Frame-Sliced Partitioned Parallel Signature Files

Fabio Grandi° Paolo Tiberio° Pavel Zezula◇*

°Dipartimento di Elettronica, Informatica e Sistemistica,
Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy

◇Technical University of Brno
Udolni 19, 602 00 Brno, Czechoslovakia

Abstract

The retrieval capabilities of the signature file access method have become very attractive for many data processing applications dealing with both formatted and unformatted data. However, performance is still a problem, mainly when large files are used and fast response required. In this paper, a high performance signature file organization is proposed, integrating the latest developments both in storage structure and parallel computing architectures. It combines horizontal and vertical approaches to the signature file fragmentation. In this way, a new, mixed decomposition scheme, particularly suitable for parallel implementation, is achieved. The organization, based on this fragmentation scheme, is called *Fragmented Signature File*. Performance analysis shows that this organization provides very good and relatively stable performance, covering the full range of possible queries. For the same degree of parallelism, it outperforms any other parallel signature file organization that has been defined so far. The proposed method also has other important advantages concerning processing of dynamic files, adaptability to the number of available processors, load balancing, and, to some extent, fault-tolerant query processing.

*A substantial part of the work was done while the author was in Bologna on leave from the Technical University of Brno.

15th Ann Int'l SIGIR '92/Denmark-6/92

1 Introduction

The signature file access method has become a well-known concept for implementing associative retrieval on data files kept in a stable store. It supports execution of conjunctive *partial-match* queries. Unlike many other access structures designed for this purpose (e.g. multi-dimensional tree or hash access structures, grid files, etc.), the number of search dimensions — attributes or components — can be large and need not even be fixed for all records in a specific file. Such features have become attractive for many application areas including information retrieval, office automation, statistical databases, computerized libraries, Prolog databases, and conventional database management systems.

The effect is achieved by applying a special kind of auxiliary file, the *signature file*, which contains data object (or record) abstractions, called *signatures*. Signatures are bit patterns that are substantially smaller in size than the records themselves. Depending on the *signature extraction method* adopted, all signatures in the file have a specific structure.

Sequential organization of signature files works well for small files. However, for larger files, performance becomes a problem. Other organizations of signature files, see for example [3,18,21] for a review, substantially improve performance for specific types of queries. However, many applications still need faster and/or more stable performance (i.e. not dependent on the type of query) which cannot be provided by signature files yet.

We believe that an effort should be made to develop a signature file organization that would perform well for all, or at least most, of its potential applications. As the main research direction here, we study an application of parallel computing techniques to the signature file implementation. We suggest a *fragmentation scheme* which is based on recent signature file designs, namely the *frame-slice* approach [4] and the *dynamic*

partitioning technique called *quick filter* [21].

The rest of the paper is organized as follows. In Section 2 we summarize the most important existing techniques for signature file organization which are relevant to our work. In Section 3 a new technique for parallel signature file processing is proposed. An evaluation of the technique appears in Section 4. Conclusions and future research directions form the content of Section 5.

2 Assumptions and Preliminaries

The literature concerning signature file techniques is enormous. Because of the limited space of this paper, we suppose that the reader is familiar with the basic principles of the signature file access method and multiprocessor computer architectures. In this section, we survey only the most relevant topics concerning our research.

Our fragmentation scheme is based on the approach of *partitioned, superimposed signature files*, which is a way of achieving *horizontal* fragmentation, and on the idea of *sliced organizations*, in order to make also *vertical* fragmentation possible.

For the sake of simplicity, we assume data objects, O_i $(i = 1, 2, \ldots, N)$ to be represented by a *superimposed object signature*, S_i, which is a fixed-length bit vector $b_{i1}, b_{i2}, \ldots, b_{iF}$. N is the size of the file, and F is the *signature length*.

To obtain the *word signature*, each *object descriptor* (*term* or *word*) is hashed into a bit vector in such a way that m bit positions are set to "1" while all the other bits remain zero. To obtain the *object signature* (or simply *signature*, for short), all word signatures for that object are superimposed (i.e. "or"-ed).

When objects are searched for a word or a set of words, a *query signature*, SQ, is generated from the *query*, Q, in the same way as described for the object signature above. An object signature, S_i, qualifies for a given query if and only if for all bit positions in SQ, which are set to 1, the corresponding bit positions in S_i are also set to 1. This, so-called *inclusion* condition can be formally defined as follows:

$$\{S_i | (S_i \wedge SQ) = SQ\}$$

All disqualified object signatures indicate that their associated objects are guaranteed not to contain words specified in the query. On the other hand, the signature file may indicate that an object contains a particular word, while actually it does not. This situation is known as *false drop*.

2.1 Partitioned Superimposed Signature Files

Horizontal fragmentation is the main idea of *partitioned signature files*. Organizations based on this approach have a reasonable storage overhead, very good performance (except for low-weight queries), efficient update, and suitability for parallel processing. Furthermore, previous works [9,14,21] offer not only interesting performance evaluations but also a theoretical basis and formulas for estimating performance in specific situations.

A *partitioned organization* groups object signatures into *partitions* (e.g. disks, blocks or pages). Then, some of the partitions need not be searched while executing a query, and in this way, the number of accesses can be reduced. The criterion for inserting the i-th signature in a partition is based on the value of the *signature key*, KS_i. The key, in general, is a substring of the signature. All signatures in the j-th partition have the same signature key, which is also the *partition key*, KP_j. When a query signature is generated, its key, KSQ, is obtained by extracting a substring from SQ in the same position as defined for keys of the object signatures. The j-th partition is then activated (i.e. searched) only if KSQ includes the partition key KP_j. The following expression defines the set of activated partitions for a given query:

$$\{j | (KSQ \wedge KP_j) = KSQ\}$$

The main implementation problem of any partitioned organization is the definition of keys. This, consequently, determines a function which maps signatures into partitions. Three partitioning schemes with different key specifications have been designed and analyzed in [9]. They are known as the *Fixed Prefix*, *Extended Prefix*, and *Floating Key* methods. Although there has been an attempt (see [10]) to make these schemes dynamic, the way of determining the keys still remains static and, in this way, limits the dynamic features of the organization.

