words, the signature $\mathbf{S}_i$ qualifies if it contains ones in all positions in which the query signature $\mathbf{Q}$ contains bits with value "1".

Equation 2 is based on a different strategy. It takes advantage of the fact that bit positions determined in object signatures by "1"'s in the query signature are the only positions which must be considered in the query evaluation process – values of the remaining object signature bits are not able to influence the result of querying and need not be explicitly tested (accessed) for qualification.

Equation 3 is similar to Equation 1, but considers only key-parts of signatures, thus the response, as determined by Equation 3, forms in general a superset of the query response determined by Equations 1 and 2.

4 Signature File Organizations

The performance of any secondary storage access structure is a complex function of (a) the way how the entire data search space is allocated on the memory, (b) the search strategy used for retrieval, and (c) storage hardware parameters. In order to get a unique platform for comparing query response costs of different signature file organizations, we define an abstract storage system and claim usual assumptions of similar analytical models.

Let the *storage system*, $B_c = (b_1, b_2, \ldots, b_m)$, be formed by a set of m buckets, where the bucket b_l, $l = 1, 2, \ldots, m$, is a directly accessible continuous disk memory space of size $c \geq f$ bits. Then, the *query processing cost*, $C_\mathbf{Q}$, is the *cardinality* of the subset of B_c determined by buckets accessed while processing $\mathbf{Q}$. Notice, that we suppose no bucket to be accessed more than once.

In the following, we define three well known signature file allocation strategies (organizations) and establish analytical formulas for estimating $C_\mathbf{Q}$. The formulas are based on the following assumptions:

Symbol	Description
O_i	i-th object of the data file
T	number of terms specifying a data object
$\mathbf{S}$	signature file
n	signature file size (i.e., the number of signatures in $\mathbf{S}$)
$\mathbf{S}_i$	i-th signature of $\mathbf{S}$
$w(\mathbf{S}_i)$	i-th signature weight
f	signature size [bits]
v	number of bits set to "1" for a term
$\mathbf{Q}$	query signature
k	signature key size [bits]
x	frame-slice size [bits]
$\mathbf{S}^j$	j-th bit-slice
$\mathbf{F}^j$	j-th frame-slice
$\mathbf{K}_i$	i-th signature key
$\mathbf{K_Q}$	query signature key
B_c	system of storage buckets
m	number of buckets in B_c
b_l	l-th bucket of the storage system B_c
c	storage bucket capacity [bits]
α	bucket load ratio
$C_{\mathbf{Q}}^{XX}$	cost for processing Q in XX organization
SS	sequential signature file organization
BS	bit-sliced organization
KB	key-based partitioned organization
KS	key-based bit-sliced organization
KS^x	key-based frame-sliced organization
HS	hierarchical signature file

Table 1: Major symbols

1. bits with value "1" are uniformly distributed in signatures, and

2. signatures in $\mathbf{S}$ form random samples, i.e. they are independent of each other.

It is interesting to observe that each of the considered organizations complies with a different query response evaluation strategy (see Equations 1, 2, and 3). For convenience, used symbols and their meanings are summarized in Table 1.

4.1 Sequential File

In the *Sequential Signature* file, SS, the $(i+1)$-th element of the file $\mathbf{S}$ (i.e., the signature $\mathbf{S}_{i+1}$) is stored contiguous to and immediately after the i-th element of $\mathbf{S}$. Respecting our storage system architecture, each bucket, except for the last one, is filled with $\lfloor c/f \rfloor$ signatures where i-th signature of $\mathbf{S}$ is stored in the bucket b_l, $l = \lceil i/\lfloor c/f \rfloor \rceil$. To avoid waste of space, c should be a multiple of f.

For each query (or a set of queries), signatures of $\mathbf{S}$ are accessed in their order of occurrence until all signatures are processed, and the response for the query signature $\mathbf{Q}$ is determined according to Equation 1. The query processing cost $C_{\mathbf{Q}}^{SS}$ is constant for a specific file, because it does not depend on the query signature weight, and can be formalized as

$$C_{\mathbf{Q}}^{SS} = \lceil n/\lfloor c/f \rfloor \rceil \tag{4}$$

4.2 Bit-Sliced Organization

The *Bit-Sliced*, BS, signature file organization, proposed in [10], is a multiple sequential organization of the signature file bit-slices. Buckets contain parts of these bit-slices, and a specific bucket never comprises data of different slices. Each bucket, except for the last one of each slice, is filled with c bits. One bit slice occupies $\lceil n/c \rceil$ buckets, and i-th bit of the j-th slice (i.e., the bit $s_{i,j}$) is stored in the bucket b_l, $l = (j-1) \cdot \lceil n/c \rceil + \lceil i/c \rceil$. BS organization adopts for query evaluation the strategy defined by Equation 2, and the query processing cost, $C_{\mathbf{Q}}^{BS}$, can be determined as follows:

$$C_{\mathbf{Q}}^{BS} = w(\mathbf{Q}) \cdot \lceil n/c \rceil \tag{5}$$

Obviously, $C_{\mathbf{Q}}^{BS}$ depends on query signature weight – better performance is obtained when $w(\mathbf{Q})$ is low. Because $C_{\mathbf{Q}}^{BS}$ is only a function of $w(\mathbf{Q})$, n, and c, the query processing cost of BS is not directly dependent on the signature size f, though, for a specific query, $w(\mathbf{Q})$ is a function of f.

