International Journal of Information Technology

Vol. 8, No. 1 August 2002

A Dynamic Signature File Declustering Method based on the Signature Difference

Jae Ryong Shin, Chung Beom Son, Jae Soo Yoo, *Byoung Mo Im

Dept. of Computer and Communication
Chungbuk National University
48 Gaesindong Cheongju, Chungbuk, Korea, 361-763

{jrshin, cbson}@netdb.chungbuk.ac.kr
yjs@cbucc.chungbuk.ac.kr
ibm@dbserver.kaist.ac.kr

Abstract

For processing a signature file in parallel, an effective signature file declustering method is needed. The Linear Code Decomposition Method(LCDM) used for the Hamming Filter may give a good performance in some cases, but due to its static property, it fails to evenly decluster a signature file when signatures are skewed. In addition, it has other problems such as limited scalability and non-determinism. In this paper we propose a new signature file declustering method, called the Inner-product method, which overcomes those problems in the LCDM. The Inner-product method declusters a signature file dynamically based on the signature difference which is computed by using signature inner product. We show through the various experiments that the Inner-product method outperforms the LCDM under various data workloads.

Keywords Declustering Method, Signature File, Parallel Database, Dynamic Index Structure, Information Management

I. INTRODUCTION

Information retrieval and management systems that have been a major field of computing for a long time typically deal with not only formatted data but also unformatted data. A storage organization widely advocated for unformatted data as well as formatted one is to use the signature file method[1]. In general, each document signature that is an element of the signature file is constructed from word signatures by using the superimposed coding method. When processing a query, the signature file is scanned in advance and many non-qualifying documents are discarded. Figure 1 illustrates the construction of a document signature using superimposed coding method, where a document consists of three words, "Database", "Parallel" and "Information". Here, a signature length is twelve and the number of bits that are set to '1' in a word signature is two.

Document D = (Database, Parallel, Information)

Keywords	Word Signature
Database	0110 0000 0000
Parallel	0000 1000 0001
Information	0001 0001 0000
Document Signature	0111 1001 0001

Fig. 1. Document signature construction using superimposed coding

Since the size of a signature file is much smaller than that of a data file, it has been shown that the signature file can effectively work as a filter that immediately discards most non-qualifying documents for a given query[2]. Although sequential organization of a signature file works well for a data file with a small size, its performance becomes a problem when the size of a data file is large. Other organizations of a signature file can improve their performance based on a tree or hashing techniques. The bit-sliced signature file[3], and the frame-sliced signature file[4] have been proposed for static environment, while the S-tree[5], the Quick filter[6, 7, 8] and the HS file[9] have been proposed for dynamic environment.

There have also been many attempts to make the schemes run for parallel environment. The Fragmented Signature File(FSF)[10], Key-Based Partition Method[11] and the Hamming Filter[8, 12] partition a

signature file to process a query in parallel. The Hamming Filter shows good declustering performance for some partial match queries. It declusters a signature file by using the Linear Code Decomposition Method (LCDM) that is used for detecting and correcting errors while transmitting data[13, 14]. The LCDM yields good declustering performance in some cases, but due to its static property, it fails to evenly decluster a signature file when signatures are skewed. In addition, it has other problems such as limited scalability and non-determinism. The MIN-entropy is better than the LCDM because the MIN-entropy method statically declusters signature file based on statistic information of previously allocated signatures. However, MIN-entropy is not suitable to dynamic environment of large scale parallel database systems where insertions happen frequently.

In this paper we propose a new signature file declustering method, called Inner-product that overcomes the problems in the LCDM and the MIN-entropy. The Inner-product method declusters a signature file by using signature inner-product. The signature inner-product of two signatures is a scalar value, where each signature is considered as a bit string. In this method, we compare the degree of differences between a new signature and representative signature of each node. The Inner-product method allocates a new signature to processing node with minimum among the inner products of the new signature and the representative signature of each node.

The LCDM is based on a static information that does not reflect the current status of signature allocation. On the other hand the Inner-product method declusters signature files dynamically based on the current status of signature allocation. Thus, Inner-product method can cope with a variety of workloads and configurations. We show through performance evaluation based on the statistical modeling that the Inner-product method gives better retrieval performance than the LCDM for data sets with various distributions such as uniform distribution, normal distribution and exponential distribution. We also address the signature insertion time by using asymptotic notation. It shows that Inner-product method works well in a dynamic environment where insertions occur frequently.

The rest of the paper is organized as follows. In Section II, we review various parallel signature file organizations. Section III presents our proposed Inner-product method and Section IV shows experimental results. Finally, Section V gives concluding remarks.

II. SIGNATURE FILES FOR PARALLEL PROCESSING

Parallel systems are increasingly being used for high performance applications that require efficient access to large amounts of data, e.g., large-scale transaction processing, decision-support systems and

multimedia systems. Three system architectures for multiprocessor database computers have been proposed: Shared Memory(SE), Shared Disk(SD), and Shared Nothing(SN)[15]. There have been considerable debates on which architecture is the most suitable for a database management system implementation. While the coherency control problem limits the number of processors in both SE and SD systems, the modular design of SN architecture enables incremental growth and scalability to hundreds of nodes. Thus most large-scale parallel processing systems for information processing are based on the SN architecture. However, SN systems are very sensitive to the distribution of data on disks (i.e., data declustering) that may lead to the data skew problem. To avoid the data skew problem, a good declustering method is necessary.

There have been many attempts to make signature files run for parallel environments such as the Fragmented Signature File(FSF), the Key-Based Partition Method(KBPM) and the Hamming Filter[10, 11, 12]. They speed up the search time on signature files by distributing signatures to disks that participate in parallel processing.

Table 1. Classification of Parallel Signature Files

Categories	Partitioning Methods	Declustering Methods
Parallel Signature Files	Key-Based Partition Method Fragmented Signature File	Hamming Filter

We classify these parallel signature file organizations into two categories according to the way they distribute a signature file to disks: partitioning methods, and declustering methods. Table 1 illustrates these two categories[20]. Now we survey and analyze parallel signature files focussing on the data distribution scheme.

1. Partitioning Methods

This section explains methods that partition a signature file into sets of signature subfiles or sets of signature frames. The signature subfile is a partition of a signature file when the latter is partitioned horizontally. On the other hand, the signature frame is a vertical partition of signature file. In general, signatures in each signature subfile have a common key. Given M processors and a signature file is partitioned to N partitions, if M is equal to N, then each partition is allocated to each processing node directly. However, if M is greater than N, then additional partition allocation method is needed.

In [10] three key-based partition methods are presented. These Key-based partition methods partition a signature file horizontally. They partition signatures with same signature key into the same partition and

allocate them to the same processing node. But they are different in the way the key is selected. (1) *Fixed prefix method* selects keys with a fixed length and fixed starting position. (2) *Extended prefix method* selects the keys with variable length but fixed starting position. (3) *Floating key method* selects the keys with fixed length but variable starting position. Although the above three key-based partition methods can be used for intra-query parallelism, they are much more suitable to inter-query parallelism.