Quick filter [20,14,21] is another partitioning scheme. It uses a dynamic split function, which works well with *linear hashing* [11] as the underlying storage structure. The signature key, for all signatures in the file, is defined as the h-bit or the $(h-1)$-bit suffix of the signature. The value of h varies with the size of the file, N, according to the relationship $2^{h-1} < N \leq 2^h$. In a specific file, there are $2^h - n$ partitions with the key size $h - 1$ and $2N - 2^h$ partitions with the size h. Then, *quick filter* considers any KS_i as a binary number, the value of which determines the partition number, j, into which the signature S_i is stored.

2.2 Sliced Signature Files

The main idea of *sliced signature files* is vertical fragmentation. A storage structure that uses *bit-slice*, as opposed to *bit-string*, organization of signatures has been described by Roberts [15]. Since a signature file can be considered as a two-dimensional table, Roberts suggested storing the columns, rather than the rows, of the signature table together. The advantage of such an arrangement appears when answering a query because only some columns need to be accessed.

Another idea for extracting and organizing signatures, called the *frame-slice* approach, was recently outlined in [4]. The frame-slice signature file forces each word to hash into bit positions that are close to each other in the object signature. More precisely, the object signature, S_i, is divided into n *frames* of f_n consecutive bits ($f_n = F/n$). Each word in the object hashes into one of the n frames. Another hash function sets, for a given word, m bits to "1" within the previously determined frame. The object signatures are stored *frame-wise*, using n *frame files*.

For a *single-word query*, only one frame file has to be searched. However, with a growing number of words, the performance of query execution becomes less efficient because more and more frame files must be searched. Another problem is the control of *false drops* which can be expected to grow with the increasing number of frames.

It is quite clear that the *bit-slice* approach can be generalized for slices containing more than one bit and then, instead of *bit-slices*, we get *frame-slices*. However, they are different from the *frame-slices* discussed above because, in this generalization, one word can set bits into "1" in multiple frames.

In the following we will consider both types of frames. In order to distinguish them, we will call frames designed according to the *frame-slice* approach as *clustered frames* while the other frames will be called *unclustered frames*.

2.3 Parallel Processing

The last few years have seen a proliferation of interest in the use of parallel processing techniques where multiple processors operate together to reduce the total time required for a computational task. The use of parallel hardware can bring about substantial improvements in performance if the approach is properly applied.

Three architectures of a *multiprocessor database computer* have been proposed: *Shared Everything* (SE), *Shared Disk* (SD), and *Shared Nothing* (SN). A detailed description of these architectures can be found in [5]. There has been considerable debate about which architecture is the most suitable for a database management system implementation. According to [5], the *coherency control* problem limits the number of processors that can efficiently cooperate in SE or SD systems. On the other hand, data coherency control is not a problem in SN systems. However, SN systems are very sensitive to the distribution of data on secondary memory, which may lead to the *data skew* problem [7]. When the data are seriously skewed, rebalancing of the data load among processing nodes is necessary to achieve a good system performance. Given these drawbacks, none of the three multiprocessor database architectures stands out as the absolute winner. Depending on the application, a combination of the three parallel architectures can be employed to benefit from the advantages of each scheme.

2.3.1 Parallel Signature File Organizations

Word-serial, bit-parallel hardware signature processors have been designed [1,8] to speed up the comparison of object signatures against a query signature. They attempt to use the *bit-slice* signature file organization [15] as a fragmentation scheme for a parallel hardware implementation. In the [8] design, modules are connected to a bus and — when a query signature is broadcast to them — activated in parallel to search specific bit-slices. This concept can be directly extended to frame-slice signature files. Since each slice has the same size, the time for searching the whole signature file is approximately the time that is needed to search one signature file slice. We will call this approach *Vertical Parallelism* (VP).

Another approach has been employed for parallel signature file processing [16,13] and implemented on the Connection Machine CM-2. In this design, a subset of object signatures, that is a horizontal fragment, is assigned to each processor. Although the probing of signatures is fast — the machine consists of up to 64K bit-serial processing elements — it always goes through all the signatures in each processor. Because of the fragmentation scheme used, this approach can be called *Horizontal Parallelism* (HP). The HP technique has also another problem: it only allows *intra-query* parallelism. It still improves the speed of a single query evaluation, but *inter-query* parallelism cannot be exploited. Therefore, only one query can be serviced at a time.

The partitioned signature file concept of [9] can also be implemented in a parallel environment by assigning each partition (i.e. a horizontal fragment) to a separate processor. In this respect, the parallel implementation of a partitioned signature file is quite similar to HP. However, because only some partitions are searched for a query, the non-activated processors are available to service other queries. The fact of allowing both *intra-*

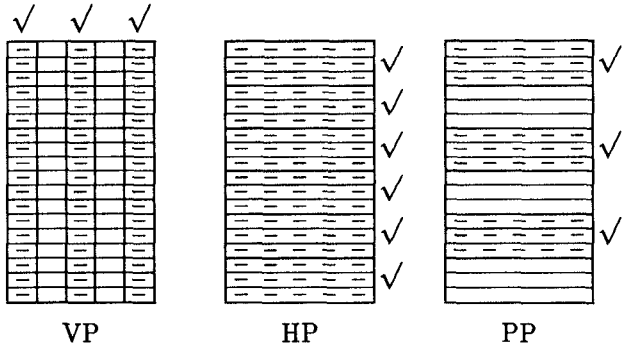

Figure 1: Parallelism in searching the Signature File. Marks ($\surd$) signify active processors.

and *inter-query* parallelism is the major advantage of the partitioning scheme. Such an approach can be called *Partitioned Parallelism* (PP).

A schema of a signature file space, searched in parallel by active processors, can be seen in Figure 1 for each of the three approaches outlined above.

3 The Design

In order to achieve a high-performance signature file organization, we try to exploit the latest developments in storage structure and computing system architectures. We call our design *Fragmented Signature File* (FSF) which is based on the following three hypotheses:

1. Performance of a partitioned signature file organization improves with increasing query weights. On the other hand, the performance of a sliced organization deteriorates with increasing query weights. A combination of these methods should lead to a design with good and more stable performance for any type of query;

2. Proper application of a multiprocessor database computer architecture can substantially improve signature file performance;

3. Accessing secondary storage is very slow. Any technique that reduces the number of accesses, even with some extra cost of internal processing and communications, is likely to shorten the total response time of the entire organization.

