Hamming Filter: A Dynamic Signature File Organization for Parallel Stores*

Pavel Zezula
Technical University of Brno - Czech Republic
zezula@cis.vutbr.cs

Paolo Ciaccia and Paolo Tiberio
DEIS - University of Bologna - Italy
{paolo,tiberio}@deis64.cineca.it

Abstract

Partitioning, in general, has become the basic strategy for organizing data files to avoid an exhaustive search when executing queries. However, hardware limitations that constrain the performance of query execution mainly become a problem for partial-match queries, where the size of the result can equal the size of the data file. In such situations, a proper application of parallelism can bring the required breakthrough in performance. Hamming Filter is a parallel, partitioned organization of signature files that are stored in fixed size buckets with a guaranteed load and is based on the idea of *linear code decomposition*. It can efficiently manage dynamic data files by means of a partitioned structure that always grows and shrinks linearly and is appropriate to multidimensional partitioning and searching. This paper proves that the organization yields no expected execution skew for partial-match queries, provided the data is not skewed and the degree of parallelism is a power of two.

1 Introduction

Users of computer systems can nowadays benefit from a considerable performance improvement achieved in the speed of processors. However, the bottleneck for applications with high I/O activities still remains the performance of secondary storage that has only been improving at a modest rate.

Recently, several attempts have been made to design computer architectures capable of meeting the requirements of applications with high I/O activities, such as Multiprocessor Database Computers [10], client-server systems [3], and disk arrays [17]. As claimed in [3], these mechanisms form an excellent environment for distributed, parallel database technology.

A common implementation problem of all parallel designs is the strategy for data placement and retrieval that should fully exploit capabilities offered by the hardware. This problem involves distributing data over several parallel hardware units (e.g. disks that can be accessed in parallel). In general, good partitioning must avoid: (1) *data skew* - much more data is placed in one partition than in the others, and (2) *execution skew* - execution time in one partition is much higher than in the others.

Three basic approaches to partitioning appear in the literature: *range* partitioning, *round-robin* partitioning, and *hashed* partitioning. Range partitioning maps continuous attribute ranges of a file to various partitions. Round-robin partitioning maps the i-th record to partition $i \bmod n$. Hashed partitioning maps each record to a disk location based on a hash function.

Recently, a new declustering method for parallel disk systems called Coordinate Modulo Distribution (CMD) has been proposed [15]. Experimental results show that the method achieves near optimum performance for range queries, provided the distribution of data on each dimension is stationary. Kamel and Faloutsos [11] proposed a parallel organization of R-trees [9] to design a server for spatial data, maximizing the throughput of range queries.

The difficult problem of *multi-attribute file* disk allocation for efficient execution of *partial-match* queries has been extensively studied for *Cartesian product*

*This work has been partially supported by the ESPRIT Basic Research Action Project No. 6309, FIDE2 (Formally Integrated Data Environment), and by the *Logidata* and *Multidata* Projects of the Italian National Research Council (C.N.R.).

**Proceedings of the 19th VLDB Conference
Dublin, Ireland, 1993.**

files. Recent works have successfully applied a coding-theoretic approach for allocating buckets on disks [1, 6, 7]. Even though there is no strictly optimal allocation method for distributing a Cartesian product file with n attributes over p disks, for arbitrary values of n and p, the theory of error-correcting codes has shown to provide schemes which are very close to the optimal case. However, since in the above designs the number of buckets, as determined by the partitioning scheme, is *a priori* fixed, problems arising from files dynamically changing in size have not been considered.

In this paper, we concentrate on partitioning for parallel processing of *signature files* [5]. We consider signature files as a way to tackle the problem of multidimensional partitioning and searching. A signature file is basically a compressed version of a corresponding data file. Even though a signature file typically causes information loss (i.e. the compression is nonreversible), it is suitable for searching due to its simple structure. Signature files contain randomized multidimensional data and, depending on the signature extraction method used, can eliminate non-uniform data term occurrences and query frequencies [4, 14]. We are also convinced that ongoing research effort in this field will bring even better signature extraction designs than we can apply today.

Many techniques have been proposed to improve the search performance of signature files in a single-processor environment. For a survey, see for example [14]. The general idea is to avoid searching the whole signature file by using a nonsequential storage structure, typically by applying a *tree* or a *hashing* technique. There have also been attempts to make the schemes run in parallel environments [8, 12, 21]. However, these designs apply a simple horizontal and/or vertical signature file partitioning, thus resulting in parallel organizations which can rarely avoid execution skew. The negative fact that some (very often, many) processors are not activated for a specific query is relieved by introducing *inter-query* parallelism, that is, by using the not active processors for executing other queries, if possible. In this paper, on the other hand, we are aiming at maximizing *intra-query* parallelism, where elimination of the execution skew is the main objective.

In general, provided a file is formed by a set of directly accessible buckets containing data objects (records, signatures), there are two ways to improve query execution performance: (1) *minimize the number of accessed buckets*, and (2) *use parallelism*. In other words, the less buckets from the total number of existing buckets that are accessed, the shorter the processing time. Naturally, if the needed buckets are accessed in parallel, performance can improve further.

Such a strictly performance-oriented view of query processing requires a simultaneous application of two, seemingly adverse, data placement strategies: *clustering* and *declustering*. Retrieved records should be clustered in buckets so the minimum number of buckets is accessed. On the other hand, accessed buckets should be declustered uniformly on all partitions, and in this way, the power of parallel processing is maximized.

Signature files are advantageous not only because they represent pieces of simple, structured, binary, and randomized data but also because many techniques exist for their organization and management. Two specific ideas have become fundamental for our work:

- The dynamic partitioning technique to cluster signatures in fixed-size partitions known as *Quick Filter* [23];
- The method for declustering binary data based on *linear error correcting codes* [6].

Our design, called *Hamming Filter*, is a new dynamic organization for signature files processed in parallel environments. It demonstrates very good performance for any partial-match query because it has practically no execution skew, provided the data is not skewed. Performance of the organization improves with the signature file size and the degree of parallelism.

The rest of the paper is organized as follows. In Section 2, we review the underlying concepts of our design. In Section 3, we present the Hamming Filter, and Section 4 contains its performance evaluation.

2 Background

We have built our design on top of the following two ideas: *Quick Filter* and *linear error correcting codes*. In the following, we will survey their most essential concepts.

2.1 Quick Filter Organization of Superimposed Signatures

This subsection provides the most essential information concerning the superimposed signature file coding method, partitioned organizations of signatures, and the main characteristics of the Quick Filter method.

2.1.1 Superimposed Coding

For the sake of simplicity, we assume that each data object O_i $(i = 1, \ldots, N)$ in the file is represented by a

superimposed object signature, S_i, with length f, the *signature size*. The number of bits with value "1" is the *signature weight*. An object signature is generated by superimposing (OR-ing) signatures of *terms* that form the object. A *term signature* is obtained by hashing the term onto an f-bit vector so exactly m bits are set to "1".