The FSF uses mixed partition scheme. By vertical partition, it decomposes 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 subfiles, where all signatures in each subfile have a common signature key. This kind of partition can be seen as a product derived from global signature file in two consequent steps, involving vertical and horizontal partition.

Figure 2 shows the mixed partition scheme that produces disjoint data partitions vertically and horizontally. These disjoint partitions are distributed to the given processing nodes. Although this method allows intra-query parallelism, it is more suitable to inter-query parallelism because, for a signature query, qualified partitions are only the subset of all partitions.

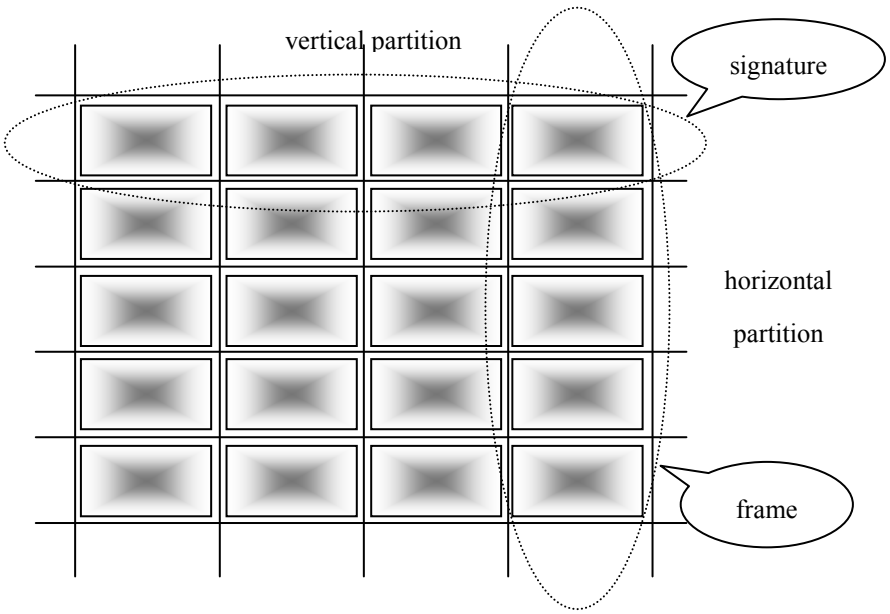

Fig. 2. The mixed partition scheme

2. Declustering Methods

The partition methods apply a simple horizontal and/or vertical signature file partitioning. Therefore, some partitions that are not activated for a specific query is relived by introducing inter-query parallelism,

that is, without using the active partitions for executing other queries, if possible. The declustering methods are aiming at maximizing intra-query parallelism, where elimination of the execution skew is the main objective. The simplest declustering method is randomly distributing signatures to processing nodes. The Hamming Filter decomposes a signature file horizontally through the Linear Code Decomposition Method(LCDM) that declusters signature file by using linear code. Also, by using the dynamic partitioning technique in fixed-size partitions, known as Quick Filter, clusters of signatures are allocated to local processing node. Therefore, the Hamming Filter can be considered as an extension of the Quick Filter by the application of the principle of linear code decomposition. The LCDM uses this *syndrome* property of the linear code[16]. Figure 3 shows that the LCDM decomposes linear space $\{0,1\}^n$ into $2^{\{n-k\}}$ $C(n,k)$ that have the same syndrome respectively. The codewords with the same syndrome are uniformly distributed by the execution load for non-skewed data, because the Hamming distance is guaranteed between its codewords. The Hamming Filter declusters a signature file by applying LCDM to the suffix of each signature.

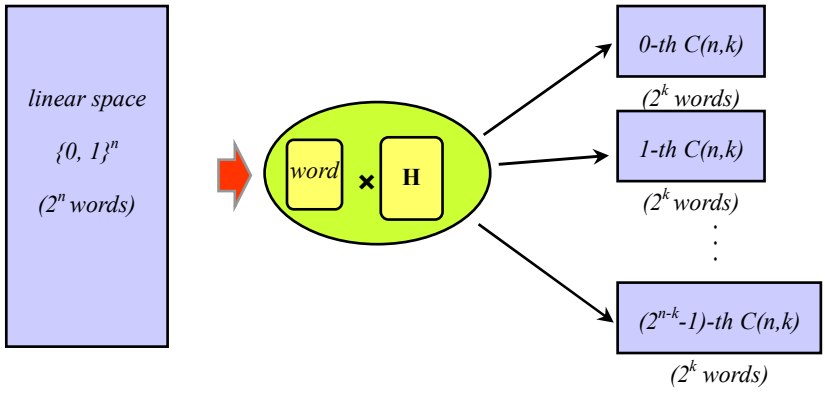

Fig. 3. Linear Code Decomposition Method

For intra-query parallelism, it is required that good declustering algorithm should avoid data skew and execution skew. The LCDM has no execution skew for non-skewed data. However, since the LCDM allocates signatures with the same suffix into the same processing node, it cannot avoid data skew if many signatures have the same suffix. In addition, it has the following problems that make the LCDM difficult for parallelism. First, it is not gracefully scalable to the number of processing nodes. The method is properly defined only when the number of processing nodes is a power of two. This is a serious restriction because most current parallel systems in practice may not have a power of two numbers of processing nodes. Second, it is not deterministic algorithm. For the suffix with the *m*-size, *m!* matrix positions can be checked for the LCDM to decluster a signature file. While the choice of the check matrix can affect the declustering performance of the LCDM, no appropriate guideline has been provided to choose an optimal

check matrix. Finally, the LCDM may lead to information loss because it only uses partial information, i.e., the suffix of the signature.

The MIN-entropy that uses signature-entropy as a signature declustering measure was proposed in [20]. This method assumes Shared Nothing (SN) parallel architecture as a base platform for large parallel signature file organizations. To minimize a signature query processing time in the SN parallel environment, data need to be declustered evenly and independently to all processing nodes. Note that intra-query parallelism is inevitable to minimize the signature query response time in the SN parallel environment.

III. A NEW DECLUSTERING METHOD FOR PARALLEL SIGNATURE FILE

In this section, we propose a new signature file declustering method, called Inner-product. We assume a SN parallel architecture as a base platform for large parallel signature file organizations. To minimize a signature query processing time in the SN parallel environment, data need to be declustered evenly and independently to all processing nodes. Note that intra-query parallelism is inevitable to minimize the signature query response time in the SN parallel environment. In a SN Parallel system with p parallel processing units, the response time to a signature query is defined as **max**$\{C_1, C_2, \ldots, C_p\}$, where $C_i (1 \le i \le p)$ is the response time (i.e., the cost measured as the number of physical page accesses) of the *i-th* processing unit. More specifically C_i is the signature search time plus the false drop time at the *i-th* processing node as shown in the Figure 4. Thus, the objective is to find a declustering scheme such that for any query, **max**$\{C_1, C_2, \ldots, C_p\}$ is the minimum among all possible signature file declusterings. Given that the above declustering problem is *NP-complete*[16, 17], our method heuristically declusters signature files based on the statistical information.