3.1 The Architecture

The main problem with designing a signature file organization in a parallel computer environment concerns the *fragmentation scheme*. This is, in fact, responsible for distributing bits of the signature file among a specific number of processing units. The optimization criterion for the distribution is the *query response time*, which should be minimized with respect to all possible queries. With p parallel processing units, the response time to a query is defined as $\max\{C_1, C_2, \ldots, C_p\}$, where C_i $(1 \leq i \leq p)$ is the response time (i.e. the cost measured as the number of physical page accesses) of the i-th processing unit. Obviously, the secondary store (disk unit) with the largest number of accesses required to fulfil the query will result in the longest delay, thus determining the response time for the whole query.

Given that the above fragmentation problem is *NP-complete* [17], our approach to the solution is heuristic. However, as we show in the performance evaluation section, it gives better results than all the approaches previously proposed for the signature file parallel organization. In the past, the fragmentation problem has been studied mainly in terms of data relations processed by a distributed database system (see e.g. [12] for more details and many references). Our attention is focused on signature file fragmentation.

Two kinds of fragmentation can be considered as fundamental: *horizontal* and *vertical*. By *vertical fragmentation* we mean the decomposition of a signature file into a disjoint set of signature file frames. On the other hand, through *horizontal fragmentation*, a signature file is decomposed into a disjoint set of signature sub-files, where all signatures in each sub-file have a common key (i.e. the signature key).

Mixed fragments are generated from a combination of horizontal and vertical fragmentations. A fragment of this kind can be seen as a product derived from a global signature file in two consequent steps, involving vertical and horizontal fragmentation. Since each of these steps can, in fact, be missing, horizontal and vertical fragments are just special cases of mixed fragments.

In general, mixed fragments can be disjoint or overlapping. In our proposal, both horizontal and vertical fragmentations produce disjoint data fragments. Consequently, the mixed fragments are also disjoint. This implies that the necessary requirement of any fragmentation scheme for *completeness* and *reconstruction* is satisfied in a trivial way.

Definition 1:

The fragmentation scheme of FSF (Fragmented Signature File) is a mixed fragmentation scheme produced by double horizontal fragmentation of vertical fragments. It consists of *frames, partitions* and *blocks*, where:

1. The *frame* R_j $(j = 1, 2, \ldots, n)$ is the j-th vertical fragment of a signature file, also called the *frame file*, containing N *frame signatures* (i.e. vertical fragments of signatures) with length f_n;

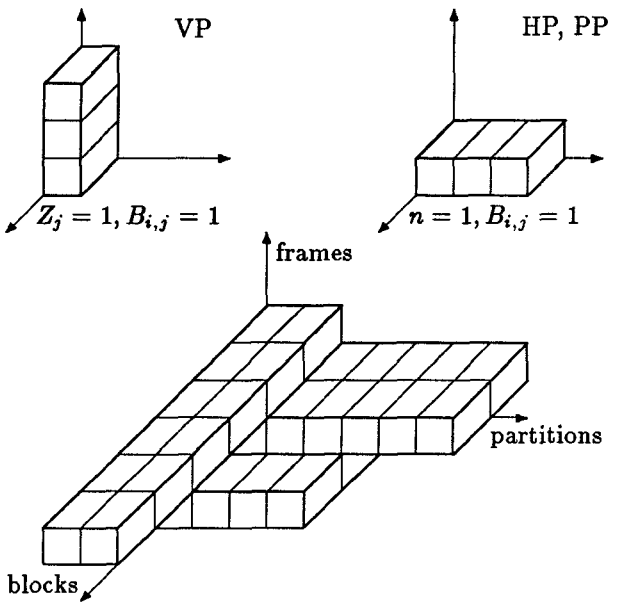

Figure 2: Architecture of Fragmented Signature File organization with its special cases.

2. The *partition* $P_{i,j}$ is the i-th horizontal fragment of the j-th frame. It should be served by a specific processing unit, $PU_{i,j}$. There are Z_j partitions available for the frame j, where $Z_j \in \{1, 2, \ldots, N\}$, and $j = 1, 2, \ldots, n$;

3. The *block* $BL_{i,j,l}$ is the l-th horizontal fragment of the partition $P_{i,j}$ containing $B_{i,j}$ blocks. The block is also the access unit of FSF.

The architecture which supports this method can be seen in Figure 2.

The complexity of our organization is determined by the fragmentation scheme and is, in fact, three-dimensional. A specific configuration depends on the amount of data, N, the number of frames, n, and the number of processing units for each frame, Z_j, where $j = 1, 2, \ldots, n$.

The number of frames, n, is a design parameter of FSF and must be determined before any data instances are inserted into the file. However, the number of partitions in the frames and the amount of signatures stored — which determine the number of blocks — can change during the file's lifetime. It should be emphasized here that this architecture implies *dynamic fragmentation* in both horizontal fragmentation steps.

Assigning different numbers of processing units to different frames can improve the performance of the signature file when some frame files are searched more often than others or when certain types of queries require faster response. A non uniform processor force on individual frames can also be used to cope with the *data skew effect*. However, we usually expect each frame to be assigned the same number of processing units (i.e. $Z_j = Z$, for $j = 1, 2, \ldots, n$). Therefore, assuming frame signatures to be generated at random (thus $B_{i,j} = B_n$, for all i and j), the solid in Figure 2, which represents a specific configuration of FSF, becomes a cube.

Our approach includes, as particular cases, all previous approaches to parallelism. In FSF architecture, vertical parallelism can be defined by the restriction: $Z_j = 1, B_{i,j} = 1$. Horizontal and partitioned parallelisms are characterized by the same restriction: $n = 1, B_{i,j} = 1$. Figure 2 also shows these special cases graphically.

The architecture of FSF extends beyond the previous approaches as follows:

1. Given the same number of processors for each frame file (i.e. $Z_j = Z$) and the uniform distribution of signatures in the file, the load is balanced because each partition contains about the same number of blocks. If the load is unbalanced, a different number of processors for different frame files can be applied as a provision to tune the data skew problem;

2. The actual number of block accesses, compared to the total number of blocks ($\sum_{i,j} B_{i,j}$), depends on a specific query. In general, it is expected to be low. Accesses to blocks can be discarded with respect to all three dimensions of the architecture. The first dimension can discard frames. The second dimension discards partitions in each activated frame. The third dimension can discard blocks within partitions.