When objects are searched for a term or a set of terms, a *query signature*, Q, is generated from the user's query in the same way as described for an object signature above. The object signature, S_i, (*signature* for short, from now on) satisfies query signature Q iff for all bit positions of Q that are set to "1", the corresponding bit positions in S_i are also set to "1"(the *inclusion condition*). Then, we say S_i *includes* Q. Formally, the set of qualifying signatures is defined:

$$\{S_i \mid S_i \textbf{ AND } Q = Q\}$$

where bitwise **AND** is used. Such a way of generating superimposed signatures is also known as the *fixed-size block* (FSB) method.

A new way of generating signatures, the *fixed-weight block* (FWB) method, was introduced recently in [14]. The main objective of FWB is to provide an optimum method of assigning weights to document terms based on their occurrence and query frequencies. As a result, the FWB method accounts for both uniform and nonuniform occurrence and query frequencies, the signature weight is constant, and the *false-drop* probability (i.e. the probability that a signature qualifies while the actual object does not) is lower than for the FSB method. However, the storage overhead of FWB is slightly higher.

2.1.2 Partitioned Signature Files

The basic idea of any signature file *partitioning scheme* is simple: while storing a file that contains N signatures, *similar* signatures are grouped in partitions, and when a query is issued, some of the partitions need not be accessed because they cannot contain qualifying signatures. Signatures stored in *partition* P_j ($j = 1, \ldots, b$) are characterized by having the same *signature key*, KP_j, with *key size* (length), l, where $l \leq f$. Obviously, KP_j is also the *key of partition* P_j. When a query signature Q is received, its key, denoted KQ, is obtained by extracting a substring of l bits from Q at the same position where the keys of the object signatures were obtained. The partition P_j is activated (i.e. accessed) iff KP_j includes KQ. The set of partitions activated by Q is thus defined:

$$\{P_j \mid KP_j \textbf{ AND } KQ = KQ\} \qquad (1)$$

Let $w(\cdot)$ denote the function that returns the *weight* of its bit vector argument, that is, the number of bits set to "1". Then, regardless of the method used to define signature keys, the basic factor influencing the performance of partitioned signature file organizations is clearly the query signature key weight, $w(KQ)$.

2.1.3 Quick Filter

The *Quick Filter* (QF) method – also called the *Dynamic Suffix* method because the keys are defined as suffixes – partitions a signature file according to a split hash function that uses the signature's suffix as its argument. *Linear hashing* [16] has been employed as the storage structure in [23]. Such organization makes the signature (and partition) key size, l, closely related to the number of partitions, b. More precisely, in any QF file, $2^{l-1} < b \leq 2^l$ holds, and the keys of $2^l - b$ partitions have size $l - 1$, whereas the keys of the other $2b - 2^l$ partitions have size l. For a given number of partitions, b, the signature key size, l, can be computed as $l = \lceil \log_2 b \rceil$. As a consequence of the linear hashing implementation, each of the partitions can have entries in an *overflow area*.

Quick Filter is a fully dynamic, partitioned, signature file organization with a controlled load and a small space overhead. Partitions, in fact, are physical *buckets* of secondary memory (e.g. blocks or pages). Maintenance is handled easily, and the file can both *grow* and *shrink* linearly. Performance is dependent on query signature weight; QF works much better for high-weight queries, but even an exhaustive search – it is used when query signatures with zero weights in their suffixes are executed – is comparable to a search in the sequential file organization. QF is also more efficient for large rather than for small files. Use of underlying storage structures other than linear hashing is discussed in [23].

The performance of some basic methods for partitioning of signature files, including Quick Filter, can be easily evaluated by means of a closed approximated formula that introduces no appreciable errors for all practical purposes [2]. The key performance parameter is called the *bucket activation ratio* (BAR), which is defined as the ratio of the number of buckets activated by a query (i.e. those buckets that must be searched) to the total number of buckets. The formula for estimating BAR follows:

$$BAR(w(Q)/f, l) = \left(1 - \frac{w(Q)}{2f}\right)^l \qquad (2)$$

2.2 Binary Linear Error Correcting codes

Linear codes are used in *Information and Coding Theory* to detect and correct errors while transmitting data. It has been shown by examples and verified by simulation in [6] that linear codes can be useful for building a declustering scheme for binary Cartesian product files. The method has been mathematically analyzed and extended to general (i.e., not only binary) Cartesian product files in [7]. More recently, Abdel-Ghaffar and El Abbadi have also considered the use of nonlinear codes and investigated the optimality ranges for the different alternatives [1]. It should be remarked that our framework differs from that considered in [1, 6, 7] because partial-match queries on signature files are actually *inclusion* queries (see Eq. (1)), and the buckets considered for storage have a fixed size and a controlled load.

In the following, the most essential principles of linear codes related to our paper are summarized. More information can be found in any book dealing with Information and Coding Theory, e.g. [18, 20].

A *linear binary code*, C, is a subspace of the vector space, $\{0,1\}^n$, of all bit strings of length n. If C is a subspace of dimension k, we speak about a linear $C(n,k)$ code. Every binary $C(n,k)$ code consists of 2^k *codewords*, where each of the codewords has k *information* and $m = n - k$ *check bits*. The *minimum distance* of code $C(n,k)$ is the minimum number of different bits (i.e., Hamming distance) between any two codewords.

Each linear binary $C(n,k)$ code can be described by a system of $m = n - k$ homogeneous linear equations. The matrix, $\mathbf{H}$, of this system is called the *check matrix* of the code. If $\mathbf{H}$ is the check matrix of a linear $C(n,k)$ code, then any codeword $W = (w_1, w_2, \ldots, w_n)$ is a solution of the system $\mathbf{H} \cdot W^T = 0^T$, where 0 is the zero binary vector of dimension m (upperscript T denotes vector transposition). In general, for every word $W \in \{0,1\}^n$, we define the word $Y = (y_1, y_2, \ldots, y_m)$ as $\mathbf{H} \cdot W^T = Y^T$. The word Y is called the *syndrome* of W. Because the dimension of Y is m, the number of possible syndromes is 2^m.

The idea of declustering binary data by means of linear codes can be formalized by the following definition.

Definition 2.1 (Linear code decomposition) *Linear code decomposition of the linear space $\{0,1\}^n$ is a decomposition of the set of 2^n binary words of length n into 2^m pair-wise disjoint groups by using the $C(n,k)$ linear code check matrix $\mathbf{H}$. Each of the groups contains the 2^k words with the same syndrome and with the minimum distance given by the linear code $C(n,k)$.*

The basic idea is that, by putting together signatures with the same syndrome (i.e. signatures with high distance), not many signatures in a specific group should qualify for a given query, and many groups, if not all, should contain qualifying signatures.

In the following we consider, without loss of generality, the specific case of Hamming codes, that are linear codes defined as follows.