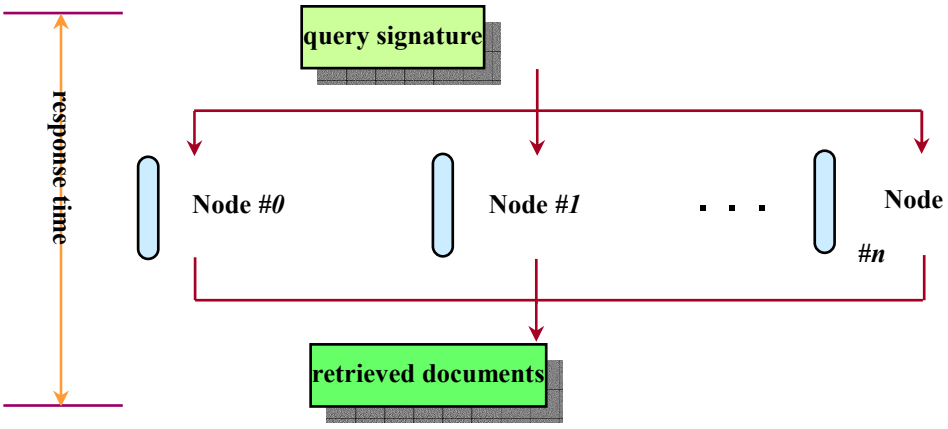

Fig. 4. Response time of a signature query

As basic criteria, to minimize a signature query response time, the following two skews have to be avoided in a (1) data skew i.e., much more data is placed in one fragment than in the others, and (2) execution skew i.e., execution time in one fragment is much higher than in the others. Before describing our method, it is necessary to introduce some notations and definitions.

1. Definition and Terminology

Notation

- n, the size of signature
- m_i, the number of signatures in the node i
- $c_{ik} = \sum_{j=1}^{m_i} s_{ijk}$, the k-*th* vertical bit sum of all signatures($1 \le j \le m_i$) in the node i
- s_{ijk}, the k-*th* bit of the j-*th* signature in the node i

Definition 3.1 Two signatures are *completely different* if all elements of two signatures are different.

Definition 3.2 (*count vector*) The count vector of node i that is denoted by $cv_i = <c_{i1}, c_{i2}, \ldots, c_{ij}, \ldots, c_{in}>$ is the vertical bit sum vector of all signatures previously allocated to the node i. The count vector construction procedure is presented in Figure 5

$$
\begin{array}{rr}
 & 1\,0\,0\,1\,0\,1\,0\,1\,1\,1\,0\,0\,0\,0\,0\,1 \;=\; s_{i1} \\
 & 1\,0\,1\,0\,0\,0\,1\,1\,1\,1\,1\,1\,0\,0\,0\,0 \;=\; s_{i2} \\
+ & 0\,0\,1\,0\,1\,1\,0\,0\,1\,0\,1\,0\,0\,1\,0\,1 \;=\; s_{i3} \\
\hline
 & <2,0,2,1,1,2,1,2,3,2,2,1,0,1,0,2> \;=\; cv_i
\end{array}
$$

Fig. 5. The Count Vector of i-th node

Definition 3.3 (*unit signature*) The unit signature of node i that is denoted by $uv_i = <u_{i1}, u_{i2}, \ldots, u_{ij}, \ldots, u_{in}>$ represents the status of signature allocation of node i, which is normalized count vector by '0' or '1'. For example, u_{i1} is '1' when c_{i1} is greater than the *mean* of cv_i. Otherwise u_{i1} is '0'.

Definition 3.4 (*inner product of two signatures*) The inner product of two signatures, s_i, s_j that is denoted by $s_i \bullet s_j$ is the sum of products of pair elements in two signatures. For example, given s_i = (1 0 0 1 0 1 0 1 1 1 0 0 0 0 0 1), s_j = (1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 0), $s_i \bullet s_j$ = 1*1 + 0*0 + 0*1 + 1*0 + 0*0 + 1*0 + 0*1 + 1*1 + 1*1 + 1*1 + 0*1 + 0*1 + 0*0 + 0*0 + 0*0 + 1*0 = 4.

Lemma 3.1 The inner product of two same signatures is the sum of element of the signature as follows:

$$s_i \bullet s_i = \sum_{k=1}^{n} s_{ik}$$

Proof. If $s_i = (s_{i1}, s_{i2}, \ldots, s_{in})$, then $s_i \bullet s_i = s_{i1} * s_{i1} + s_{i2} * s_{i2} + \ldots + s_{in} * s_{in} = s_{i1} + s_{i2} + \ldots + s_{in} = \sum s_i$. Because $s_{ij} * s_{ij} = s_{ij}$ *for all j where* $0 \le j \le n$. The reason is that s_{ij} is binary number.

Definition 3.5 (*complementary signatures*) The complementary signatures mean that the values of each bit of two signatures are different each other.

For example, we call the following signatures s_i and s_j the complementary signatures.

$$s_i = (1,0,0,1,1,1,1,0),\ s_j = (0,1,1,0,0,0,0,1)$$

Lemma 3.2 The inner product of two complementary signatures is 0 as follows:

$$s_i \bullet s_j = 0$$

Proof. Given $s_i = (s_{i1}, s_{i2}, \ldots, s_{in})$ and $s_j = (s_{j1}, s_{j2}, \ldots, s_{jn})$, then $s_i \bullet s_j = s_{i1} * s_{j1} + s_{i2} * s_{j2} + \ldots + s_{in} * s_{jn} = 0 + 0 + \ldots + 0 = 0$, because $s_{ik} * s_{jk} = 0$ *for all k where* $0 \le k \le n$. The reason is that either s_{ik} or s_{jk} is 0, because each bit of s_i and s_j is complementary.

Theorem 3.1 Given two signatures, s_i and s_j,

$$0 \le s_i \bullet s_j \le \mathbf{min}\{\sum s_i, \sum s_j\}$$

Proof. We prove the theorem by showing that the following three mutually exclusive cases are satisfying the above expression.

case 1) $s_i = s_j$

then by **Lemma 3.1** $s_i \bullet s_j = \sum_{k=1}^{n_i} s_{ik}$. Therefore it satisfies the theorem.

case 2) s_i, s_j are complementary.

then by **Lemma 3.2** $s_i \bullet s_j = 0$. Therefore it satisfies the theorem.

case 3) s_i, s_j are different.

i) $0 \le s_i \bullet s_j$: It is trivial because $s_i \bullet s_j$ is sum of products that is equal to or greater than 0.

ii) $s_i \bullet s_j \leq \min\{\sum s_i, \sum s_j\}$: let $s_i = (s_{i1}, s_{i2}, \ldots, s_{in})$, $s_j = (s_{j1}, s_{j2}, \ldots, s_{jn})$ then $s_i \bullet s_j = s_{i1} * s_{j1} + s_{i2} * s_{j2} + \ldots + s_{in} * s_{jn}$. If the number of elements set to 1 of s_i is less than that of s_j then $s_i \bullet s_j$ does not exceed the number of elements set to 1 in s_i. On the contrary, if the number of elements set to 1 of s_j is less than that of s_i, then $s_i \bullet s_j$ does not exceed the number of elements set to 1 in s_j. Thus it satisfies the theorem also.