However, this architecture also requires some overhead. In FSF organization, ordering of frame signatures stored in individual frames need not be the same. Therefore, a signature cannot be simply assembled from its frame signatures on the basis of their position in frame files, as assumed in the original frame-slice organization design [4]. On the contrary, relationships between the individual parts must be explicitly maintained in order to make the assembling process manageable. Bad implementation of the relationships causes additional space overhead and may even increase the retrieval cost. Our suggestion for dealing with this problem is the subject of the following subsection.

3.2 Implementation

Implementation of FSF in a parallel system environment concerns four mutually related problems: the choice of the parallel computer system architecture, the frame

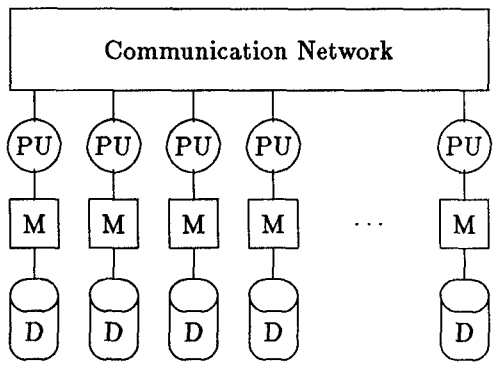

Figure 3: The *Shared Nothing* (SN) architecture. Meaning of the symbols: PU, processing unit; M, local main memory; D, local disk.

(i.e. vertical) fragmentation, the implementation of the relationships between the frame signatures, and the algorithm which allocates the frame signatures to the individual partitions and blocks (i.e. double horizontal fragmentation). We will discuss these problems in detail in the following.

3.2.1 The parallel architecture

Because of its simplicity, we have decided to accept the *Shared Nothing* (SN) architecture of multiprocessor database computers. Its main features can be seen in Figure 3. Nevertheless, we are aware of the fact that the fragmentation scheme can also be implemented on other parallel database computer architectures. In our design, SN architecture uses $p = \sum_j Z_j$ processing units, where each unit consists of a processor, a memory module, and a disk drive. One partition of a frame is assigned to each processing unit and stored in the local disk. The processing units from 1 to Z_1 contain blocks of the first frame. The units from $Z_1 + 1$ to $Z_1 + Z_2$ contain blocks of the second frame, and so on.

3.2.2 Frame generation

As we have already explained, frames can be generated in two different ways: clustered frames and unclustered frames. However, different frame construction methods can lead to different false-drop levels and/or different signature size. A detailed analysis of the problem is beyond the scope of this paper and will be the subject of our future research. Anyway, at present, our findings indicate that the differences are negligible for sufficiently large frames.

3.2.3 Relationships between frame signatures

In order to maintain relationships between frame signatures of a specific signature, an object identifier, OID, is appended to each frame signature. The OID value is the same for all the frame signatures of a specific signature but is unique in the signature file. Although it introduces a storage space overhead proportional to the number of frames, this solution is general because OID is considered to be an identification independent of data and storage [6]. The main advantage is that the OID method does not restrict the application domain. On the other hand, it requires performing *unions* and *intersections* of sets of OIDs obtained as a result of query evaluation on the individual processing units. In general, this process can be defined by the expression:

$$\bigcap_j \bigcup_i \mathrm{Res}(PU_{i,j}),$$

where $\mathrm{Res}(PU_{i,j})$ is the *Response* (i.e. the returned set of OIDs) of the i-th processing unit of the j-th frame. However, not all processing units need be activated for a specific query. In this case, only the responses of the activated units are considered.

This approach to implementation of relationships between signature frames can lead to additional processing overhead unless parallelism is used. Our suggestion is to perform these set operations according to the following procedure:

1. Each active processing unit works independently in parallel on a particular partition of a specific frame file; it accesses blocks with potentially qualifying signature frames; it examines them for qualification; it collects the OIDs of the qualifying signature frames, $\mathrm{Res}(PU_{i,j})$;

2. All the OIDs collected by a processing unit (i.e. its response) are dynamically maintained as sorted sets in the local main memory of the unit;

3. If multiple processing units are activated for a given query, they do not usually finish at the same time. There may be differences in data load on individual processing units, but mainly query weights of different frame query signatures are not likely to be the same. As soon as a processing unit commits, the union or intersection can be performed accumulating the resulting set on the unit which commits last. Operations are performed according to the following rules:

   (a) If there is another committed process of the same frame, compute the union of the two sets;

(b) If there is no other active processing unit belonging to the same frame, the frame processing commits;

(c) If there are other committed frame processes, compute the intersection of their responses.

4. Repeat Step 3. until no active processing unit runs in order to execute a given query.

The fact that the query processing time on individual units is not the same allows our system to perform the OID unions and intersections in parallel. Implementation of the union of disjoint sets is very efficient. Collected OID sets are disjoint as a consequence of the horizontal fragmentation scheme applied. Moreover, the frame which commits first is obviously expected to produce the smallest set of OIDs, and owing to intersections, this set can never grow. It follows that the intersections of all the responses are performed optimally.

3.2.4 Assignment of frame signatures

Assigning frame signatures to specific partitions and blocks is the crucial implementation problem of our fragmentation scheme. The frame signatures must be stored in such a way that subsequent query processing is possible. However, the assignment algorithms must also allow the evolution of the configuration. Specifically, the algorithms must respect the possibility that the number of frame signatures, as well as the number of processing units, can change.

We propose to solve the problem by applying two *split-hash functions*: one for selecting a partition, served by a specific processing unit in a frame, and the other for determining a proper block in the partition. The advantage of such a solution is that by applying the *linear hashing* method [11], the requirements for the dynamic frame signature assignment are automatically satisfied. We fully exploit experience gained from designing the *quick filter* [21], where the same idea was successfully used in a one-dimensional problem.

