EXPLOITING SYMMETRIES FOR LOW-COST COMPARISON OF FILE COPIES

Daniel Barbara
Hector Garcia-Molina

CS-TR-117-87

November 1987

EXPLOITING SYMMETRIES FOR LOW-COST COMPARISON OF FILE COPIES

Daniel Barbara
Hector Garcia-Molina

Department of Computer Science
Princeton University
Princeton, NJ 08544

Bernardo Feijoo

Departamento de Matematicas y Ciencias de la Computacion
Universidad Simon Bolivar
Caracas, Venezuela

ABSTRACT

In this paper we examine a new technique for comparison of remotely located file copies. With this new technique up to two differing pages can be located and any number of multiple differing pages can be detected. The technique uses a communication overhead of $O(\log^2(N))$, where N is the number of pages in the file. It is based on a set of symmetries of an hypercube with dimension $\log(N)$.

November 10, 1987

EXPLOITING SYMMETRIES FOR LOW-COST COMPARISON OF FILE COPIES

Daniel Barbara
Hector Garcia-Molina

Department of Computer Science
Princeton University
Princeton, NJ 08544

Bernardo Feijoo

Departamento de Matematicas y Ciencias de la Computacion
Universidad Simon Bolivar
Caracas, Venezuela

1. INTRODUCTION

Files are replicated in a distributed system in order to improve reliability and performance. However, due to human errors or hardware failures copies may diverge. It then becomes necessary to compare the remotely located files and identify the differences. There are a number of strategies for this task, ranging from transmitting the entire file and performing a local comparison to some relatively sophisticated methods [Me83,Fu86]. The more sophisticated approaches are required in cases where the files are large and the cost of identifying the differences must be kept low.

Such a case arrises, for example, in a triple modular redundant (TMR) database system that has been built at Princeton [Pi86]. The system is implemented on three SUN 120 computers, each with a full copy of a database. Transactions are submitted to any of the three nodes, but before it is executed it is reliably broadcast to the other two nodes. Once it is certain that all three nodes have the transaction, each node executes the transaction independently on its local database. The three results are sent to the user, who then uses voting to select the correct one. The system can tolerate one arbitrary failure of a machine (e.g., a head crash, or the processor writes the wrong balance into an account) and still guarantees correct data and transaction results.

The system continues to operate with two computers until a failed computer is fixed and restarted. The restarted machine must then identify the portion of the database that is corrupted (if any) and then request a copy of that portion from the operational machines. This is where the file compare algorithm is used.

The time to correct the database must be kept as small as possible. One reason is that during this time the system is running with only two computers and thus another failure may be catastrophic. A second reason is that during recovery new transactions keep arriving. The operational nodes continue to process them (they cannot allow a single failed node to halt the system), but the recovering node must execute them *after* it obtains a correct database copy. The longer recovery takes, the longer the catch up period will be [Pi87]. Beyond a certain limit, the recovering node may never catch up. Thus, it is essential to identify the differing database pages as quickly as possible.

In this paper we present a file comparison mechanism that can identify up to two differing file pages with a single message transmission. The basic idea is that each site maintains a collection of *signatures* for the database. To compare two remotely located files, one site sends its signatures to the other. The second site compares the two sets of signatures. If there are two or less pages that differ, then the second site can immediately determine what these pages are. If there are three or more page differences, the algorithm detects this are yields a set of pages that is a superset of the pages that differ. Since the algorithm only compares probabilistic signatures, there is always a small probability that the result will be incorrect (e.g., there may be undetected differences). By making the signatures lager, the probability of error can be made arbitrarily small.

We start by discussing the two other published approaches for remote file comparison. They will be explained by means of a simple example, and then contrasted to our proposal. In doing so, we will argue that it is important in some cases (like the TMR database system) to identify at least *two* page differences with only one message. The other two approaches either send many more messages or can only detect one difference.

2. OVERVIEW OF THREE FILE COMPARE STRATEGIES

All three strategies assume that the file is divided into a collection of pages P_1, P_2, ..., P_n. For simplicity we assume that n is a power of 2, i.e., $n = 2^m$. (This can be generalized.) For each page P_i we can compute a signature $sig(P_i)$. One can think of the signature as a check sum, although the are more sophisticated ways to compute them [Me83]. If the signature contains b bits, then the probability that two different pages have the same signature is 2^{-b}.

The signatures for j pages can be combined into one by performing an exclusive or. If the original signatures have b bits, the combined signature will also have b bits. The combined signature can be used to compare the j pages to their copies in a single operation. If the combined signatures are identical, then the copy pages are probably no different. (The probability that one or more of the j pages are different but have the same combined signature is approximately 2^{-b} [Fu86].) If the combined signatures are different, then there are probably some differences among the pages. However, the combined signature does not by itself identify the pages that are different.