Theorem 3.2 If 1's positions within a signature s_i are more different from those of signature s_j than those of signature s_k, then $s_i \bullet s_j < s_i \bullet s_k$.

Proof. The inner product of two signatures increases only when two signatures have element with 1 in the same position. Since s_i and s_j have less same 1's positions than s_i and s_k, then $s_i \bullet s_j < s_i \bullet s_k$.

For example, let's consider the following three signatures. As shown in this example, we easily find that 1's positions of s_i are more different from those of signature s_i than signature s_k. We can compute that $s_i \bullet s_j$ is 0 and $s_i \bullet s_k$ is 2. As a result, we find $s_i \bullet s_j < s_i \bullet s_k$.

$s_i = (1,1,1,1,0,0,0,0)$, $s_j = (0,0,0,0,1,1,1,1)$, $s_k = (0,0,1,1,1,1,0,0)$

2. Basic Concepts

The following observations are the bases on which our proposed signature file declustering method is developed. In parallel environment, the response time of a signature query can be minimized when the execution loads are uniformly distributed to all processing nodes. Here, the execution load is closely related with the number of bits that are set to '1' in signatures because we do not need to compare the bits set to '0' when processing a signature query. Therefore, signature file should be declustered based on the bits set to '1'. With theseconsiderations , we maintain p count vectors, where p is the number of all processing nodes. The count vector of *i-th* node, cv_i, is the vertical sum of signatures previously allocated to the *i-th* node as defined in **Definition 3.2**. The cv_i roughly denotes a distribution of the potential execution load for node i. The basic idea of our method is to minimize the variance of elements in count vectors of each processing node. By doing this, qualified signatures for any query can be evenly declustered to all the processing nodes. We propose a new heuristic method to minimize the variance of count vectors, called the Inner-product. The Inner-product method uses the *unit signature* that is a normalized count vector.

3. The Inner-product Method

In this section, we describe our proposed signature file declustering method, called Inner-product method. The Inner-product method is a kind of greedy method, which finds a feasible solution of minimum inner product in each allocation step. It declusters a signature file by using signature inner-product between *unit signatures* and the new signature to be allocated. The signature inner-product of two signatures is a scalar value as defined in **Definition 3.4**, where each signature is considered as a bit string. To minimize the variance of the count vector, we cluster different signatures in 1's position into the same processing node. **Theorem 3.2** allows us to compare the degree of difference between signatures by using signature inner products. Thus, in this method, we compare the degree of differences between a new signature and representative signature of each node, that is, the unit signature of each node. The *unit signature* represents the status of signature allocation in each processing node as defined in **Definition 3.3**. The Inner-product method allocates a new signature to the processing node with minimum inner product among the inner products of the new signature and the *unit signatures* of each node.

The Inner-product method uses three kinds of data for each processing node. These are *count-vector, signature-count,* and *unit-signature.* The *signature-count* is the number of signatures allocated to each node. The detailed procedure of our proposed Inner-product method consists of four phases.

- Initialization Phase

  The initial states are as follows given that the number of processing nodes is p. Here, sc_i, cv_i, and uv_i denote the *signature-count*, *count-vector*, *unit-signature* of node i, respectively.

  $sc_i = 0, (1 \le i \le p)$

  $cv_i = 0, (1 \le i \le p)$

  $uv_i = 0, (1 \le i \le p)$

- Pre-calculation Phase

  In this phase, inner-products of all processing nodes are computed before allocating the signature s_j as follows.

  $uv_i{+}{+} \quad = uv_i \bullet s_j$, where $1 \le i \le p$ and s_j is a signature to be allocated

- Node Selection Phase

  After all the *inner-products* of p nodes are computed, we select one processing node with the

minimum uv_i++. That is, the node p_k where a new signature is allocated is defined by p_k = **min**$\{uv_i$++$\}$, $(1 \le i \le p)$. If more than two nodes have the same uv_i++, then we select one with the lower *signature-count*. In addition, if the *signature-counts* of more than two nodes are same, then we randomly select one processing node but it rarely happens.

- Node-allocation Phase

  Let p_k be node k selected during the node selection phase. Then, we allocate the signature s_j into the node p_k. After allocating s_j, the *signature-count* of p_k is increased by 1 and the *count-vector* of p_k is changed and unit signature uv_k is recalculated as follows.

  $sc_k = sc_k + 1$

  $cv_k = cv_k + s_j$

  if $cv_{ki} > mean(cv_k)$ $\quad uv_{ki} = 1, \quad$ *otherwise* $uv_{ki} = 0$.

In Figure 6, we can see that new signature is allocated to 1-*th* node by selecting minimum *inner-product* among the *n* nodes.

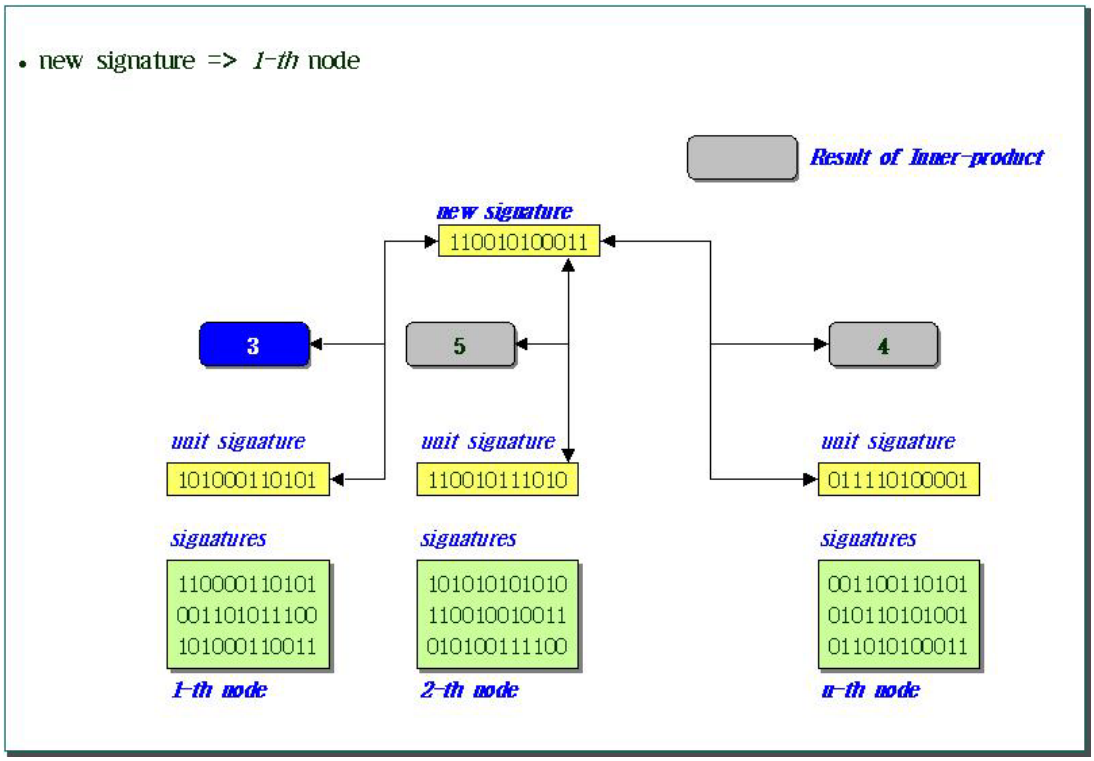

Fig. 6. The processing map of the Inner-product method

The detailed algorithm of the Inner-product method is as follows:

Algorithm: Inner-product

Input:

- *s_j*[*n*] : a new signature to allocate, one dimensional bit string, where *n* is the size of signature

Output:

- *p* : processing node number where a new signature is to be allocated

Variable:

- *PN* : the total number of processing nodes.
- *cv*[*PN*][*n*] : count vector, two dimensional array of integer.
- *uv*[*PN*][*n*] : unit signature, two dimensional array of binary positions.
- *uv_plus*[*PN*] : temporal unit signature, two dimensional array of binary positions.
- *sc*[*PN*] : signature count, one dimensional array of integer.

Process:

/* initialize */

/* *static* means the same thing in the C language syntax */

static sc[*i*] = 0, ($0 \le i < PN$)

static cv[*i*][*j*] = 0, ($0 \le i < PN$, $0 \le j < n$)

static uv[*i*][*j*] = 0,($0 \le i < PN$, $0 \le j < n$)

for(i = 0; i < *PN*; i++) {

/* construct uv++ */

uv_plus[*i*] = *uv*[*i*] • *s_j*;

}

/* MIN function returns index of unit signature plus with minimum value. If two or more unit signature pluses are the same, then it returns the index node with minimum signature count. If the signature count are also same, then it returns a randomly selected index */

p = **MIN**(*uv_plus*[*i*]);

for(i = 0; i < *n*; i++) { /* allocate the signature *s_j* to *p-th* processing node. */

cv[*p*][*i*] = *cv*[*p*][*i*] + *s_j*[*i*];

}

sc[*p*] = *sc*[*p*] + *1;* /* increase the signature count of *p-th* processing node */

```
/* reconstruct unit signature of p-th processing node; the mean function returns the mean of
count vector*/
for( i = 0; i < n; i++ ) {
        if ( cv[p][i] > mean(cv_p) )
                uv[p][i] = 1;
        else
                uv[p][i] = 0;
}
return(p);              /* return output */
```

Example) Let cv_i and cv_j be the count vectors of node i and node j, respectively such that cv_i = <4, 5, 4, 7, 4, 6, 4, 6, 3, 5, 5, 7> and cv_j = <5, 4, 7, 4, 6, 4, 6, 4, 5, 3, 7, 5>, where n = 12. Then, unit signature uv_i and uv_j are <0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1> and <0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0>, respectively. Suppose we have a new signature, s_k to be allocated such as s_k = <1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0>. Then $s_k \bullet uv_i$ is 0 and $s_k \bullet uv_j$ is 4. Thus, the Inner-product method allocates s_k to node i because node i has lower *inner product.*

IV. PERFORMANCE EVALUATION

1. Experiments

This section evaluates the performance of the Inner-product method with respect to retrieval time and insertion time. The experiments were performed on a Sparc-20 workstation with 128Mbytes of main memory. In the experiments, 10,000 and 100,000 documents are used. Each document consists of 20 fields such as author, title and eighteen keywords. The databases were constructed based on the result from analysis of 10,000 technical reports and journals in the Chungbuk National University. Figure 7 illustrates the type of sample document in the database. The input and design parameters are shown in Table 2. In addition, the experiments were based on the same synthetic data that was used in [20]. Three types of synthetic data are generated such as uniform, normal, and exponential distribution data sets. Three basic distributions were used over the range of $[-2^{31}, 2^{31}-1]$: 1) a uniform distribution, 2) a normal distribution

N(0,σ), where $\sigma = \frac{1}{3}2^{31}$ and 3) an exponential distribution $\frac{1}{\theta}e^{\frac{(x+2^{31})}{\theta}}$, where $\theta = \frac{1}{4}2^{32}$. The data skewness increases as it goes from uniform to exponential.

Attribute	**# for Document**
Accession number	1
Author	1~6
Title	1~10
Subject	1~10
Category	1~7
Date	1
Publisher	1
Vol-Num	1
Page	1
Total Attribute Values per Document	9~38
Average Attribute Values per Record	20

Fig. 7. The type of sample document in database

Table 2. Parameters

Parameters	**Description**	**Value**
N	the number of documents	10K, 100K
P	the number of processing nodes	8, 16
D	the number of words per document	20
n	the signature size in bits	512
m	the number of bits set to '1' per word	16

In these first experiments, the response times of signature retrieval of the four methods (Inner-product, Min-entropy, LCDM, and random method) are compared. The response time of signature retrieval is the signature search time plus the false drop time in the processing node. Since the response time of intra-query parallelism depends on the processing node with the maximum number of retrieved signatures, we

use the maximum number of the retrieved signatures as the measure of the response time of each query. One retrieved signature causes at least one disk access because its false drop time is based on the real document match processing. The maximum number of the retrieved signatures for i-query is **max**$\{R_{i1}, R_{i2}, \ldots, R_{ip}\}$, where $R_{ij}(1 \le j \le p)$ is the number of retrieved signatures of the j-th processing node for i-query.

Our experiments are classified into two environments such as real documents and synthetic documents. Because of the limited experimental environment, we use real 10,000 documents from the library of Chungbuk National University. Therefore we first decluster a data file with real 10,000 documents into 8 processing nodes. And then we decluster a data file with synthetic 10,000 documents into 8 processing nodes and a data file with synthetic 100,000 documents into 16 processing nodes. Many information retrieval systems in practice may have millions of documents, which is much larger than our test data sets. Though we are using relatively small sets of test data due to limited environment for experiments, the behavioral characteristics for the proposed method may not be different for various sizes of test data. 100 sample queries are randomly selected for test. We use single-term queries to make the response sizes large enough to show the effect of parallel processing more clearly. In all figures, the declustering effects of the four methods are compared. In addition, one more graph is presented for easy understanding of our method's characteristics. The additional graph shows the optimal case, i.e., the theoretical minimum of the maximum number of the retrieved signatures for each query, which is the lower bound of declustering. For example, given a query, q, the theoretical minimum of the maximum number of the retrieved signature is the $\lceil n/m \rceil$, where n is the total number of retrieved signatures and m is the number of processing nodes.