Without loss of generality, we assume that a *frame signature*, FS, is a sequence of bits $(b_1, b_2, \ldots, b_{f_n})$ and that the number of blocks on the disk of a processing unit is B_n. We define the *levels of hashing* for addressing the partitions (processing units) and the blocks of a frame signatures as k and h, respectively.

The first hashing function, $\mathrm{HP}(FS, k, Z_j)$, determines the partition to which the frame signature FS of the j-th frame file is assigned. Here the address space goes from 1 to Z_j. The parameters of the function are constrained by

$$2^{k-1} < Z_j \leq 2^k$$

Given the simplest case of a single processing unit for every frame file (i.e. $Z_j = 1$ and $k = 0$) we define $\mathrm{HP}(FS, 0, 1) = 1$. For every other case, where $k > 0$, the following function is used:

$$\mathrm{HP}(FS, k, Z_j) = 1 + \begin{cases} \sum_{\ell=0}^{k-1} b_{\ell+1}\, 2^\ell & \text{if } \sum_{\ell=0}^{k-1} b_{\ell+1}\, 2^\ell < Z_j \\ \sum_{\ell=0}^{k-2} b_{\ell+1}\, 2^\ell & \text{otherwise} \end{cases}$$

As soon as a partition is determined, the frame signature FS is stored in a block according to the second hashing function $\mathrm{HB}(FS, h, B_n)$. We suppose that the addressing space goes from 1 to B_n. Again, the parameters are constrained by

$$2^{h-1} < B_n \leq 2^h$$

At the beginning, one block is expected to be available in each partition unit (i.e. $B_n = 1$ and $h = 0$). In this case $\mathrm{HB}(FS, 0, 1) = 1$. For every other case, where $h > 0$, the following function is used:

$$\mathrm{HB}(FS, h, B_n) = 1 + \begin{cases} \sum_{\ell=0}^{h-1} b_{f_n-\ell}\, 2^\ell & \text{if } \sum_{\ell=0}^{h-1} b_{f_n-\ell}\, 2^\ell < B_n \\ \sum_{\ell=0}^{h-2} b_{f_n-\ell}\, 2^\ell & \text{otherwise} \end{cases}$$

Blocks addressed by the HB function in partitions are expected to be of fixed size. Consequently, for a growing signature file, the number of such blocks must increase. Linear hashing [11] is an elegant way of dealing with this problem because the file can grow or shrink by adding or deleting only one block (physical page) at a time.

The same approach works also when the system needs reconfiguration. For example, if one processor is added or removed for a given frame, the change affects only one (or two) processing unit(s).

4 Performance Evaluation

This section concerns performance estimation of FSF. We aim at deriving results which can be considered as representative of statistics collected from a large sample of fixed-weight queries. In this perspective, we use *expected* values of *random variables* as fair approximations of the *mean* values of the physical quantities they represent. Consequently, they are used in formulas as if they were *deterministic values*. In other words, we do not take into account error propagation through *higher-order moments*.

For performance evaluation, we consider a sample signature file with parameters listed in Table I. We assume a fixed number, $p = 32$, of available processors. For the sake of simplicity, we further assume the number of partitions used to be the same for every frame. In particular, we assume that the k-prefix used as the partitioning

Number of signatures	N	100,000
Signature size	F	2048
OID size	d	24
Load factor	α	0.75
Block size	D	16384 (2Kbytes)

Table I: Parameters of a sample signature file.

key has a constant length in each frame; therefore there are exactly 2^k partitions per frame. This particular solution does not restrict the data-adaptive features of FSF design, thanks to the dynamic implementation of horizontal fragmentation by using the idea of *quick filter* [21]. Letting p be the number of processors, each of them with a partition of frame signatures stored on its local disk, the constraint on the hashing level is:

$$n\ 2^k = p \tag{1}$$

In the experiments, we study how the system performance changes when different combinations of n and k values, respecting constraint (1), are adopted in the FSF organization.

Our proposal will be compared with the earlier designs, mainly with *horizontal parallelism* ($n = 1, k = \log_2 p$) and *vertical parallelism* ($n = p, k = 0$). It should be borne in mind that, in both cases, signature file partitions are organized and searched sequentially. In this respect, *quick filter* organization of partitions in frames is one of the most important differences of FSF design.

4.1 Storage requirements

In the most general case, signatures are divided into n frame files with signature frames $f_n = F/n$ bits long. Then, the signature frames are distributed into $2^k = p/n$ partitions in each frame. The k-bit prefix of signature frames need not be stored in the partitions, because it has a common value, the partition key, in all the frame signatures making up a partition. Therefore, in a sequential block of dimension D bits,

$$N^s_{nk} = \frac{D}{f_n - k} \tag{2}$$

signature frames of length $f_n - k$ can be stored. Each partition of a frame, made up of $N/2^k$ frame signatures, needs

$$B^s_{nk} = \left\lceil \frac{N}{2^k N^s_{nk}} \right\rceil \tag{3}$$

sequential blocks on disk. The value of B^s_{nk} has a weak and non-linear dependence on n. However, we can slightly overestimate the B^s_{nk} value by neglecting k with respect to f_n in (2), yielding:

$$B^s_{nk} \simeq B^s_{n0} = B^s_p = \left\lceil \frac{NF}{pD} \right\rceil \tag{4}$$

which does not depend on n when p is fixed. The exact values of B^s_{nk} for our sample signature file are reported in Table II ($B^s_p = 391$ in this case).

If, on the other hand, partition blocks are dynamically organized as quick filters (as suggested in FSF), then the value

$$N^q_{nk} = \frac{\alpha D}{f_n - k + d} \tag{5}$$

represents the average number of signature frames that can be stored in one block. The parameter α ($\alpha < 1$) represents the *loading factor* induced by the application of linear hashing, whereas d represents the length in bits of the OIDs ($2^d \geq N$). The total number of disk blocks needed to store one partition of the frame, B^q_{nk}, can be computed as:

$$B^q_{nk} = \left\lceil \frac{N}{2^k N^q_{nk}} \right\rceil \tag{6}$$

Thus, the hash level, enforced in any frame partition, equals:

$$h_{nk} = \lceil \log_2 B^q_{nk} \rceil \tag{7}$$

Once again, the hash key used by the quick filter need not be explicitly stored in blocks, as all frame signatures hashed in the same block have a common key. Owing to the linear hashing method, some blocks use a suffix of length h_{nk} as the hash key, whereas the other blocks use a suffix of length $h_{nk} - 1$. We put forward the conservative hypothesis that, in every block, $h_{nk} - 1$ bits are always omitted. The number of blocks and the hash level can be iteratively computed as the numerical solution of the simultaneous equations (6), (7), and:

$$N^q_{nk} = \frac{\alpha D}{f_n - k + d - h_{nk} + 1} \tag{8}$$

In our simulations, this expression will replace (5). It should also be noticed that the k-prefix and the h-suffix are assumed to be non-overlapping in the signature frame.