4.3 Key-Based Partitioned Organization

The *Key-Based Partitioned*, KB, organization stores in buckets signatures having identical keys [5] – key is defined as a part of a signature, e.g. its prefix or suffix. Since KB uses a *hashing* function to allocate signatures in buckets, buckets are typically not filled entirely, and the ratio of the occupied data space and the bucket capacity is, in general, expressed as $\alpha \leq 1$. Some KB organizations, e.g. Quick Filter [17], are even able to manage dynamic files. Each bucket, b_l, $l = 1, 2, \ldots, m$, is represented by a key $\mathbf{K}_l^{(B)}$ of size $k = \log_2 \lceil n/\lfloor \alpha \cdot c/f \rfloor \rceil$ and filled on average with $\lfloor \alpha \cdot c/f \rfloor$ signatures from $\mathbf{S}$, for which $\mathbf{K}_i = \mathbf{K}_l^{(B)}$. The number of buckets, m, depends on actual signature file size and can be determined as $m = \lceil n/\lfloor \alpha \cdot c/f \rfloor \rceil$.

Respecting the strategy suggested by Equation 3, the query processing cost of KB organization, $C_{\mathbf{Q}}^{KB}$, can be determined as follows:

$$C_{\mathbf{Q}}^{KB} = \left\lceil \frac{n}{\lfloor \alpha \cdot c/f \rfloor} \right\rceil \cdot \left(1 - \frac{w(\mathbf{Q})}{2f}\right)^{\log_2 \left\lceil \frac{n}{\lfloor \alpha \cdot c/f \rfloor} \right\rceil} = m \cdot \left(1 - \frac{w(\mathbf{Q})}{2f}\right)^{k} \tag{6}$$

where

$$\left(1 - \frac{w(\mathbf{Q})}{2f}\right)^{\log_2 \left\lceil \frac{n}{\lfloor \alpha \cdot c/f \rfloor} \right\rceil}$$

is a formula developed in [3] for estimating the *Bucket Activation Ratio* (i.e., the ratio of accessed to existing buckets) obtained when processing the query signature $\mathbf{Q}$.

Similar to BS, the query processing cost of KB is also a function of $w(\mathbf{Q})$, but the processing cost of KB decreases with increasing query signature weights. Notice that

Equation 6 is precisely valid for integer k (i.e., $2^k = m$). However, it also gives good approximations in the other cases.

4.4 Discussion

If we try to compare performance of SS, BS, and KB organizations (by applying Equations 4, 5, and 6, respectively) considering specific $\mathbf{S}$ and implementation platform B_c, we find out that there is not a clear winner, provided query signatures with different weights can occur as requests for information from the file. For the clarity of exposition, we use query signature weights in the following figures to relate the performance and query complexity, even though users specify queries using a certain number, $T_{\mathbf{Q}}$, of terms. The reason is that queries with the same number of terms can result in query signatures with different weights. The expected value can be estimated as follows:

$$E(w(\mathbf{Q})) = f \cdot \left(1 - \left(1 - \frac{1}{f}\right)^{v \cdot T_{\mathbf{Q}}}\right)$$

As Figure 1 illustrates, the bit-sliced organization is very good for low weight queries,