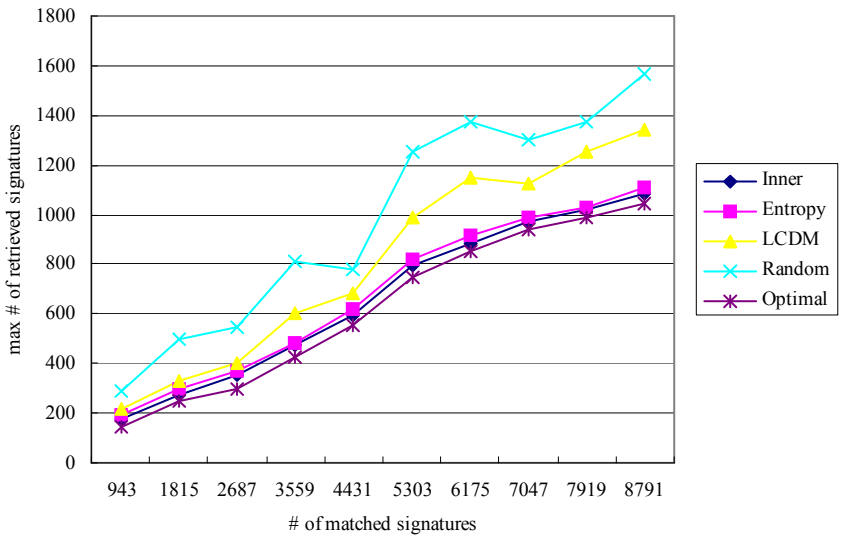

Fig. 8. Real data set with 10,000 documents in 8 PNs

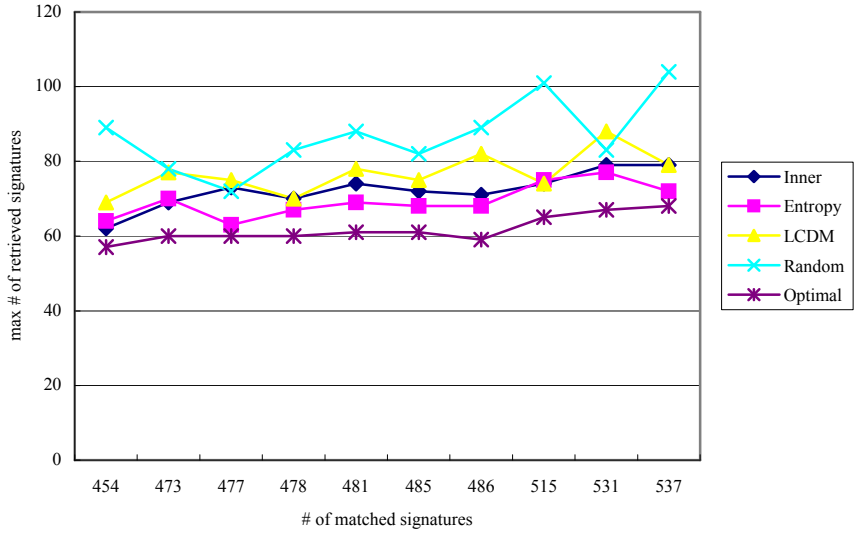

Fig. 9. Uniform distribution data set with 10,000 documents in 8 PNs

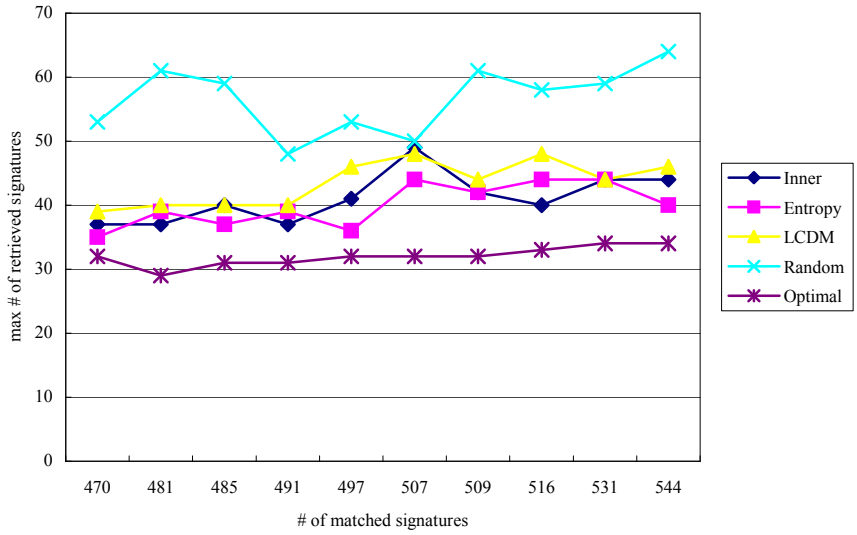

Fig. 10. Uniform distribution data set with 100,000 documents in 16 PNs

Figure 8 shows the retrieval performance of our proposed method for real data set. The retrieval performance of our proposed method is similar to that of the MIN-entropy for static environments and outperforms those of the LCDM and the random method. Figure 9 and Figure 10 shows the retrieval performance of our proposed signature file declustering method for uniform distribution data set. The

Inner-product, Min-entropy and LCDM show better performance than the random method. The Inner-product is better than the LCDM and almost equal to the MIN-entropy in most cases. Figure 10 shows similar results to Figure 9. Thus, we can say that our proposed method is scaled up without losing generality.