Definition 2.2 (Hamming code) *A binary linear code is a (perfect) Hamming code if its check matrix is formed by columns of all possible $2^m - 1$ non-zero binary words of length m (with none of them repeated). Hamming code $C(n,k)$ is a perfect code that can be defined for any number of check bits, $m \geq 2$, with $n = 2^m - 1$ and $k = n - m$. The Hamming distance between any two codewords of the code is $d \geq 3$.*

Example 2.1 Consider the Hamming code $C(7,4)$, that is, $m = 3$, for which the check matrix $\mathbf{H}$ can be written as:

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Then, the code, i.e. the set of words with syndrome 0, includes the following $2^k = 16$ words:

(0000000)	(0100101)	(1000011)	(1100110)
(0001111)	(0101010)	(1001100)	(1101001)
(0010110)	(0110011)	(1010101)	(1110000)
(0011001)	(0111100)	(1011010)	(1111111)

If we complement the first bit of the above codewords, we get the words with syndrome $Y = (011)$. In general, by complementing the i-th bit, we get words whose syndrome Y is the i-th column vector of matrix $\mathbf{H}$ ($i = 1, 2, \ldots, n$). □

3 Hamming Filter

Hamming Filter is a dynamic organization of superimposed signatures distributed over a number of disks that can be processed in parallel. It is designed for a partitioned storage structure, defined as follows:

Definition 3.1 (Partitioned Storage Structure) *$PSS(p, b_0, b_1, \ldots, b_{p-1})$ is a Partitioned Storage Structure that consists of p parallel partitions where the i-th partition is formed by b_i buckets. A bucket consists of one or more physical pages (blocks) and contains signatures (a horizontal fragment of the signature file).*

Buckets have a fixed size but can be connected with an overflow area. Each signature is stored in exactly one bucket. Buckets are directly accessible and, once accessed, are searched entirely (i.e. exhaustive search is used in buckets). The number of partitions, $p > 0$, is constant for a given store while the number of buckets, $b_i > 0$, can change in time. Let $B_{i,j}$ designate the j-th bucket of the i-th partition, where $i = 0, 1, \ldots, p-1$ and $j = 0, 1, ..., b_i - 1$.

Hamming Filter is based on the idea of *horizontal fragmentation* of the signature file and is, in fact, a generalization of the Quick Filter method to a multiple-disk environment. The main difference is that the partitioning space is two-dimensional. Whereas in the Quick Filter, the partitions are basically the access units of the secondary store (blocks, pages), partitions of the Hamming Filter are sets of such units, conveniently called *buckets*. Then, the mapping into partitions is performed by using the principle of linear code decomposition (see Definition 2.2) while the mapping into buckets within partitions is performed by means of the Quick Filter.

3.1 Fragmentation Scheme

The number of partitions p is related to the Hamming code parameter m in the following way:

$$p = 2^m = n + 1$$

Signatures are assigned to buckets of PSS in two steps. In the first step, signatures are declustered, and the identification number of the partition where a signature will be stored is determined as a function, $PI(S, p)$, of the signature, S, and the number of partitions, p. In the second step, similar signatures in a partition are clustered and the identification number of the bucket for storing a specific signature is determined by the function $BI(S, p, b_i)$. In the following, we will discuss both steps in detail. Finally, the address, $Addr(S)$, for storing the signature S within PSS, is obtained by the function $I(S, PSS)$, which returns the ordered pair of partition and bucket identification numbers:

$$Addr(S) = I(S, PSS) = (PI(S, p), BI(S, p, b_i)) = (i, j)$$

3.1.1 Partition Assignment

For a given signature $S = (c_1, c_2, \ldots c_f)$, let W be its n-bit suffix, that is, $W = S^{[n]} = (c_{f-n+1}, c_{f-n+2}, \ldots, c_f)$. Then, the partition index, i, is determined by the formula:

$$i = PI(S, p) = [(\mathbf{H} \cdot W^T)^T]_2$$

This formula multiplies the matrix $\mathbf{H}$ by the transposed vector W^T, considers the (transposed) result vector as a binary number from the set $\{0, 1, \ldots, 2^m - 1\}$, and determines the partition index. Thus, a signature is allocated in the i-th partition if and only if its suffix yields a syndrome with decimal value i. It follows that all the signatures in i-th partition have distance $d \geq 3$. In order to illustrate the partition assignment function, we designate the result of the multiplication as $Y = (y_1, y_2, \ldots, y_m)$. Then, we can write $\mathbf{H} \cdot W^T = Y^T$, where

$$y_z = \sum_{x=1}^{n} h_{z,x} w_x, \quad z = 1, 2, ..., m$$

All arithmetic operations are taken modulo 2.

Example 3.1 Consider Hamming code $C(7, 4)$ (see Example 2.1), and suppose that the n-bit suffix of signature S is $S^{[n]} = (1001001)$. Then:

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

The decimal value of the binary transposed result is $(101)_2 = 5$, meaning that any signature with the suffix (1001001) will be stored in the 5-th partition. □

3.1.2 Bucket Assignment

Signatures are assigned to buckets using the Quick Filter principle in the following way. The last m bits of a signature belonging to the i-th partition are ignored, and only the remaining $k = n - m$ information bits of the suffix are retained. Such a definition of the signature key guarantees the uniform occurrence distribution inside a partition because all possible 2^k configurations of bits can in such area occur. By contrast, this would not be true if all n bits were considered (see Example 2.1).[1]

Let $l_i = \lceil \log_2 b_i \rceil$. Then, the l_i-bit suffix is used as an argument to store the signature in a bucket of the i-th partition. The bucket index, j, in partition i is determined as:

[1] In order to guarantee that all the 2^k addresses are generated, the submatrix consisting of the last m columns of $\mathbf{H}$ has to have m distinct rows. For Hamming codes, the simplest way to achieve this is to have the last m columns of $\mathbf{H}$ forming an identity matrix.

$$j = BI(S, p, b_i) =$$
$$= \begin{cases} \sum_{x=0}^{l_i-1} c_{f-x-m} 2^x & \text{if } \sum_{x=0}^{l_i-1} c_{f-x-m} 2^x < b_i \\ \sum_{x=0}^{l_i-2} c_{f-x-m} 2^x & \text{otherwise} \end{cases}$$

When $b_i = 1$, then $l_i = 0$, and in this case, $BI(S, p, b_i) = 0$ results.

Example 3.2 Consider the n-bit suffix $S^{[n]} = (1010001)$, and a number of partitions $p = 8$. Suppose the number of buckets in the i-th partition is $b_i = 10$. It follows that the maximum bucket key size in partition i is $l_i = \lceil \log_2 b_i \rceil = \lceil \log_2 10 \rceil = 4$ and the size of the ignored suffix of the signature needed to determine accessed buckets is $m = log_2 p = log_2 8 = 3$. Then, $BI((\ldots 1010001), 8, 10) = 2$. The explanation is easy. By taking 4 bits, $l_i = 4$, we get $(1010)_2 = 10$. However, 10 is not smaller than $b_i = 10$, and therefore, only $l_i - 1$ bits must be considered, namely (010), which, as a binary number, gives 2. □

For large files and small values of m (i.e. $m < 5$), l_i is likely to exceed $n - m = k$. In this case, bucket assignment is based on one or more bits that are not used to determine the partition index; as a consequence, the number of buckets in the i-th partition is not limited to 2^k.