To explain the three file comparison mechanisms we will use an eight page file example. We will call the pages *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*. We will refer to the signatures of these pages by the same name. When we form combined signatures, we will concatenate the letters of the pages involved. For instance, *abc* is the exclusive or of the *a*, *b*, *c* signatures.

The mechanism proposed by Metzner [Me83] is based on a signature binary tree of depth $m+1$. The tree for our $m = 3$ example is shown in Figure 2.1. Suppose we wish to compare the two copies of the file on machines X and Y, and say they differ in page c.

Computer X sends the top level combined signature *abcdefgh* to Y. Since $sig(c)$ differs, the combined signature will also be different and Y knows that the files are different. Next, Y sends the second level signatures to X. Since the *efgh* signatures are identical but the *abcd* ones differ, then X knows that the problem is in the first half only. Hence, it returns the *ab* and *cd* signatures to Y, which uses them to diagnose that the difference is somewhere in the c or d pages. Finally, Y sends the c and d signatures to X, which pinpoints the difference as page c.

If more than one page differs, then the appropriate tree paths are sent. Thus, any number of differences can be identified. The price that must be paid is that (1) $m + 1$ message exchanges are needed to identify even one difference, and (2) many signatures

```
      abcdefgh
     /        |
  abcd        efgh
  /   |       /    |
 ab    cd    ef     gh
 / |  / |    / |    / |
a  b  c d  e  f  g  h
```

Figure 2.1
Signature tree for Metzner's strategy

must be stored by each node. The 2^m signatures at the leaves will be kept in any case (probably stored in the pages themselves), but this strategy requires 2^m-1 additional signatures. For a 2^{20} page file (about one million pages), this would be roughly one million signatures, or eight megabytes if each signature is 64 bits. If it is not possible to keep all the signatures in main memory, then the cost of managing the non-leaf signatures will be prohibitive.

With the approach suggested by Fuchs et al [Fu86], a much smaller number of signatures must be stored. The signatures are organized as a 2 by m array. The $m = 3$ case is shown in Figure 2.2. (As will be discussed shortly, only $m + 1$ signatures are actually needed.)

abcd	efgh
abef	cdgh
aceg	bdfh

Figure 2.2
Signature array for Fuchs' strategy

Note that the first row corresponds to the second level of Metzner's tree. The second row corresponds to the third level, except that the odd numbered signatures of the level (*ab* and *ef*) have been combined into one. The even ones are similarly combined. The same is true for the last row and the last level of the tree.

Say nodes *X* and *Y* wish to compare the file when page *c* differs. Node *X* sends the entire array of signatures to *Y*. Node *Y* notices that the *abcd* signatures differ, so there must be a difference in the first half and not in the second. At the second row, the *abef* signatures are identical, so the *a* and *b* pages must also be identical, leaving only the *c* and *d* pages as candidates for a problem. At the third level, the *bdfh* signatures match, so the *d* pages must be equal. This identifies page *c* as the culprit.

This strategy can only precisely identify a single difference. If more than one page differ, then a superset is identified. For example, suppose that pages *b* and *c* differ. The first row can tell us that there are no differences in the second half. Unfortunately, all signatures in rows 2 and 3 differ, so the best that can be done is to identify all four pages *a*, *b*, *c*, and *d* as differing.

In summary, Fuchs' strategy correctly diagnoses a single difference and detects (but does not pinpoint) two or more differences. The number of signatures needed in the array is $m + 1$. To see that only $m + 1$ are needed and not $2m$, note that with the first column and the signature *abcdefgh* one can compute the second column. For a one million page file (2^{20}), the number of signatures is 21, much smaller than the one million signatures required by the previous approach.

The strategy we propose here is similar to Fuchs' except that *two* differences are precisely identified and the array is of size $m(m+1)/2 + 1$. Figure 2.3 presents the array for a 2^3 page file. (We show $m(m+1)$ signatures in the figure. Again, the second column can be computed from the first and the signature *abcdefgh*.)

abcd	efgh
abef	cdgh
aceg	bdfh
abgh	cdef
acfh	bdeg
adeh	bcfg

Figure 2.3
Signature array for our strategy

To illustrate, suppose that pages *c* and *e* differ at nodes *X* and *Y*. The third row tells us that the differences must be in pages *a*, *c*, *e*, or *g*. The fourth row identifies the differing pages as one of *c*, *d*, *e*, or *f*, so it must be the intersection of these two sets that is causing the problem, i.e., pages *c* and *e*. The first row tells us that both halves of the file have problems, so it must be the case that both *c* and *e* (and not just one of them) that have differences. Exactly why this all works will be explained later on in the paper.

Our mechanism precisely identifies one or two differences. The mechanism also detects the situation where the are more than two differences and it identifies a superset of the pages that differ. Identifying two differences is clearly better than identifying a single one, but there are two important questions to address: (1) Is the extra cost tolerable? and (2) Is it worth it?