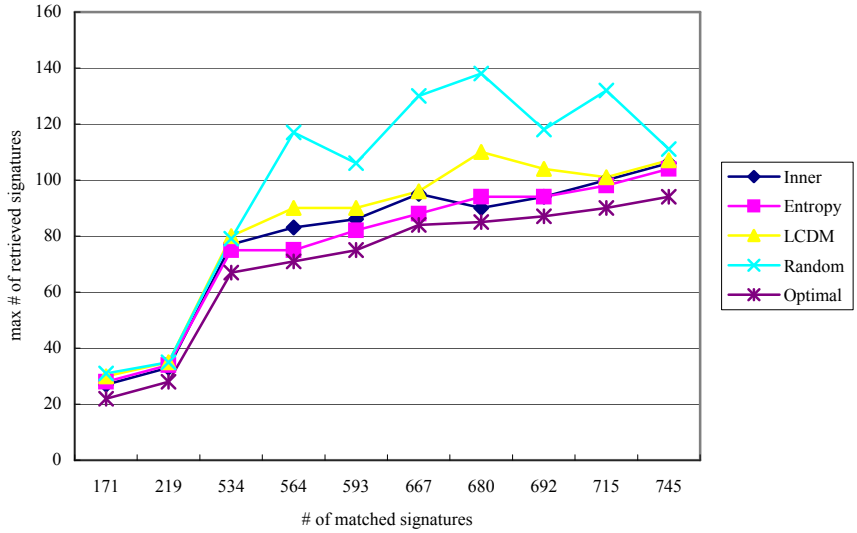

Fig. 11. Normal distribution data set with 10,000 documents in 8 PNs

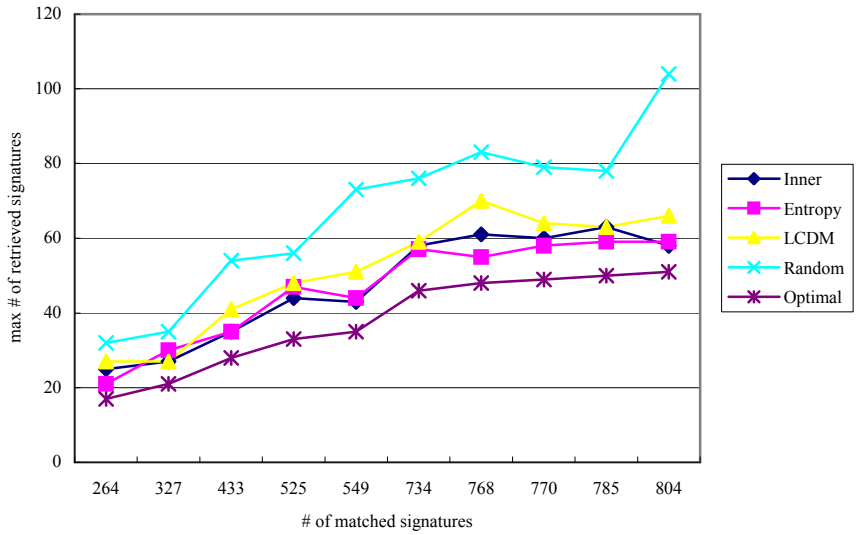

Fig. 12. Normal distribution data set with 100,000 documents in 16 PNs

Figure 11 and Figure 12 shows the retrieval performance of our proposed signature file declustering

method on normal distribution set. We can see through the two figures that the same performance behaviors occur on normal distribution data set.

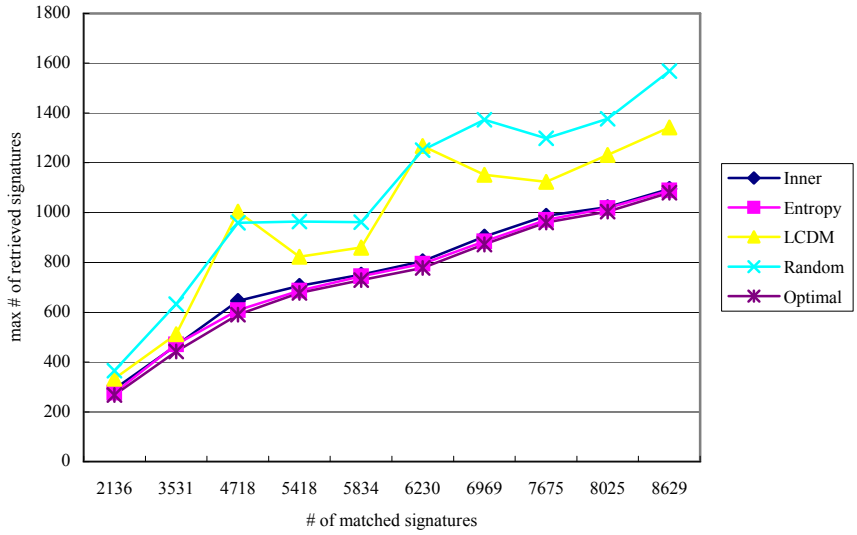

Fig. 13. Exponential distribution data set with 10,000 documents in 8 PNs

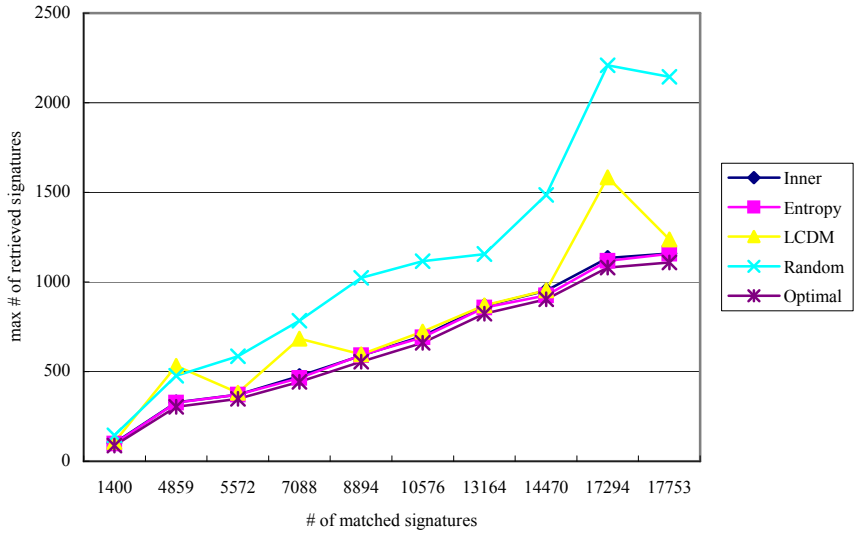

Fig. 14. Exponential distribution data set with 100,000 documents in 16 PNs

Finally, the performances on data files with exponential distribution data set are shown in Figure 13 and Figure 14. In this two graphs also, we can see the same behavior. However, for large data set, the performance of the random and the LCDM methods are much less than Inner -product. The reason is that as the matched signatures grow rapidly, the Inner-product gets stable by using accumulated information

dynamically. Although the LCDM is better than the random method in the most of the queries, it is worse than or similar to random method in some queries. This is because the large amount of matched signatures has the same suffix and those signatures were allocated to the same processing node. But, the Inner-product and the MIN-entropy method do not have such problem because it dynamically declusters signature file based on statistic information of the previously allocated signatures. As a result, we conclude that the Inner-product and MIN-entropy give better retrieve performance than the LCDM and the random method over the various probability distributions of workloads. In addition, the Inner-product and MIN-entropy method are highly scalable and has deterministic behavior, which is not the LCDM case.

In the second experiment, we compared the insertion time of signature declustering methods by using *Big"Oh"*(O()), a kind of asymptotic notation. The insertion time consists of two terms as follows:

$$I_{time} = D_{time} + O_{time}$$

where, I_{time} : *Insertion Time*

D_{time} : *Declustering Time*

O_{time} : *Local Organization Time*

However, we only concern about D_{time} because O_{time} is the same for all signature file declustering methods. Also, we compare only one signature insertion time. The computation time of the four declustering methods is described by using *O()* notations with variable *n* and *p* as follows :

- Random : independent on *p* and *n, thus O(1)*
- LCDM = *matrix multiple computation time* = $O(p^2)$
- MIN-entropy = *cv++ construction time + signature entropy construction time + minimum signautre entropy finding time + cv increment time*
  = $O(pn) + O(pn) + O(p) + O(n) = O(pn)$
- Inner-product = *uv++ construction time + minimum uv++ calculation time + cv increment time*
  = $O(pn) + O(p) + O(n) = O(pn)$

  where, *n* = the size of signature, *p* = the number of processing nodes

Therefore, when $p << n$, the LCDM outperforms the Inner-product and the MIN-entropy. However, when $p >> n$, the Inner-product outperforms the LCDM and the MIN-entropy. This is because the LCDM is not gracefully scalable to the number of processing nodes and the MIN-entropy takes construction time of signature entropy additionally in all processing nodes.