3.2 Query Processing

The task of the query processing algorithm is to determine addresses of buckets – the *Response Set* – that must be accessed to execute the query signature Q. It is a distinctive feature of the Hamming Filter that buckets are determined directly (i.e. without any additional access to the database). The query processing arguments are PSS and Q, and the algorithm proceeds as follows:

1. Generate all possible signature suffixes, W_x, of length n, which include $Q^{[n]}$:

$$(Q^{[n]} \textbf{ AND } W_x) = Q^{[n]}$$

2. Determine the Response Set, RS, for query signature Q as a set of bucket addresses:

$$RS = \bigcup_x I(W_x, PSS)$$

This set contains addresses of buckets that must be accessed to check their signatures for qualification.

3.3 Partitioned Store Evolution

By definition, PSS has two dimensions: *partition* and *bucket*. In a given store, the number of partitions is fixed (determined by the parameter p), and any requirement for a change in this dimension results in a complete reorganization. On the other hand, individual partitions are dynamic; the number of buckets can change, but it never becomes zero. Such a feature of the store enables dealing with dynamic signature files where the number and contents of signatures change in time.

The strategy for load balancing in PSS changes the number of buckets, which, even though overflow areas can be appended, have primary areas of a fixed size. We use the *linear hashing* principle [16], and the *hints* for store *growth* or *shrink* are the bucket *overflow* or *underflow*, respectively. The specific procedures for the i-th partition growth and shrink follow:

Partition Growth

1. *add a new empty bucket* B_{i,b_i};
2. $b_i = b_i + 1$;
3. $l_i = \lceil \log_2 b_i \rceil$;
4. $\forall S \in B_{i,b_i-1-2^{l_i-1}}$ *store the signature in the bucket at address* $I(S, PSS)$.

Partition Shrink

1. $l_i = \lceil \log_2 b_i \rceil$;
2. $\forall S \in B_{i,b_i-1}$, *put the signatures into the bucket* $B_{i,b_i-1-2^{l_i-1}}$;
3. $b_i = b_i - 1$;
4. *release the bucket* B_{i,b_i}.

The key point of the growth and shrink procedures is the management of added and removed buckets. In the Hamming Filter, a new bucket is always added at the end of a partition, and the content of the bucket with index $b_i - 1 - 2^{l_i-1}$ (called the *split pointer* in *linear hashing*) is divided to fill the new bucket. When a bucket should be removed from a partition, it is always the last one, and its contents are moved to the bucket determined again by the split pointer. The value of the split pointer increases linearly with the number of buckets, b_i.

4 Performance Analysis

In this section, we analyze performance of the Hamming Filter in terms of bucket accesses needed to perform a query. For the sake of simplicity, we consider that in any of the $p = 2^m$ partitions exactly 2^k buckets are allocated. We further suppose no data skew, which is (at least for large files) justified because the signatures are randomized pieces of data and the bucket assignment is, in fact, another randomization function imposed on the same data. On the other hand, execution skew is the main concern in our performance study of the query evaluation, for Hamming Filter response time is determined by the partition with the highest execution cost (i.e. the partition where the highest number of buckets are accessed).

We start with examples showing how the number of accessed buckets for a given query signature can be computed and then generalize the approach for computing the *worst-case* and estimating the *average-case* performance for groups of query signatures with a specific suffix weight. We also compare the performance with the optimum case, in which the same number of buckets in each partition is always accessed. Finally, we extend the analysis to random suffix weights as determined by the weight of the whole query signature.

4.1 Analytical Computation of the Number of Accessed Buckets

Given a query signature Q whose n-bit suffix has weight $w(Q^{[n]})$, then $2^{n-w(Q^{[n]})}$ out of 2^n buckets qualify. Unfortunately, this expression says nothing about the distribution of the accessed buckets on individual partitions – the execution skew. However, the execution skew is the crucial issue for any parallel system because the partition with the highest number of accessed buckets determines the total cost (i.e. response time).

The Hamming Filter computes the partition index (represented by the syndrome vector Y) for a signature by multiplying the parity check matrix $\mathbf{H}$ by the transposed signature suffix $W \equiv S^{[n]}$, specifically $\mathbf{H} \cdot W^T = Y^T$. Remember that each signature maps into exactly one partition. Now, let us consider a query signature Q, with suffix $Q^{[n]}$, and a specific partition represented by the syndrome Y. Then, $\mathbf{H} \cdot W^T = Y^T$ defines a set of m linear equations in the unknown W. Because W must include $Q^{[n]}$, the elements (i.e. the bits) of W corresponding to ones in $Q^{[n]}$ are fixed at "1", whereas the others are considered as variables. The number of variables, which depends on the weight of $Q^{[n]}$, is exactly $n - w(Q^{[n]})$. It is clear that this system of m linear equations can, in general, have more than one solution; each solution represents a qualifying bucket in the partition determined by the syndrome Y.

4.1.1 Some Examples

Consider the parity check matrix $\mathbf{H}$ for the $C(7,4)$ Hamming code and partition 0, represented by vector $Y = (000)$. To determine suffixes of signatures (bucket keys) in partition 0 that qualify for a query signature with suffix $Q^{[7]} = (1001001)$, the task is characterized by the following matrix equation:

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ w_2 \\ w_3 \\ 1 \\ w_5 \\ w_6 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

This can be transformed into the system of linear equations:

$$\begin{array}{lccccccccc} E1: & w_2 & + & w_3 & + & w_5 & + & 1 & = & 0 \\ E2: & & & w_3 & + & w_6 & + & 0 & = & 0 \\ E3: & & & & & w_2 & + & 1 & = & 0 \end{array}$$

We get three equations with four unique variables. Because the variables are binary and the equations *linearly independent* (due to the construction of $\mathbf{H}$), the system has $2^{4-3} = 2$ solutions. According to equation $E3$, $w_2 = 1$. It follows from equation $E2$ that w_3 and w_6 are simultaneously 0 or 1. Finally, $w_5 = 0$ (provided $w_3 = w_6 = 0$) or $w_5 = 1$ (provided $w_3 = w_6 = 1$). Our example, therefore, has the two solutions (1101001) and (1111111). Using the bucket assignment procedure, bucket indexes are then obtained by dropping the last $m = 3$ bits, thus yielding 1101 and 1111, respectively. Only these two buckets in partition 0 can contain qualifying signatures, and only they must be accessed during query execution. To compute the necessary bucket accesses of any other partition, the left-hand sides of equations $E1$, $E2$, and $E3$ remain the same, and only the right-hand sides change. It can be easily checked that for any other partition the result is still 2. No execution skew exists for our sample query.

If we change the query signature Q and suppose that $Q^{[7]} = (1101001)$, we get the following system of equations:

$$\begin{array}{lccccccc} E1: & w_3 & + & w_5 & + & 0 & = & y_1 \\ E2: & w_3 & + & w_6 & + & 0 & = & y_2 \\ E3: & & & & & 0 & = & y_3 \end{array}$$

Because there are no variables in $E3$, the system only has solution for $y_3 = 0$, implying that only the partitions with indexes $0 = (000)_2$, $2 = (010)_2$, $4 = (100)_2$, and $6 = (110)_2$ contribute to the Response Set. In these partitions, $2^{3-2} = 2$ buckets can contain qualifying signatures, for the system of equations induced by the query is reduced to two equations with three variables.