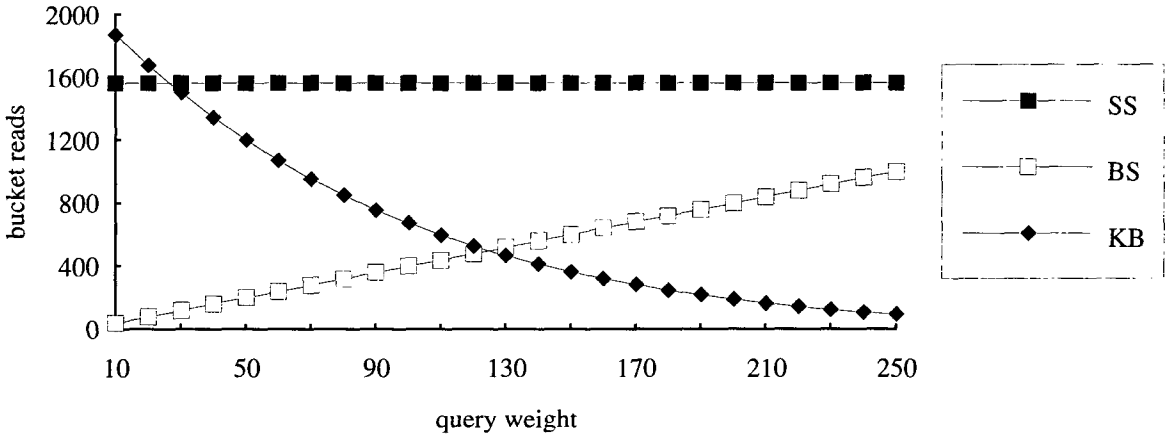

Figure 1: Performance comparison of SS, BS, and KB organizations; $n = 10^5$, $f = 512$, $c = 32768$ (4 Kbytes), $\alpha = 0.75$.

but its performance for high weight queries is much worse. On the other hand, the key-based partitioned method performs very well for high weight queries, though it is quite slow for queries resulting in signatures with low weights. The sequential organization has a stable performance (i.e., not dependent on specific query signature weight), but it may only be convenient for short files or in applications where high performance is not the primary concern.

Recently, several successful attempts have been made to improve search performance of signature files by exploiting parallelism [2, 6, 8, 16]. All of them even claim (nearly) *linear speedup* – the linear speedup is achieved when an organization using twice as much processors can perform the same task two times faster – for most of conceivable queries. Notice that, when linear speedup is achieved for a given organization, it is the maximum which a parallel architecture can provide to improve performance of a single processor organization at hand.

However, even such significant advancements in performance of signature file organizations, caused by proper utilization of parallelism, cannot change the performance characteristics of these organizations, i.e. the way how the response time is changing in relation to the query signature weight.

Provided queries of a specific application do not differ much in the query signature weight, choice of a convenient signature file organization is usually possible, see [12] for a guide. However, independent of the implementation environment, applications needing to process query signatures of very diverse weights find it difficult (or even impossible) to choose a convenient organization because they can always expect bad performance, at least for some queries.

5 Key-Based Bit-Sliced Organization

In order to overcome the above problem of performance instability in organizing and processing signature files, we conveniently combine the BS and KB approaches into a single design, capable of performing both high and low weight queries efficiently. The main idea is to group signatures with same keys, as it is done in the key-based partitioning. But, instead of putting all signatures of a group into a single bucket as in KB, f buckets are created, each of them filled just with a bit-slice of signatures from the group. We call the design the *Key-Based Bit-Sliced* organization, KS.

Response for the query signature $\mathbf{Q}$ with the KS organization can be formalized as follows:

$$R_{\mathbf{Q}} = \{\mathbf{S}_i | (\mathbf{K}_i \; AND \; \mathbf{K_Q} = \mathbf{K_Q}) \wedge \forall_j : q_j = 1 \Rightarrow (s_{i,j} = 1)\} \tag{7}$$

Obviously, by simultaneously pruning the signature file search space in its horizontal and vertical dimensions, the necessary number of tested bits is effectively reduced. Furthermore, whenever the efficiency of vertical pruning is decreasing due to increasing query signature weights, the horizontal pruning is automatically becoming more efficient for exactly the same reasons. By combining strategies offered by Equations 2 and 3, Equation 7 determines the same (i.e., the actual) response as Equations 1 and 2 do; notice that Equation 3 determines just a superset of this.

However, the power of horizontal pruning of KB and KS organizations is not the same due to the difference in signature key sizes which these organizations use – signature key sizes determine the power of horizontal pruning in query execution. Similar to KB organization, signatures in KS are also hashed into groups, thus buckets are only filled partially, which can also be expressed by the ratio α. Provided the bucket capacity c and the ratio α are given, KB and KS organizations should store specific $\mathbf{S}$ in the same number of buckets – a signature file is determined by the number of signatures n and the signature size f. However, while each bucket of KB organization is identified by a different key, f buckets share the same key in KS. This implies that the number of distinct key values in KB is practically f times higher.

As a result, the key size of KS, denoted $k(1)$, is constrained by $k(1) = \log_2\lceil n/\lfloor \alpha \cdot c \rfloor \rceil$, compared to $k = \log_2\lceil n/\lfloor \alpha \cdot c/f \rfloor \rceil$, which is true for the KB organization. Fortunately, the key of KS is obviously not f times shorter, because of the logarithmic dependence of the key size on the number of distinct key value. This is very important considering performance, because the key size has the primary effect on the bucket activation ratio.