4.2 Block access estimations

In this subsection, we derive expressions for calculating the average number of blocks accessed per partition to answer a query composed of a given number of terms (words).

If the signature file partitions are organized sequentially, as supposed for the vertical and horizontal parallelism methods, all qualifying partitions must be entirely searched, and thus the query cost is equal to the partition dimension:

$$C^s_{nk}(Q) \;=\; B^s_{nk} \;\simeq\; B^s_p \qquad (9)$$

Obviously, this cost is constant for any query, Q, and does not depend on the choice of the frame type (*clustered* or *unclustered*).

Now we will derive a formula for the block access estimation of FSF. Provided that the average value $w_{nk}(Q)$ of the number of "1" bits set by a given query Q in a qualifying frame (i.e. the frame query signature weight) is known, the expected number of "1" bits set in the h-bit suffix can be estimated as:

$$w_{nk}(Q,h) \;=\; \frac{w_{nk}(Q)h}{f_n} \qquad (10)$$

In the formula, h/f_n represents the probability that a bit, randomly chosen in the signature frame, lies in the suffix (assuming uniform distribution of "1" bits in the query signature).

The number of blocks accessed per qualifying partition can be evaluated, according to [21], as follows:

$$\begin{aligned} C^q_{nk}(Q) \;=\; & B^q_{nk} \\ & - \Big[\left(2B^q_{nk} - 2^{h_{nk}}\right) \left(1 - 2^{-w_{nk}(Q,h_{nk})}\right) \\ & + \left(2^{h_{nk}} - B^q_{nk}\right) \left(1 - 2^{-w_{nk}(Q,h_{nk}-1)}\right) \Big] \end{aligned} \qquad (11)$$

The next step involves estimation of the frame query signature weight, $w_{nk}(Q)$, which is the number of bits set to "1" in a non-zero frame of a query signature. However, this depends on the type of frame (*clustered* or *unclustered*).

4.2.1 Evaluation of $w_{nk}(Q)$ for unclustered frames

Let us suppose that $T(Q)$ is the number of terms (words) representing a query. Then, the query weight, $w(Q)$, which is the number of "1" bits, can be computed as follows:

- m/F is the probability that a given bit of the query signature is set to "1" by one term;
- $1 - m/F$ is the probability that a given bit is *not* set to "1" by one term;
- $(1 - m/F)^{T(Q)}$ is the probability that a given bit is *not* set to "1" by the superimposition of $T(Q)$ terms (assumed independent);
- $1 - (1 - m/F)^{T(Q)}$ is the probability that a given bit of the query signature is set to "1" by the superimposition of the $T(Q)$ terms.

Therefore, the expected number of bits set to "1" can be estimated as:

$$w(Q) \;=\; F\left[1 - \left(1 - \frac{m}{F}\right)^{T(Q)}\right] \qquad (12)$$

The number of frames hit, $H_{nk}(Q)$ — that is the number of frames in which bits with value "1" occur — can be evaluated as an application of Yao's formula [19], namely:

$$H_{nk}(Q) \;=\; n\left[1 - \frac{\binom{F - f_n}{w(Q)}}{\binom{F}{w(Q)}}\right] \qquad (13)$$

It estimates the number of frames hit in selecting, without replacement, $w(Q)$ distinct bits out of the total F bits grouped in n frames.

Finally, assuming a uniform distribution of "1" bits over the frames hit, the average number of bits set to "1" in every qualifying frame can be estimated as:

$$w_{nk}(Q) \;=\; \frac{w(Q)}{H_{nk}(Q)} \qquad (14)$$

4.2.2 Evaluation of $w_{nk}(Q)$ for clustered frames

The expected number of frames that qualify for a query composed of $T(Q)$ terms, provided that the frames are generated as clustered, can be computed as an application of Cárdenas's formula [2] in the following way:

$$H_{nk}(Q) \;=\; n\left[1 - \left(1 - \frac{1}{n}\right)^{T(Q)}\right] \qquad (15)$$

It gives the number of frames hit in selecting, with replacement, $T(Q)$ terms when every frame has a constant probability $1/n$ to be selected by a term. Assuming a uniform distribution of terms in the frames hit, the average number of terms hashed in one of the frames hit is:

$$T_{nk}(Q) \;=\; \frac{T(Q)}{H_{nk}(Q)} \qquad (16)$$

Finally, the average number of bits set to "1" in a frame hit of the query signature, can be computed with a formula quite similar to (12), namely:

$$w_{nk}(Q) \;=\; f_n\left[1 - \left(1 - \frac{m}{f_n}\right)^{T_{nk}(Q)}\right] \qquad (17)$$

n	k	B^s_{nk}	B^q_{nk}	O_{nk}
32	0	391	643	64.45%
16	1	388	578	48.97%
8	2	388	548	41.24%
4	3	389	534	37.28%
2	4	390	527	35.13%
1	5	390	524	34.36%

Table II: Space overhead versus n ($p = 32$ fixed).

The reason is that, within a frame of f_n bits, the query sets (with replacement) $T_{nk}(Q)$ times exactly m distinct bits to "1". Thus, m/f_n is the constant probability that a bit of the query signature frame is set to "1" by one of the $T_{nk}(Q)$ terms.

4.3 Storage space overhead

The introduction of quick filter implementation of partitions gives rise to storage space overhead because linear hashing lowers the block occupancy (down to $\alpha \simeq 0.75$) and because OIDs must be duplicated in signature frames. By contrast, with sequential partitions, signature frames are always identified by their position in a frame. On the other hand, the partition key and the block key suppressions in FSF reduce the storage space overhead.