For this query, we have execution skew because partitions are divided in two groups. No buckets are accessed in partitions of the first group whereas two buckets are accessed in partitions of the second group. It follows that the query execution is again equal to the cost of accessing two buckets even though the weight has increased.

However, performance of the Hamming Filter does not depend merely on the number of "1"bits in the query signature suffix. This can be easily illustrated by considering another query signature suffix, $Q^{[7]} = (1110001)$. It has the same weight as in the previous example, but the system of equations is now:

$$\begin{array}{lccccccc} E1: & w_4 & + & w_5 & + & 0 & = & y_1 \\ E2: & w_4 & + & w_6 & + & 0 & = & y_2 \\ E3: & & & w_4 & + & 1 & = & y_3 \end{array}$$

A system of three equations with three variables has exactly one solution, which implies a single access in all partitions, i.e. no execution skew.

4.1.2 Generalization

According to the above discussion, we can precisely define the *response time*, $R(Q)$, of the Hamming Filter with respect to query Q.

Definition 4.1 *Given a query signature Q, whose n-bit suffix, $Q^{[n]}$, has weight $w(Q^{[n]})$, the response time, $R(Q)$, of the Hamming Filter is the maximum number of accessed buckets in a partition, expressed as:*

$$R(Q) = \begin{cases} 2^{V-E} & \text{if } V \geq E \\ 1 & \text{if } V < E \end{cases}$$

where V is the number of variables, $V = n - w(Q^{[n]})$, and E is the number of equations with at least one variable.

Let H_i designate the i-th row of the matrix $\mathbf{H}$ for $i = 1, 2, \ldots, m$. Then, the number of equations, E, can be computed:

$$E = \sum_{i=1}^{m} g_i(Q)$$

where

$$g_i(Q) = \begin{cases} 1 & \text{if } w((\mathbf{NOT}\ Q^{[n]})\ \mathbf{AND}\ H_i) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Two important observations concerning V and E can be made:

1. The number of variables only depends on the query signature suffix weight;
2. The number of equations with at least one variable depends on the specific setting of bits in the query signature suffix. Thus, for a given query signature suffix weight, we can have a varying number of equations, depending on the specific configuration of bits in the suffix. Note that $g_i(Q) = 1$ iff $Q^{[n]}$ does not include H_i. In this case, there exists at least one bit position where H_i has the value "1"and $Q^{[n]}$ has the value "0"; this ensures that the variable corresponding to such a position will appear in the equation.

4.2 Response Time for Query Signatures with Specific Suffix Weights

We first consider the problem of determining the *maximum response time*, designated R_w^+, over all query signatures Q whose suffix $Q^{[n]}$ has the weight $w = w(Q^{[n]})$:

$$R_w^+ = \max\{R(Q) : w(Q^{[n]}) = w\}$$

In order to derive an analytic expression for R_w^+, we take advantage of Definition 4.1 and write:

$$R_w^+ = 2^{V-E_w^-} = 2^{n-w-E_w^-}$$

where E_w^- is the *minimum number of equations* that can arise from a query signature suffix with weight w. Note that, in general, the number of equations can vary from 1 to m.

The following observation and lemma are fundamental for our purpose.

Observation 4.1 *Given the parity check matrix $\mathbf{H}$ of the $C(n,k)$ Hamming code, the bitwise* **OR** *of any r rows ($r = 1, 2, \ldots, m$) of $\mathbf{H}$ yields a binary vector whose weight is $2^m - 2^{m-r}$.*

For instance, for the $C(7,4)$ Hamming code, the **OR** of any $r = 2$ rows results in a vector with weight $2^3 - 2^{3-2} = 6$.

Lemma 4.1 *Consider the system of linear equations $\mathbf{H} \cdot W^T = Y^T$. Independently of the specific setting of bits in the query signature suffix, at least E_w equations with variables are obtained if the suffix query weight does not exceed the value*

$$w = n - 2^{E_w - 1} \qquad E_w \in \{1, 2, ..., m\}$$

Proof: To get E_w equations from $Q^{[n]}$, it is necessary that E_w rows of $\mathbf{H}$ are not included in $Q^{[n]}$. If w is not less than $2^m - 2^{m-r}$, then, according to Observation 4.1, there exists a query signature suffix $Q^{[n]}$ with weight w that can include r rows, $(r = 1, \ldots, m)$, thus leaving $m - r$ equations with variables. Consequently, $w = 2^m - 2^{m-r} - 1$ is the maximum weight that always guarantees at least $m - r + 1$ equations. The result follows after substituting E_w for $m - r + 1$ and n for $2^m - 1$. □

For instance, to get at least $E_w = 2$ equations with the $C(7,4)$ Hamming code, w must not exceed 5 because the **OR** of any $3-2+1 = 2$ rows of $\mathbf{H}$ has weight 6.

Theorem 4.1 (Maximum Response Time) *The maximum response time, R_w^+, for a query signature Q whose suffix $Q^{[n]}$ has weight w is calculated:*

$$\begin{aligned} R_w^+ &= 2^{V-E_w^-} = \\ &= 2^{n-w-\lfloor \log_2(n-w) \rfloor - 1} = \\ &= 2^{V-\lfloor \log_2 V \rfloor - 1} \end{aligned} \quad (3)$$

where:

$$E_w^- = \lfloor \log_2(n-w) \rfloor + 1$$

Proof: It follows from Lemma 4.1 after solving for E_w and taking the integer part of the result. □

Corollary 4.1 *If $V < 3$, then $R_w^+ = 1$.*

Proof: From Equation (3), we get $R_w^+ = 1$ if

$$V = \lfloor \log_2 V \rfloor + 1$$

Obviously, the equation only has a solution for $V = 1$ or 2, confirming Theorem 4.2 from [6], which states that declustering by the Hamming codes enjoys a response time of 1 for $w > n - 3$, that is, $V < 3$. □

4.2.1 Optimum Performance of Hamming Filter

The aim of this subsection is to investigate if and under which conditions the Hamming Filter behaves in the optimum way, producing no execution skew.

Definition 4.2 (Optimum Response Time) *The optimum response time, R_{opt}, for a query signature whose suffix has weight w is defined as the number of qualifying buckets, 2^{n-w}, divided by the number of partitions, $p = 2^m$:*

$$R_{opt} = \begin{cases} \dfrac{2^{n-w}}{2^m} = 2^{2^m-1-m-w} & \text{if } n - w \geq m \\ 1 & \text{otherwise} \end{cases}$$

w	R_w^+	R_{opt}
16	2048	1024
17	1024	512
18	512	256
19	256	128
20	128	64
21	64	32
22	32	16

w	R_w^+	R_{opt}
23	16	8
24	16	4
25	8	2
26	4	1
27	2	1
28	2	1

Table 1: Example of the range of query signature suffix weights in which the Hamming Filter cannot guarantee optimum response time for the case $m = 5$.

From this definition, we can derive the following basic results concerning the Hamming Filter's worst-case performance.