On the other hand, the cost, which KS is forced to pay by decreased performance due to the shorter size of its signature key, is much compensated by using the bit-sliced access

strategy in the activated groups. The actual performance for a query signature Q can be estimated as follows:

$$C_{\mathbf{Q}}^{KS} = w(\mathbf{Q}) \cdot \left\lceil \frac{n}{\lfloor \alpha \cdot c \rfloor} \right\rceil \cdot \left(1 - \frac{w(\mathbf{Q})}{2f}\right)^{\log_2 \left\lceil \frac{n}{\lfloor \alpha \cdot c \rfloor} \right\rceil} = w(\mathbf{Q}) \cdot m(1) \cdot \left(1 - \frac{w(\mathbf{Q})}{2f}\right)^{k(1)} \quad (8)$$

where $m(1) = \lceil n/\lfloor \alpha \cdot c \rfloor \rceil$ is the number of distinct key values in the KS organization. Note that access to the buckets storing bit-slices which are part of the key is not necessary. For simplicity, this effect is ignored in the above formula. This does not introduce significant errors since the size of the key is typically is much lower than the signature size.

5.1 Evaluation of retrieval performance

Similar to KB organization, performance of KS is a function of the signature and the signature file sizes, bucket capacity and its load ratio, and the query signature weight. Before presenting results obtained from the analysis of Equation 8, defining retrieval performance of KS organization, we first provide some illustrative examples concerning the general behavior of KS.

Figure 2 shows the behavior of KS in relation to BS and KB organizations considering the same environment as in the previous section, see Figure 1. We can observe that KS

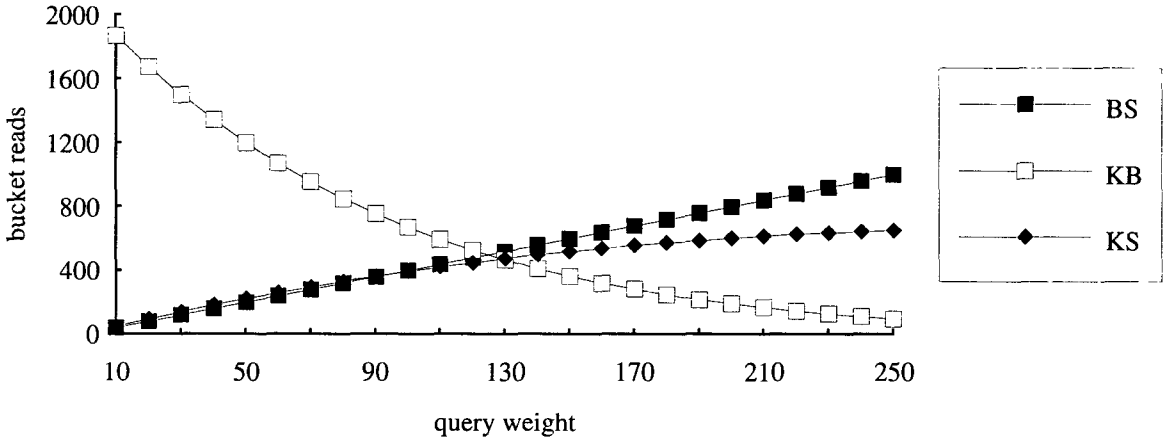

Figure 2: Performance comparison of BS, KB, and KS organizations; $n = 10^5, f = 512, c = 32768, \alpha = 0.75$.

is practically always able to outperform KB, considering the number of accessed buckets for any kind of a query, but the improvements are not uniform. From what we have seen, KS can be considered as a more efficient version of KB, retaining the excellent performance when processing low weight query signatures and achieving a gradually improved performance of processing query signatures with higher weights.

Now, let's try to see the performance in case of a 10 times larger file, i.e. a file containing $n = 10^6$ signatures, see Figure 3. As expected, performance of KS shows to be better and is practically constant, except for query signatures with very low weights, where, however, the performance is even better.

The last experiment we do concerns a file of 10^6 shorter signatures, i.e. signatures 256 bits long. Figure 4 shows performance of such file provided BS, KB, and KS organizations