For FSF, the percentage of storage space overhead, contrasted with the sequential organization of partitions, can be evaluated as:

$$O_{nk} = 100\,\frac{B^q_{nk} - B^s_{nk}}{B^s_{nk}} \tag{18}$$

$$\simeq 100\left[\frac{1}{\alpha}\left(1 + n\frac{d}{F}\right) - 1\right] \tag{19}$$

The result is obtained using B^s_p approximation for B_{nk}, neglecting the benefits of prefix and suffix suppressions and, finally, cancelling out the ceiling function in the B^s_p and B^q_{nk} expressions. Eq. (19) clearly shows that space overhead tends to grow linearly with the number of frames, provided that the parallelism degree, p, is fixed. Storage space requirements and exact overhead for the sample signature file are shown in Table II.

4.4 Simulation results

In this subsection, the simulation results are given. Queries with all the possible query weights, ranging from m to $F/2$ bits, are considered.

Figure 4 shows FSF performance with unclustered frames. For very low-weight queries, the performance is worse than the performance of VP, HP, or PP, mainly when the frame width is large. In such situation, a low number of bits is set to "1" in the frames hit, owing to the unclustered frame generation technique. For medium- or high- weight queries, however, performance improves significantly with any distribution of the parallelism between the number of frames and partitions.

Figure 5 shows the performance of FSF when using the clustered frames. For any (n, k) pair, it outperforms the previously analyzed unclustered frame technique. The reason is that, even for queries composed of few terms, at least m bits are set to "1" in a query signature frame hit, thus allowing greater selectivity of the quick filter mechanism. Moreover, if the number of available frames is increased, quick filter selectivity also increases, because frame signatures of reduced size are searched by query frames with the same minimum number of "1" bits. As a result, the traditional parallel signature file organizations can be outperformed by FSF with clustered frames even for very low weight queries. When the number of frames is over 16 (see Figure 5), FSF provides the best results in any case. However, the number of frames cannot be too high with respect to the signature size; otherwise, the signature frames become too small. In such a case, we might get a high probability of conflicts generated by the hashing function — which can cause overflows and slow down the retrieval — and a high probability of false drops. Further research should be made to throw light on this point and derive theoretical lower bounds for the frame width. In our example, the minimum width of frames is 64 bits, which seems a safe value to avoid hash conflicts ($2^{64} \gg N$ different addresses can be generated) and high false-drop probability (about four query terms must be hashed by the generation technique in the same clustered frame in order to set one half of the 64 bits to "1").

The simulation results comply very well with the theoretical conditions for good performance of partitioned signature file organizations. They suggest that in order to achieve good performance, the proportion of the signature size to the number of signatures should be as small as possible, and the query signature weight should be maximized. These findings have been reached independently in [20] and [9]. With clustered frames, such characteristics are always more closely approximated than in the single partitioned file implementation. The number of signatures remains constant, but the frame signature size is substantially smaller (depending on the number of frames used). In addition, because the generation of bits with value "1" is directed to specific frames, the density of "1"s in non-zero query signature frames is usually higher than the density of "1"s in the complete signature. Unclustered frames can never, in this respect, be better.

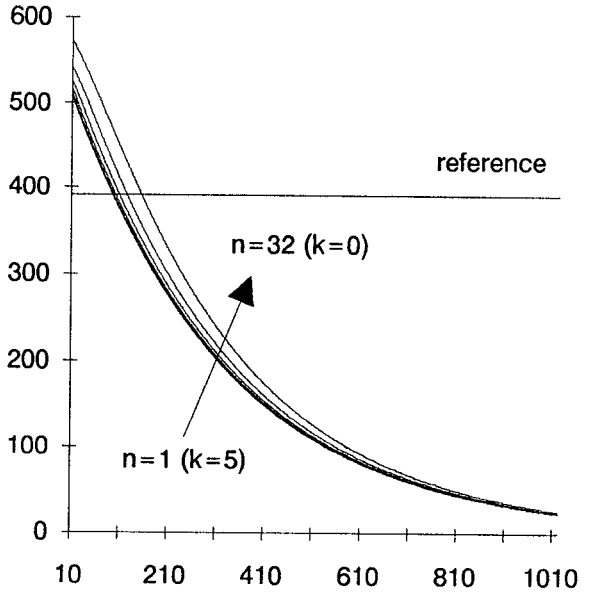

Figure 4: Query costs, $C^q_{nk}(Q)$, versus query weights, $w(Q)$, for $p = 32$ and $n = 1, 2, 4, 8, 16, 32$ in the presence of unclustered frames. The straight line represents the performance, $C^s_p(Q)$, of the reference solution.

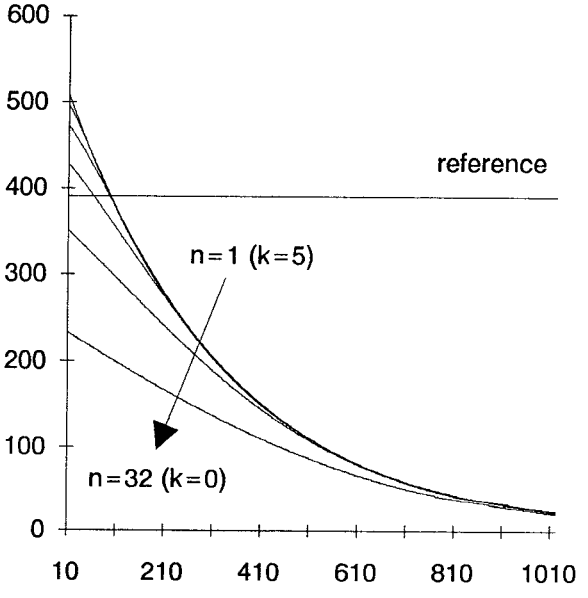

Figure 5: Query costs, $C^q_{nk}(Q)$, versus query weights, $w(Q)$, for $p = 32$ and $n = 1, 2, 4, 8, 16, 32$ in the presence of clustered frames. The straight line represents the performance, $C^s_p(Q)$, of the reference solution.

5 Conclusions

We have proposed a method that uses a new dynamic mixed fragmentation scheme in order to improve performance of signature file query processing. The method is supported by a parallel computer architecture so that adequate performance can be achieved for large and/or on-line applications.