Corollary 4.2 *The worst-case response time of the Hamming Filter is equal to the optimum response time, $R_w^+ = R_{opt}$, if $E_w^- = m$.*

Proof: Immediate from Theorem 4.1 and Definition 4.2. □

Theorem 4.2 (Optimality) *The worst-case response time of the Hamming Filter is equal to the optimum response time if one of the following conditions on the query signature suffix weight is satisfied:*

$$R_w^+ = R_{opt} \quad \text{if} \quad w > n - 3 = 2^m - 4 \quad (4)$$

$$\text{or} \quad w < 2^{m-1} \quad (5)$$

Proof: The case $w > n-3$ has already been proven to lead to optimum response time in Corollary 4.1. The case $w < 2^{m-1}$ follows from Lemma 4.1 and Corollary 4.2 because the maximum value of w that always guarantees $E_w^- = m$ equations is $n - 2^{m-1} = 2^{m-1} - 1$. □

To summarize, the Hamming Filter is optimum if $w > n-3$ or if less than approximately 50% of the bits in the query signature suffix are set to 1.[2] Note that, when $m = 2$, Hamming Filter performance is *always* optimum and gives response 1 for any query signature with $w > 0$. Furthermore, in the case $m = 3$, the only value of w for which optimality cannot be guaranteed is $w = 4$. In the case $m = 5$, the values of w for which optimality cannot be guaranteed are considered in Table 1. Although a precise characterization of the likelihood of Hamming Filter suboptimal behavior is given in the next section, it is important to notice that the case $w \geq 2^{m-1}$ is not very likely to occur because:

[2] More precisely, the percentage is $2^{m-1}/(2^m - 1) * 100$.

1. The weight of the whole query signature (not just of its suffix) never exceeds $f/2$ and, in practical cases, is much lower [22];

2. The bits set to "1"in the signature query are uniformly distributed over the f positions; therefore, it is very unlikely that more than 50% of the n suffix bits are set to "1"(see Section 4.3).

4.2.2 Average Performance of Hamming Filter

The maximum response time, R_w^+, for a query signature whose suffix has weight w can be quite higher than the optimum response time, R_{opt}, as shown in Table 1, but the average case is much better. For instance, when $m = 3$ and $w = 4$, for which $R_w^+ = 2$ and $R_{opt} = 1$, only 3 out of $\binom{7}{4} = 35$ queries with suffix weight $w = 4$ yield a response time of 2. In this case, the *average response time* is only 1.08.

The average response time, designated $\overline{R}_w$, is computed by considering all query signatures whose suffix has weight w and depends on the distribution of the response time, $R(Q)$, over the set of such queries. To characterize this distribution, the related problem of determining how many queries with a given suffix weight w lead to exactly E equations with variables $(E = 1, \ldots, m)$ can be considered.

Lemma 4.2 *The probability that a query signature whose suffix has weight w, and thus $V = n - w$ "0"s, leads to exactly E equations is:*

$$\Pr\{E\} = \frac{\binom{m}{E}}{\binom{2^m - 1}{V}} \sum_{j=1}^{E} (-1)^{E-j} \binom{E}{j} \binom{2^j - 1}{V} \tag{6}$$

Proof: Given in [24]. □

Lemma 4.3 *The number of equations produced by a query signature whose suffix has weight w is a random variable with the expected value:*

$$\overline{E}_w = m \left(1 - \frac{\binom{n - 2^{m-1}}{n - w}}{\binom{n}{n - w}} \right) \tag{7}$$

Proof: Consider the i-th row, H_i, of the check matrix. It yields an equation with variables iff there exists at least one bit position where H_i has the value "1"and $Q^{[n]}$ has the value "0". The number of different $Q^{[n]}$ with weight w is $\binom{n}{n-w}$. Because the weight of H_i is 2^{m-1}, there are exactly $\binom{n-2^{m-1}}{n-w}$ query suffixes such that their $n - w$ "0"s do not correspond to any "1"of H_i. The result then follows.[3] □

Theorem 4.3 (Average Response Time) *The average response time, $\overline{R}_w$, for a query signature whose suffix has w "1"s and $V = n - w$ "0"s, given the condition that $w \leq n - m$, which implies $V \geq m$, is calculated:*

$$\overline{R}_w = \frac{2^{V-m}}{\binom{2^m - 1}{V}} \sum_{j=1}^{m} \binom{m}{j} \binom{2^j - 1}{V} \tag{8}$$

Proof: It suffices to consider the expression of $\Pr\{E\}$ given by Eq. (6). Due to the condition $V \geq m$, it is:

$$\overline{R}_w = \sum_{E=1}^{m} 2^{V-E} \Pr\{E\} \tag{9}$$

and the result follows after some combinatorial manipulation. □

The condition $V \geq m$ allows 2^{V-E} to be used in place of $\lceil 2^{V-E} \rceil$ in Eq. (9), and thus to derive a simpler formula for the average response time. Note, however, that this leaves out from analysis only $\lceil m - 3 \rceil$ cases from the total number of V cases and, consequently, $\lceil m - 3 \rceil$ values of w (e.g. $w = 27$ and 28 in the case $m = 5$). As Figure 1 indicates, $\overline{R}_w$ is very close to the optimum response time, R_{opt}, and in the worst case ($w = 26$), overhead is about 8%. On the other hand, in all the other cases, it is completely negligible.

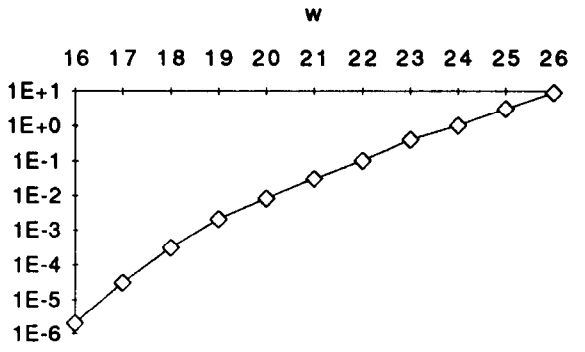

Figure 1: Average percentage execution overhead, that is $(\overline{R}_w - R_{opt})/R_{opt} * 100$, of Hamming Filter for $m = 5$. Only the values of $w \leq n - m = 26$ for which $R_w^+ > R_{opt}$ are considered.

[3] The result can also be derived by explicitly computing $\sum_E E \Pr\{E\}$.