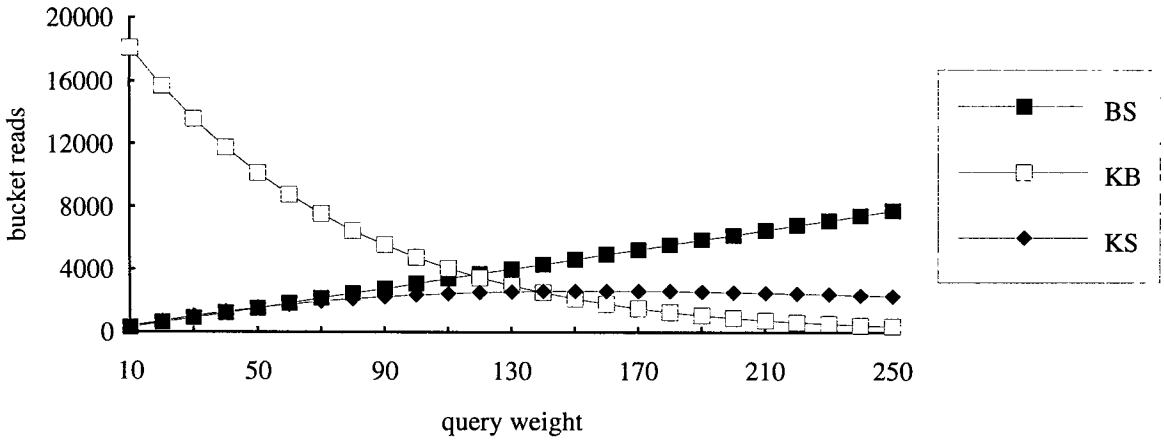

Figure 3: Performance comparison of BS, KB, and KS organizations; $n = 10^6, f = 512, c = 32768, \alpha = 0.75$.

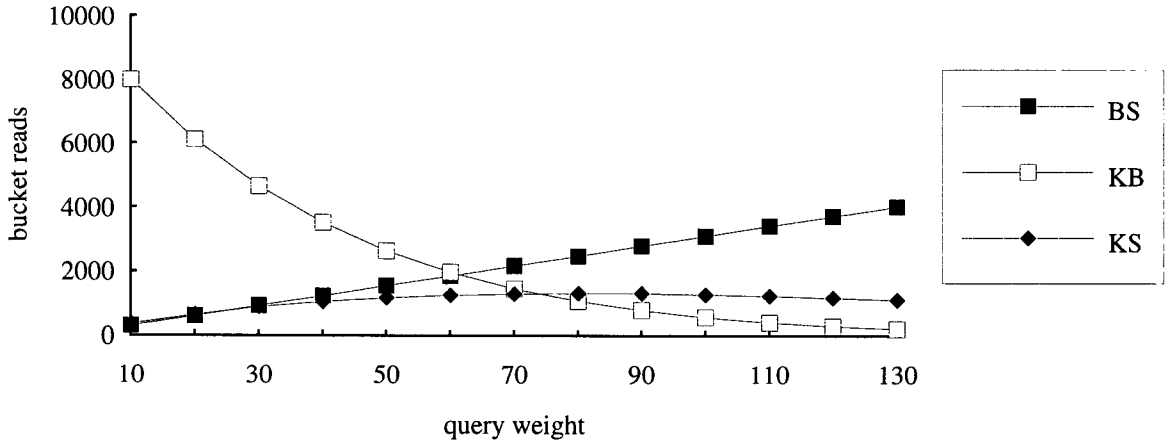

Figure 4: Performance comparison of BS, KB, and KS organizations; $n = 10^6, f = 256, c = 32768, \alpha = 0.75$.

are used. Apparently, the performance of KS is not influenced at all by the length of signatures.

In order to gain more insights into the behavior of KS, we consider how the retrieval cost, expressed in Equation 8, depends on the query signature weight. To this end we differentiate $C_{\mathbf{Q}}^{KS}$ with respect to $w(\mathbf{Q})$, from which it is derived (for brevity we omit intermediate steps) that the derivative is proportional ($\propto$) to:

$$\frac{\partial C_{\mathbf{Q}}^{KS}}{\partial w(\mathbf{Q})} \propto 1 - w(\mathbf{Q}) \cdot k(1) \cdot \frac{1}{2f - w(\mathbf{Q})} \tag{9}$$

which equals 0 when the query signature weight has value:

$$w(\mathbf{Q}) = \frac{2f}{k(1)+1} = \frac{2f}{\log_2 \left\lceil \frac{n}{\lfloor \alpha \cdot c \rfloor} \right\rceil + 1} \tag{10}$$

In correspondence of this value the retrieval cost is maximum. Since the weight of query signatures does not exceed $f/2$, the maximum is actually reached only if

$$\frac{2f}{k(1)+1} \leq \frac{f}{2} \Leftrightarrow k(1) \geq 3$$

In other terms, if the key size is 3 or more, the graph showing performance of KS exhibits a peak value, after which retrieval costs decrease, otherwise the cost monotonically grows with $w(\mathbf{Q})$. Indeed, Figure 3 refers to a case where the maximum is reached, according to Equation 10, when $w(\mathbf{Q}) \approx 161$, which is within the range of admissible weight values. On the other hand, Figure 2 shows a different behavior since the maximum would be reached for $w(\mathbf{Q}) \approx 308$, which is well beyond $f/2 = 256$.