The method is flexible in three dimensions — the amount of data, the number of frames, and the degree of parallelism in a frame — and is, in fact, a generalization of all previous approaches which have been proposed for signature file parallel processing. However, as the performance evaluation section indicates, this method outperforms the others, provided that the same level of parallelism is used. On the other hand, the flexibility of FSF offers a significant latitude for specific configuration designs. Optimization of this subject has been left open as a future research direction.

The hypothesis that the use of a partitioned approach in a sliced environment will stabilize performance for different query weights has only partially proved to be true. However, even though the performance is not constant (yet always better for the high-weight queries), it is much better compared to the performance of the sliced and partitioned methods alone.

There are also other advantages of FSF. Whenever the file size changes or the degree of parallelism in frames varies, reorganization is always localized into exactly two blocks or processing units. It does not seriously suffer from the data skew effect, and the load always stays very high. It it obvious that the method also allows intra-query parallelism. This issue would, however, require deeper analysis and evaluation. In this paper we leave the problem open and suggest it be considered as another future research direction.

Unreliability of computers in a parallel architecture is an important issue that must be seriously taken into account. In this respect, FSF can be considered as fault-tolerant, at least to a certain extent. Failure of a processing unit does not mean failure of the whole signature file organization. At first, the damaged processing unit may not be required for a given query. Moreover, even if it is required, only the false drops can increase. The consequence is that the performance may decrease, but the organization can still serve user queries correctly.

References

[1] Ahuja, S.R. and Roberts, C.S. "An Associative Parallel Processor for Partial Match Retrieval using Superimposed Codes". *Proceedings of the 7th Annual Symposium on Computer Architecture*

(France, May), IEEE, 1980, pp. 218-227.

[2] Cárdenas, A.F. "Analysis and Performance of Inverted Database Structures". *Communications of the ACM*, Vol. 18, No. 5 (May 1975), pp. 253–263.

[3] Faloutsos, C. and Christodoulakis, S. "Description and Performance Analysis of Signature File Methods for Office Filing". *ACM Transactions on Office Information Systems*, Vol. 5, No. 3, (July 1987), pp. 237-257.

[4] Faloutsos, C. "Signature-Based Text Retrieval Methods: A Survey". *Data Engineering Bulletin* (special issue on document retrieval), IEEE Computer Society, Vol. 13, No. 1 (March 1990), pp. 25-32.

[5] Hua, K.A. and Lee, C. "Handling Data Skew in Multiprocessor Database Computers Using Partition Tuning". *Proceedings of the 17th International Conference on Very Large Data Bases VLDB '91* (Barcelona, Spain, Sept.), VLDB Endowment, 1991, pp. 525-535.

[6] Khoshafian, S.N. and Copeland, G.P. "Object Identity". *Proceedings of the 1st ACM Symposium on Object Oriented Programming Systems, Languages and Applications OOPSLA '86* (Portland, Oregon, Oct.), ACM, 1986.

[7] Lakshmi, S. and Yu, P.S. "Effects of Skew on Join Performance in Parallel Architectures". *Proceedings of International Symposium on Databases in Parallel and Distributed Systems* (Austin, Texas, Dec.), 1988, pp. 107-117.

[8] Lee, D.L. "A Word-Parallel, Bit-Serial Signature Processor for Superimposed Coding". *Proceedings of the 2nd International Conference on Data Engineering* (Los Angeles, California, Feb.), IEEE, 1986, pp. 352-359.

[9] Lee, D.L. and Leng, C. "Partitioned Signature Files: Design Issues and Performance Evaluation". *ACM Transactions on Office Information Systems*, Vol. 7, No. 2 (Apr. 1989), pp. 158-180.

[10] Lee, D.L. and Leng, C. "A Partitioned Signature File Structure for Multiattribute and Text Retrieval". *Proceedings of the 6th International Conference on Data Engineering* (Los Angeles, California, Feb.), IEEE, 1990, pp. 389-397.

[11] Litwin, W. "Linear Hashing: A New Tool for Files and Table Addressing". *Proceedings of 6th International Conference on Very Large Data Bases VLDB '80* (Montreal, Canada, Aug.), VLDB Endowment, 1980, pp. 212-223.

[12] Meghini, C. and Thanos, C. "The Complexity of Operations on a Fragmented Relation". *ACM Transactions on Database Systems*, Vol. 16, No. 1, March 1991, pp. 56-87.

[13] Pogue, C. and Willett, P. "Use of Text Signatures for Document Retrieval in a High Parallel Environment". *Parallel Computing*, Vol. 4, 1987, pp. 259-268.

[14] Rabitti, F. and Zezula, P. "A Dynamic Signature Technique for Multimedia Databases". *Proceedings of 13th ACM International Conference on Research and Development in Information Retrieval SIGIR '90* (Brussels, Belgium, Sept.), 1990, pp. 193-210.

[15] Roberts, C.S. "Partial Match Retrieval via the Method of the Superimposed Codes". *Proceedings of the IEEE*, Vol. 67, No. 12 (Dec. 1979), pp. 1624-1642.

[16] Stanfill, C. and Kahle, B. "Parallel Free-Text Search on the Connection Machine System". *Communications of the ACM*, Vol. 29, No. 12 (Dec. 1986), pp. 1229-1239.

[17] Sung, Y.Y. "Parallel Searching for Binary Cartesian Product Files". *Proceedings of the Annual Computer Science Conference CSC '85* (New Orleans, March), ACM, 1985, pp. 163- 172.

[18] Tiberio, P. and Zezula, P. "Selecting Signature Files for Specific Applications". *Proceedings of the 5th Annual European Computer Conference IEEE-CompEuro '91* (Bologna, Italy, May), IEEE, 1991, pp. 718-734.

[19] Yao, S.B. "Approximating Block Accesses in Database Organizations", *Communications of the ACM*, Vol. 20, No. 4 (Apr. 1977), pp. 260–261.

[20] Zezula, P. "Linear Hashing for Signature Files". In *Network Information Processing Systems* (Boyanov, K. and Angelinov, R. eds.), Elsevier (North-Holland), IFIP, 1989, pp. 243-250.

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