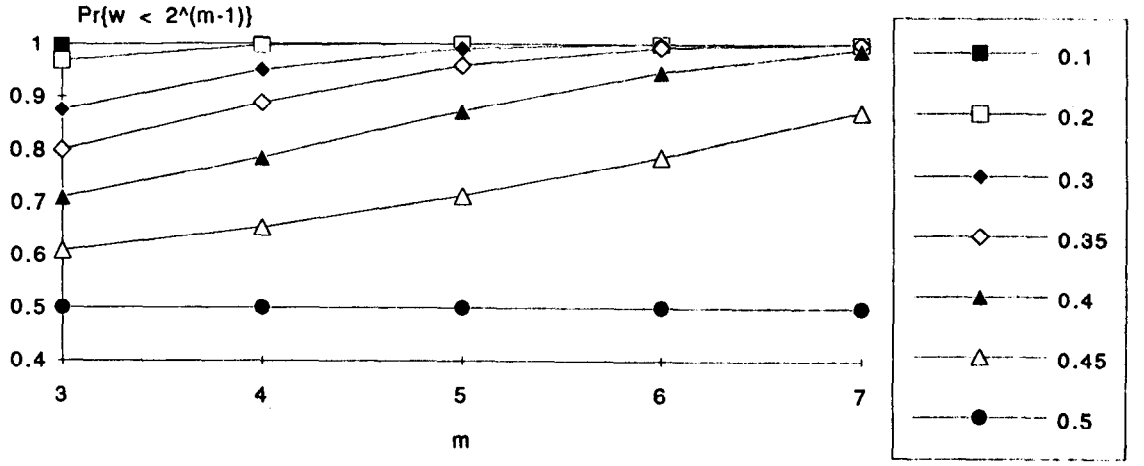

Figure 2: Probability that the suffix weight, w, is less than 2^{m-1}, as a function of m. Different curves correspond to different values of the ratio $w(Q)/f$.

4.3 Performance for a Given Query Signature Weight

It is a well-known fact that query signatures cannot have arbitrary weights. Theoretically, the weight of a specific query, $w(Q)$, cannot be greater than one half of its size, f, because this is the weight of the object signatures. However, the weights of query signatures are usually much lower. Intuitively, the query signature suffix weight should also be quite low and, consequently, the performance of Hamming Filter optimum.

According to [2], the weight of a suffix of length n is a random variable, w, which follows a hypergeometric distribution that can be reliably approximated by the binomial distribution:

$$\Pr\{w\} = \binom{n}{w} \left(\frac{w(Q)}{f}\right)^{w} \left(1 - \frac{w(Q)}{f}\right)^{n-w}$$

For a given value of the signature weight, $w(Q)$, the cumulative probability that the suffix weight is less than 2^{m-1} and, consequently, that performance is optimum can be computed. The most unfavorable case is, intuitively, when $w(Q) = f/2$. In this case, we have:

$$\Pr\{w\} = \binom{n}{w} 2^{-n} = \binom{2^m - 1}{w} 2^{-2^m+1}$$

Therefore, the probability that $w < 2^{m-1}$ equals:

$$\Pr\{w < 2^{m-1}\} = \sum_{w=0}^{2^{m-1}-1} \binom{2^m - 1}{w} 2^{-2^m+1} = \frac{1}{2}$$

When $w(Q) < f/2$, this probability grows as shown in Figure 2. The result strongly supports confidence in optimum, or very near to optimum, performance of the Hamming Filter because typical signature file applications have $w(Q) \ll f/2$.

A second kind of analysis can be done by considering the expected number of equations with variables, $\overline{E}$, resulting for a given signature weight. We have proven in Corollary 4.2 that Hamming Filter performance is optimum when the number of such equations is m, so a high value of the ratio $\overline{E}/m$ indicates that the average performance is very close to the optimum.

Given by definition:

$$\overline{E} = \sum_{w} \overline{E}_w \Pr\{w\}$$

then, taking advantage of Equation (7) and the binomial approximation, we obtain:

$$\overline{E} = m\left(1 - \left(\frac{w(Q)}{f}\right)^{2^{m-1}}\right) \qquad (10)$$

In the most unfavorable case, specifically $w(Q)/f = 0.5$:

$$\overline{E} = m\left(1 - 0.5^{2^{m-1}}\right)$$

Apart from the case $m = 2$ (where the performance of Hamming Filter is always optimum), $\overline{E}$ is very close to m (for $m = 3$, $\overline{E}/m = 0.9375$) as Figure 3 shows. It should be observed that the ratio $\overline{E}/m$ is an increasing function of m and, consequently, of the degree of parallelism.

In order to evaluate the average response time of the Hamming Filter, it suffices to observe:

$$\overline{R} = \sum_{w} \overline{R}_w \Pr\{w\} \qquad (11)$$

Relative performance can be evaluated with respect

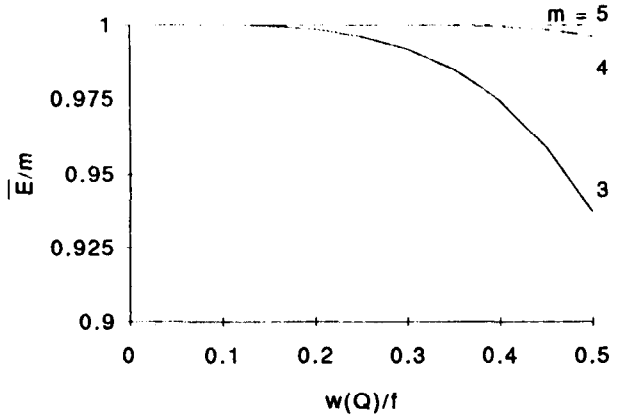

Figure 3: The ratio $\overline{E}/m$ as a function of the ratio $w(Q)/f$ for different values of parameter m.

to the optimum response time, $R_{opt}(w(Q))$, for a given query signature weight. Given 2^n buckets, $R_{opt}(w(Q))$ is the expected number of accessed buckets in each partition, assuming no execution skew. By means of Eq. (2) that defines the bucket activation ratio (BAR), optimum response time can be evaluated:

$$R_{opt}(w(Q)) = BAR(w(Q)/f, n)\ 2^k = \left(1 - \frac{w(Q)}{2f}\right)^n 2^k$$

Figure 4 plots the percentage of execution overhead of the Hamming Filter, that is $(\overline{R} - R_{opt}(w(Q)))/R_{opt}(w(Q)) * 100$. The overhead is less than 10% even in the extreme case when $w(Q)/f = 0.5$ and $m = 3$. For $m > 3$, the overhead never exceeds 0.1% and, in most cases, is some orders of magnitude lower.

5 Conclusions

The Hamming Filter is a partitioned parallel organization for signature files. Signatures are stored in fixed size buckets of parallel partitions by means of a dynamic fragmentation scheme. The bucket assignment and the query execution are performed in a direct access mode (i.e. without any additional access to auxiliary data). The Hamming Filter can be considered as an extension of the Quick Filter by the application of the principle of linear code decomposition. In this way, the desired data distribution effect of simultaneous declustering and clustering of signatures has been achieved in a single design. However, the Hamming Filter is not only a fragmentation scheme for signatures but also a complete integrated organization for parallel secondary stores.

The main original contributions of the paper include:

- design of the dynamic partitioning scheme for a persistent store consisting of a specific number of parallel partitions containing a variable number of storage buckets;
- design of the query processing and the store evolution algorithms able to provide high performance for retrieval and control of data load at a very low cost;
- performance analysis:
  - analytical computation of the number of accessed buckets;
  - explicit formulas for determining the maximum, optimum, and average response time for query signatures with specific suffix weights;
  - analytical formalization of the query signature weight effect on the performance of the Hamming Filter.