The peak values observed in Figures 3 and 4 can analytically be evaluated by substituting into Equation 8 the value of the weight computed from Equation 10. After simplification, and with the advice that $m(1) = \lceil n/\lfloor \alpha \cdot c \rfloor \rceil$ is the number of distinct key values, we obtain:

$$\max\{C_{\mathbf{Q}}^{KS}\} = \frac{2f \cdot m(1)}{k(1)+1} \cdot \left(\frac{k(1)}{k(1)+1}\right)^{k(1)} \tag{11}$$

which is valid when $k(1) \geq 3$. Since $f \cdot m(1)$ is the total number of bucket in a KS organization, the quantity

$$\frac{2}{k(1)+1} \cdot \left(\frac{k(1)}{k(1)+1}\right)^{k(1)} \tag{12}$$

defines the maximum value of the Bucket Activation Ratio (BAR) for KS, that is, the maximum fraction of buckets that have to be accessed to process a signature query.

Figure 5 summarizes above analysis by showing how the maximum BAR depends on the size of the key, $k(1)$. Also shown is the value of the ratio $w(\mathbf{Q})/f$ for which the maximum BAR is obtained, as derived from Equation 10. For instance, when $k(1) = 6$, the maximum BAR is about 0.11, which is obtained if $w(\mathbf{Q})/f \approx 0.285$. This precisely means that, when the size of the key is 6, about 11 percent of the buckets have to be accessed in the worst case, and this occurs in case of medium weight queries, setting about 28.5 percent bits at "1".

5.2 Generalization

As the performance section above indicates, the proposed combination of the key-based and the bit-sliced approaches to a signature file organization is able to provide substantial increase in retrieval performance in comparison to performance of its constituent strategies (i.e., KB and BS organizations). However, KS also inherits properties concerning updates. In this respect, KS is able to deal with dynamic files, provided the group splitting strategy of Quick Filter is adopted – see [17] for details. But, because a group in KS contains f buckets, all of these buckets must be accessed to insert a signature (or must be split during a file expansion), thus modifications certainly are a matter of concern[1] – expensive update is a well known property of files with the BS organization.

In this respect, KS is more convenient for static (e.g., archive like) files where no or very few updates are required. In case of highly dynamic files, KB organization seems to

[1] Contrary to BS, KS can allocate groups of f buckets as contiguous buckets on disk. In this way random reads (writes) can be substituted by sequential ones, which alleviates the update problem. However, this violates our assumption about the uniform cost of accessing a bucket.

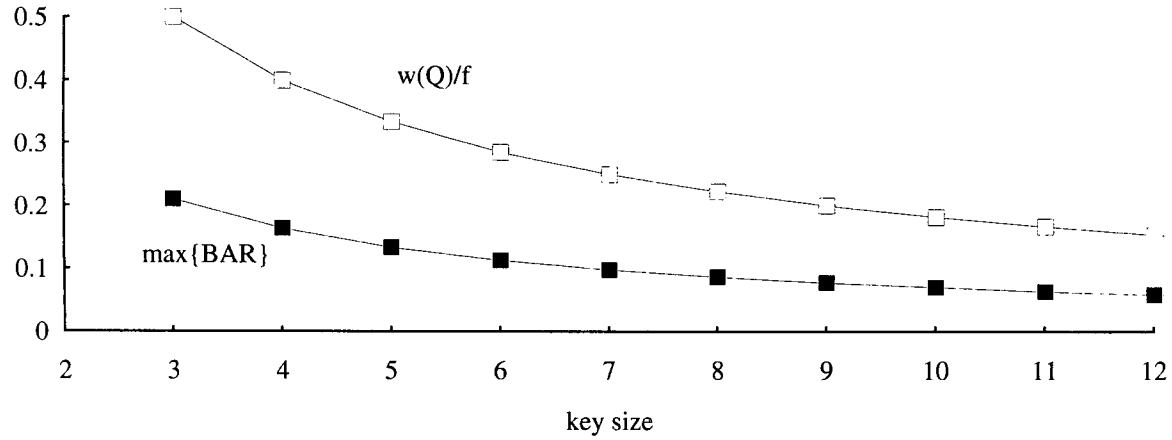

Figure 5: Maximum Bucket Activation Ratio (BAR) of the KS organization, and fraction of bits in the query signature ($w(\mathbf{Q})/f$) for which such maximum occurs.

be more convenient. This has lead us to propose a generalization of KS, designated KS^x, which sacrifices some of the retrieval efficiency of KS in order to increase update efficiency. By vertically splitting the signature file into frame-slices $\mathbf{F}^j$, rather than bit-slices, buckets of KS^x can be filled by parts of these frame-slices and the file organized as follows.

A group of signatures in the KS^x organization, identified by a specific signature key, contains f/x buckets, and, compared to KS (or equivalently KS^1), the signature key size $k(x) = \log_2\lceil n/\lfloor \alpha \cdot c/x \rfloor \rceil$ is gaining a bit in its length – depending on the value of x – which implies a better bucket activation ratio. On the other hand, accessed buckets can contain data (i.e., signature bit-slices) which need not be searched, causing additional access overhead resulting in an increase of the retrieval response time.