As a result, the MIN-entropy is suitable to static environment where retrieval is a main operation and insertion seldom occurs. Meanwhile, the Inner-product method is suitable to dynamic environment of large scale parallel database systems where insertions happen frequently.

2. Discussions

Until now, we have investigated and analyzed the performance and characteristics of signature file declustering methods. Based on these researches, we address comparisons to choose the most appropriate signature for a variety of environments. In Table 3, signature file declustering methods in four points such as scalability, simplicity, insertion time and retrieval time are compared.

Table 3. Comparisons of signature file declustering methods

	Inner-product	**MIN-entropy**	**LCDM**	**Random**
Scalability	Good	Good	Bad	Excellent
Simplicity	Good	Bad	Bad	Excellent
Insertion	Good	Bad	Good	Excellent
Retrieval	Excellent	Excellent	Good	Bad

We recommend the MIN-entropy for the environment where retrieval is main operation and insertion seldom occurs and scalability is important such as mesh, ring parallel systems. The Inner-product are highly recommended for the environment where insertion frequently happens and scalability is important like the MIN-entropy. The LCDM can be used for hypercube parallel systems because hypercube systems have a power of two numbers of processing nodes. The random method can be used when the retrieval time is not important.

V. CONCLUSION

We have described various problems of the LCDM method that is used as a declustering method for the Hamming Filter. The LCDM cannot avoid data skew if many signatures have the same suffix. In addition, it has the following problems that make the *LCDM* difficult for parallelism. First, it is not gracefully scalable to the number of processing nodes. The method is properly defined only when the number of

processing nodes is a power of two. This is a serious restriction because most current parallel systems in practice may not have a power of two number of processing nodes. Second, it is not deterministic algorithm. For the suffix with the *m*-size, *m!* check matrixes can be issued for the *LCDM* to decluster a signature file. While the choice of the check matrix can affect the declustering performance of the *LCDM*, no appropriate guideline has been provided to choose an optimal check matrix. Finally, the *LCDM* may lead to information loss because it only uses partial information, i.e., the suffix of the signature.

In this paper we have proposed a new signature file declustering method, called Inner-product that overcomes the problems in the LCDM. It declusters signature file dynamically based on the current status of signature allocation. Thus, the Inner-product can cope with a variety of workloads and configurations. We have showed through the performance evaluation based on the statistical modeling that the Inner-product and MIN-entropy give better retrieval performance than the LCDM for data sets with various distributions such as uniform distribution, normal distribution, and exponential distribution. We also have addressed the performance of signature insertion time by using asymptotic notation. When $p >> n$, the Inner-product outperforms the LCDM and the MIN-entropy. It is show that the Inner-product method works well in a dynamic environment where insertions occur frequently.

As a result, the proposed Inner-product method is highly recommended for the environment where insertion frequently happens and scalability is important like the MIN-entropy. And the MIN-entropy is recommended for the environment where retrieval is main operation and insertion seldom occurs and scalability is important such as mesh, ring parallel systems.

In the near future, we are investigating a formal framework that may give better understanding of the behavioral characteristics of the Inner-product method over various probability distributions of workloads.

Acknowledgments

This work was supported by grant No.(R01-1999-00244) from the Korea Science & Engineering Foundation.

REFERENCES

[1] C. Faloutsos. Signature-based text retrieval methods. *A Survey. IEEE Computer Society TechnicalCommittee on Data Engineering*, 13(1), pages 25-32, 1990.

[2] C. Faloutsos and S. Christodoulakis. Design of a signature file method that accounts for non-uniform

occurrence and query frequencies. In *Proc. of the 11th VLDB Conf.*, pages 165-170, Stockholm, Sweden, August 1985.

[3] C. S. Roberts. Partial match retrieval via the method of the superimposed codes. In *Proc. of IEEE 67*, pages 1624-1642, Dec. 1979.

[4] Z. Lin and C. Faloutsos. Frame-sliced signature files. *IEEE Transaction on Knowledege and Data Engineering*, 4(3), pages 281-289, June 1992.

[5] U. Deppisch. S-tree: A dynamic balanced signature index for office retrieval. In *ACM SIGIR*, pages 77-87, 1986.

[6] F. Rabitti and P. Zezula. A dynamic signature technique for multimedia database. In *Proc. of the 13th ACM SIGIR*, pages 193-210, Brussels, Belgium, September 1990.

[7] P. Zezula, F. Rabitti, P. Tiberio. Dynamic Partitioning of Signature Files. *ACM transaction on information system*, 9(4), pages 336-369, 1991.

[8] P. Ciaccia, P. Zezula, and P. Tiberio. Hamming filter: A dynamic signature file organization for parallel stores. In *Proc. of the 19th VLDB Conf.*, pages 314-327, Dublin, Ireland, 1993.

[9] B. M. Im, M. H. Kim and J. S. Yoo. Dynamic Construction based on frame sliced approach, *Data & Knowledge Engineering 30*, pages 101-120, 1999.

[10] P. Tiberio, F. Grandi, and P. Zezula. Frame-sliced partitioned parallel signature files. In *Proc. of 15th Ann. Int'l SIGIR*, pages 286-297, Denmark, June 1992.

[11] D.L. Lee and C. Leng. A partitioned signature file structure for multiattribute and text retrieval. In *Proc. of the 6th Int'l Conf. On Data Engineering*, pages 389-397, Los Angeles, California, Feb. 1990.

[12] P. Ciaccia, P. Zezula, and P. Tiberio. Declustering of key-based partitioned signature files. *ACM TODS*, 21(3), 1996.

[13] C. Faloutsos and D.Metaxas. Declustering using error correcting codes. In *Proc. of 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 253-258, Philadelphia, Pennsylvania, March 1989.

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

[15] M. Mehta and D. J. DeWitt. Managing intra-operator parallelism in parallel database systems. In *Proc. of the 21th VLDB Conf.*, pages 382-394, Zurich, Swizerland, 1995.

[16] M. H. Kim and S. Pramanik. Optimizing database accesses for parallel processing of multikey range searches. *The Computer Juornal*, 35(1):45-51, 1992.

[17] Y. Y. Sung. Performance analysis of disk modulo allocation method for cartesian product files. *IEEE Transactions on Software Engineering SE-13(9)*, pages 1018-1026, 1987.

[18] K.A. Abdel-Ghaffar and A. El. Abbadi. Optimal disk allocation for partial match queries. *ACM TODS*,

18(1):132-156, March 1993.

[19] K.A. Hua and C.Lee. Handling data skew in multiprocessor database computers using partition tuning. In *Proc. of 17th VLDB*, pages 523-535, Barcelona, Spain, September 1991.

[20] B. M. Im, M. H. Kim and J. S. Yoo. Declustering signature files based on a dynamic measure. *Information Processing Letters*, 74, pages 235-241, 2000.