It has been proven that for most of the possible queries, optimum performance can be guaranteed. Provided $w > 2^m - 4$, the response time is exactly 1. However, the optimum performance is also achieved if $w < 2^{m-1}$. The average performance is (if not optimum) always very near to the optimum, and queries for which the optimum performance cannot be guaranteed are not likely to occur. The relative execution overhead is a decreasing function of the code parameter m, and thus, the performance of Hamming Filter improves with the degree of parallelism.

In our design we have only considered Hamming codes. However, the extension to other linear error correcting codes is straightforward in that it suffices only to consider a different parity check matrix. Our analytical results are valid for codes with minimum distance $d = 3$. If codes with larger distance are used, an even better performance can be expected.

Our future plans will extend the analysis to the effect of using other linear codes. A proper comparison of the analytical results with a prototype system that we have just implemented will be done, and will allow us to consider the effects of data skew on the performance. Then, we want to concentrate our research effort on application of the Hamming Filter in other data environments, namely complex object stores combining attribute and reference data. The possibility of managing parallel environments where the number of disks is not limited to a power of 2 will also be investigated.

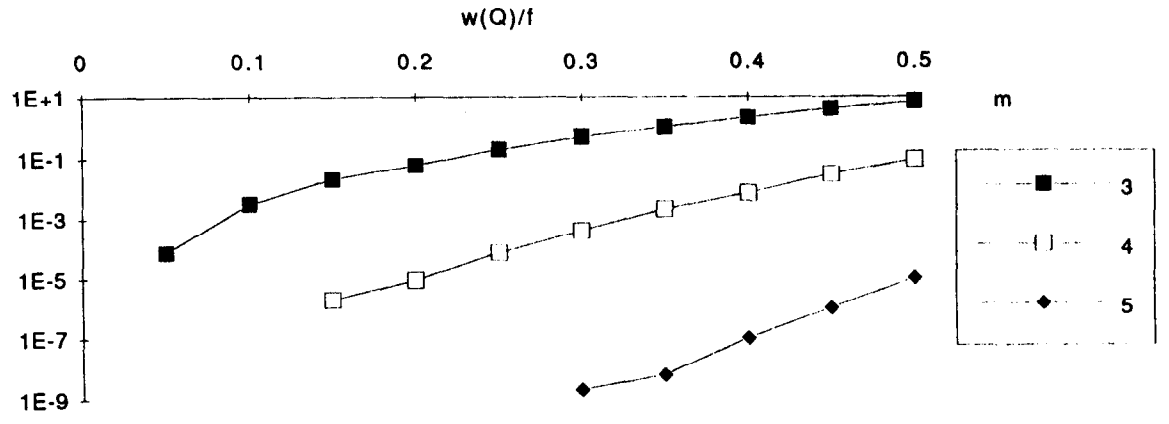

Figure 4: Average percentage execution overhead as a function of the ratio $w(Q)/f$ for different values of parameter m.

References

[1] Abdel-Ghaffar, K. A., and El Abbadi, A. "Optimal Disk Allocation for Partial Match Queries". *ACM TODS*, Vol. 18, No. 1, March 1993, pp. 132-156.

[2] Ciaccia, P., and Zezula, P. "Estimating Accesses in Partitioned Signature File Organizations". *ACM TOIS*, Vol. 11, No. 2, April 1993, pp. 133-142.

[3] DeWitt, D., and Gray, J. "Parallel Database Systems: The Future of High Performance Database Systems". *Comm. of ACM*, Vol. 35, No. 6, June 1992, pp. 85-98.

[4] Faloutsos, C., and Christodoulakis, S. "Design of a signature file method that accounts for non-uniform occurrence and query frequencies". *Proc. of the 11th VLDB Conf.*, Stockholm, Sweden, August 1985, pp. 165-170.

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

[6] Faloutsos, C., and Metaxas, D. "Declustering Using Error Correcting Codes". *Proc. of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Philadelphia, Pennsylvania, March 1989, pp. 253-258.

[7] Fujiwara, T., Ito, M., Kasami, T., Kataoka, M., and Okui, J. "Performance Analysis of Disk Allocation Method Using Error-Correcting Codes". *IEEE Trans. on Information Theory*, Vol. 37, No. 2, March 1991, pp. 379-384.

[8] Grandi, F., Tiberio, P., and Zezula, P. "Frame-Sliced Partitioned Parallel Signature Files". *Proc. of the 15th ACM SIGIR.*, Copenhagen, Denmark, June 1992, pp. 286-297.

[9] Guttman, A. "R-trees: a dynamic index structure for spatial searching". *Proc. of ACM SIGMOD*, Boston, Massachussets, June 1984, pp. 47-57.

[10] Hua, K.A., and Lee, C. "Handling Data Skew in Multiprocessor Database Computers Using Partition Tuning". *Proc. of the 17th VLDB*, Barcelona, Spain, September 1991, pp. 525-535.

[11] Kamel, I., and Faloutsos, C. "Parallel R-trees". *Proc. of ACM SIGMOD*, San Diego, California, June 1992, pp. 195-204.

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

[13] Lee, D.L., and Leng, C.-W. "Partitioned Signature Files: Design Issues and Performance Evaluation". *ACM TOIS*, Vol. 7, No. 2, April 1989, pp. 158-180.

[14] Leng, C.-W., and Lee, D.L. "Optimal Weight Assignment for Signature Generation". *ACM TODS*, Vol. 17, No. 2, June 1992, pp. 346-373.

[15] Li, J., Srivastava, J., and Rotem, D. "CMD: A Multidimensional Declustering Method for Parallel Database Systems". *Proc. of the 18th VLDB*, Vancouver, Canada, August 1992, pp. 3-14.

[16] Litwin, W. "Linear Hashing: A New Tool for Files and Table Addressing". *Proc. of the 6th VLDB*, Montreal, Canada, August 1980, pp. 212-223.

[17] Patterson, D.A., Gibson, G., and Katz, R.H. "A Case for Redundant Arrays of Inexpensive Disks (RAID)". *Proc. of ACM SIGMOD*, Chicago, Illinois, June 1988, pp. 109-116.

[18] Peterson, W.W., and Weldon, E.J. "Error Correcting Codes". *MIT Press*, 1972.

[19] Rabitti, F., and Zezula, P. "A Dynamic Signature Technique for Multimedia Databases". *Proc. of the 13th ACM SIGIR*, Brussels, Belgium, September 1990, pp. 193-210.

[20] Reza, F.M. "An Introduction to Information Theory". *McGraw-Hill*, 1961.

[21] Stanfill, C., and Kahle, B. "Parallel Free-Text Search on the Connection Machine System". *Comm. of ACM*, Vol. 29, No. 12, December 1986, pp. 1229-1239.

[22] Tiberio, P., and Zezula, P. "Selecting Signature Files for Specific Applications". *Proc. of the 5th IEEE Computer European Conference*, Bologna, Italy, May 1991, pp. 718-734.

[23] Zezula, P., Rabitti, F., and Tiberio, P. "Dynamic Partitioning of Signature Files". *ACM TOIS*, Vol. 9, No. 4, October 1991, pp. 336-369.

[24] Zezula, P., Ciaccia, P., and Tiberio, P. "Hamming Filter". Technical Report No. 92, CIOC-CNR, Bologna, January 1993.