The trade-off between these two trends, respecting a specific value of the frame-slice size x, can be formalized as follows:

$$C_{\mathbf{Q}}^{KS^x} = E(f, x, w(\mathbf{Q})) \cdot \left\lceil \frac{n}{\lfloor \alpha \cdot c/x \rfloor} \right\rceil \cdot \left(1 - \frac{w(\mathbf{Q})}{2f}\right)^{\log_2 \left\lceil \frac{n}{\lfloor \alpha \cdot c/x \rfloor} \right\rceil} \tag{13}$$

where $E(f, x, w(\mathbf{Q}))$ is the expected value of the number of frames containing at least one *hit* bit-slice (i.e., a bit-slice which must be searched), provided the signature size, the frame size, and the query signature weight are f, x, and $w(\mathbf{Q})$, respectively. A solution to this classical probabilistic problem can be found in [15] by considering the case of random selection of elements from groups, without replacement – bit positions in a query signature are generated by a random function, but $w(\mathbf{Q})$ is the actual number of different positions set to "1". Accordingly, the expected number of hit frames can be determined as

$$E(f, x, w(\mathbf{Q})) = \frac{f}{x} \cdot \left(1 - \prod_{i=1}^{w(\mathbf{Q})} \frac{f - x - i + 1}{f - i + 1}\right) \tag{14}$$

A computationally more suitable, approximate, formula for exactly the same purpose is proposed in [14].

Notice, that for $x = 1$, $E(f, x, w(\mathbf{Q})) = w(\mathbf{Q})$, while for $x = f$, $E(f, x, w(\mathbf{Q})) = 1$, thus, $C_{\mathbf{Q}}^{KS^1} = C_{\mathbf{Q}}^{KS}$ and $C_{\mathbf{Q}}^{KS^f} = C_{\mathbf{Q}}^{KB}$, which can easily be proved.

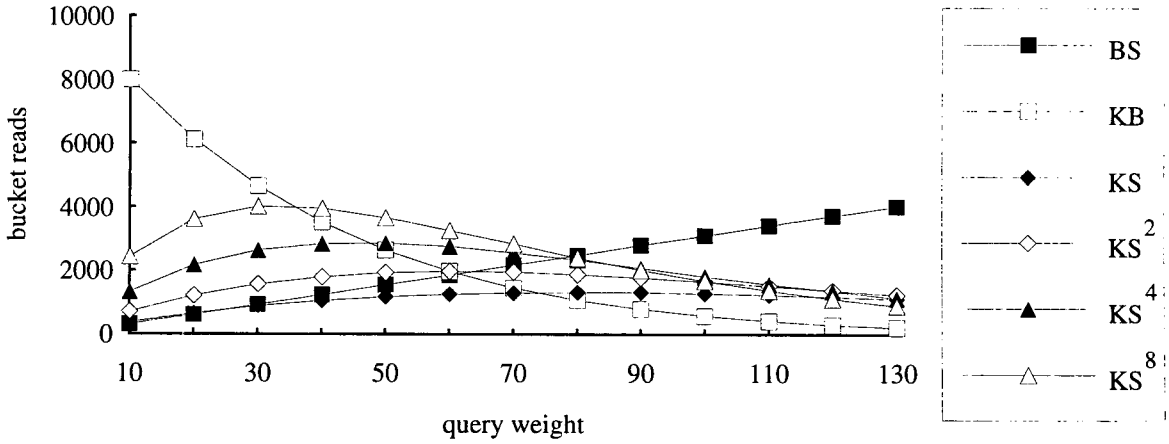

Figure 6: Performance comparison of BS, KB, and KS^x organizations; $n = 10^6, f = 256, c = 32768, \alpha = 0.75$, $x = 1, 2, 4, 8$.

Figure 6 demonstrates KS performance deterioration for the frame-slice sizes of $2, 4$, and 8. In fact, larger slices are probably not even convenient since the performance is becoming increasingly dependent on weights of query signatures – performance is much worse for low-weight queries – which implies to use KB organization where the expensive update problem does not exist at all.

6 Concluding remarks

Storage structures are typically designed to deal with particular query types (e.g., exact match, partial match, or range queries), but even queries of a type can differ significantly. Variability in the size of range or the number of terms specifying a partial match query usually results in substantially diverse processing costs of a give organization. In general, specific structures are known to be more convenient for some queries rather than for the others.

In the field of signature files – signature files are storage structures for partial match queries – some organizations are more convenient for queries containing just few terms (e.g., bit-sliced organization) while others perform very well for queries containing many terms (e.g., key-based partitioned organizations). Trends of performance characteristics of these organizations can be modelled considering a simple storage structure abstract view consisting of a set of buckets. Provided these storage buckets are accessed independently while executing a query, simple formulas for query processing cost estimation can be derived, and in this way a significant difference in performance of queries with different complexities can be demonstrated. Such situation suggests combining these approaches into a unified design which, presuming the parts are combined properly, would inherit positive properties of both approaches. This facts form the basic ideas of the key-based bit-sliced organization proposed in this article.

As our simulation results demonstrate, the search performance is indeed very good and quite stable for any kind of query, but due to the bit-sliced approach to storing signatures, the update performance can become quite high. For this reason, we further propose a generalization of KS in which frames of bit slices, rather than individual bit slices, are

stored together. In this way, the trade-off between the search efficiency and the update costs can be adjusted.

The aim of this article was to show how existing organizations can conveniently be combined to obtain a new quality. In order to make things manageable (i.e., systematically explaining concepts and the design), we have simplified the implementation environment. We are aware of the fact that mainly the assumption about the uniform cost for accessing a bucket is not correct for all storage environments and that retrieval of consecutive buckets can change the presented figures. However, such situation is very difficult to manage analytically and is now a subject of our experimental research. Future research plans also consider parallel storage environments, with an expectation to improve performance even further and solve the problem of expensive updates.

Acknowledgements

We would like to thank the anonymous referees who gave relevant suggestions to improve the quality of the paper.

The research was partially funded by the ESPRIT Long Term Research program, project no. 9141, HERMES (Foundations of High Performance Multimedia Information Management Systems), and by Italian CNR, under contract no. 94.00388.CT12.

References

[1] Cardenas, A.F., Analysis and Performance of Inverted Data Base Structures. *Communications of the ACM*, 1975, 18(5):253–263.

[2] Ciaccia, P., Tiberio, P., and Zezula, P. Declustering of Key-Based Partitioned Signature Files. Accepted for *ACM Transactions on Database Systems.*

[3] Ciaccia, P. and Zezula, P., Estimating Accesses in Partitioned Signature File Organizations. *ACM Transactions on Information Systems*, 1993, 11(2):133–142.

[4] Faloutsos, C., Signature Files. In: *Information Retrieval: Data Structures and Algorithms*, W. B. Frakes, R. Baeza-Yates (eds.), Prentice-Hall, 1992, pp 44–65.

[5] Lee, D.L. and Leng, C.-W., Partitioned Signature Files: Design Issues and Performance Evaluation. *ACM Transactions on Office Information Systems*, 1989, 7(2):158–180.

[6] Lin, Z., Concurrent Frame Signature Files. *Distributed and Parallel Databases*, 1993, 1(3):231–249.

[7] Lin, Z. and Faloutsos, C., Frame-Sliced Signature Files. *IEEE Transactions on Database Systems*, 1992, 4(3):281–289.

[8] Panagopoulos, G. and Faloutsos, C., Bit-Sliced Signature Files for Very Large Text Databases on a Parallel Machine Architecture. *Lecture Notes in Computer Science*, No. 779, M. Jarke, J. Bubenko, and K. Jeffery (eds.). Proceedings of EDBT'94, Cambridge, United Kingdom, 1994, Springer-Verlag, pp. 379–392.

[9] Rabitti, F. and Zizka, J., Evaluation of Access Methods to Text Documents in Office Systems. *Proceedings of the 3rd Joint ACM-BCS Symposium on Research and Development in Information Retrieval*, Cambridge, United Kingdom, 1984, pp. 21–40.

[10] Roberts, C.S., Partial Match Retrieval via the Method of the Superimposed Codes. *Proceedings of the IEEE*, 1979, 67(12):1624–1642.

[11] Sacks-Davis, R., Kent A., and Ramamohanarao, K. Multikey Access Method Based on Descriptors Superimposed Coding Techniques. *ACM Transactions on Database Systems*, 1987, 12(4):655–696.

[12] Tiberio, P. and Zezula, P., Selecting Signature Files for Specific Applications. *Information Processing and Management*, 1993, 29(4):487–496.

[13] Tiberio, P. and Zezula, P., Signature File Access. *Encyclopedia of Microcomputers*, Kent, A. and Williams, J.G. (eds.), Marcel Dekker, Inc., New York, 1995, Vol. 16:377–403.

[14] Whang, K.-Y., Wiederhold, G., and Sagalowicz, D. Estimating Block Accesses in Database Organizations: A Closed Noniterative Formula. *Communications of the ACM*, 1983, 26(11):940–944.

[15] Yao, S.B., Approximating Block Accesses in Database Organizations. *Communications of the ACM*, 1977, 20(4):260–261.

[16] Zezula, P., Ciaccia, P., and Tiberio, P., Hamming filter: A Dynamic Signature File Organization for Parallel Stores. *Proceedings of the 19th International Conference on VLDB*, Dublin, Ireland, 1993, pp. 314–327.

[17] Zezula, P., Rabitti, F., and Tiberio, P., Dynamic Partitioning of Signature Files. *ACM Transactions on Information Systems*, 1991, 9(4):336–369.