We believe that the extra cost is reasonable in may cases. For example, in a one million (2^{20}) page file, our mechanism must store and manage 221 signatures as opposed to 21 with Fuchs' (and a million with Metzner's). Given current memory prices, we feel the cost of the 190 extra signatures is tolerable. With respect to network costs, in many networks sending 21*4 bytes is just about as expensive as sending 211*4 bytes (e.g., both fit in a single packet). With respect to management of the signatures, note that with either strategy when a page is updated, all signatures in the arrays must be updated. Each signature can be updated with a single exclusive or operation (probably one machine instruction), so the overhead is roughly on the order of 211 instructions to 21 instructions depending on the strategy. (When P_i is changed to P'_i, we compute $sig(P_i)$ exclusive-or $sig(P'_i)$; then we exclusive-or this result with all signatures in the array.) Of course, if the file is smaller than a million pages, the cost of our approach will be even closer to that of Fuchs'.

The second issue is whether it is important to identify two differences. From a theoretical point of view, at least, it is important to know whether it is possible to identify two differences (with a single relatively short message) and what the cost is. Note incidentally, that even though our approach is similar in structure to Fuchs', it is not a straightforward generalization. That is, going from single to double identification is not just a matter of changing a parameter in Fuchs' strategy. It is substantially more complicated than that. Our approach in turn cannot be easily generalized to identify three or more failures, so it is still an open question what the general solution for identifying n differences is, if any exists.

From a more practical point of view, there may be situations where it is important to identify double differences quickly. For example, a disk failure may affect two contiguous pages and not just a single page. Fuchs' strategy precisely identifies only *half* of the contiguous differences, while our strategy identifies them all. (Referring to our example, Fuchs' strategy could identify differences in pages a and b. However, if pages b and c are the ones that differ, it would identify the set a, b, c, and d containing two extra pages. If pages d and e differ, then all eight pages would be identified as being potentially different. Our mechanism, on the other hand, would correctly identify the pages in all these examples.)

In a TMR database system like the one described earlier, transactions that run on a failed processor can cause arbitrary damage to the database. While these failures are not common, when they do occur it is possible that more than one page being updated by a transaction is corrupted. As discussed earlier, the time to get a good database copy must be kept to a minimum. Thus, in this scenario, we believe it makes sense to pay the slightly higher overhead of managing $\frac{m(m+1)}{2} + 1$ signatures during normal operation in order to recover fast in a larger percentage of the failures.

3. OUR TECHNIQUE

In this section we present our technique. As we mention in the previous section, the strategy is similar to the one presented in Fuchs et al [Fu86], except that two differences are identified. We begin this section by giving a more formal description of the work in [Fu86].

In their strategy, a checking matrix is generated as a matrix C_k^n of check symbols c_{ij}. Each one of the symbols is constructed as the exclusive or of the signature functions of a subset of the pages. A simple way of visualizing the construction of the c_{ij} symbols is by constructing a $k-ary$ tree in which the leaves are the file pages. Figure 3.1 shows the binary tree for the matrix C_2^8. In each level of the tree, two signatures are generated. The first identified by black circles and the other by white circles. The pages for each signature in each level of the tree are the leaves in the subtrees whose roots are of the same color. (The root of the tree is **not** a signature.) Although the strategy in Fuchs et al can be used for an arbitrary k, we focus here in the case $k = 2$, because our techique is based in it. We should point out that if the number of pages in the file is not a power of 2, we can do one of two things: (1) use the next largest power of 2 and assume all the missing signatures to be zero, or (2) split the file into smaller pieces such all of them are powers of two. Both ideas can be used with all the file comparision strategies described in the previous section.

Another way of visualizing the checking matrix is to show a matrix in which the

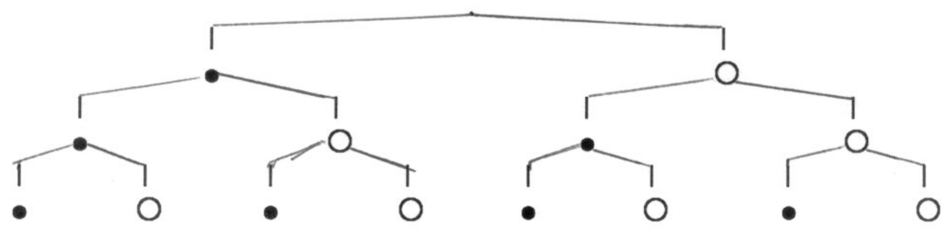

Figure 3.1

entries tell the signature to which the page belongs in that level. Figure 3.2 shows such a matrix for the case C_2^{16}. To illustrate, suppose that the file copies differ in page 1. In the first three rows, the signature affected would be the black one (β*u*), while in the last row it would be the white one (ξ*i*). The intersection of the page sets affected in each row gives us page 1 as the culprit.

Pages

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
●	●	●	●	●	●	●	●	○	○	○	○	○	○	○	○
●	●	●	●	○	○	○	○	●	●	●	●	○	○	○	○
●	●	○	○	●	●	○	○	●	●	○	○	●	●	○	○
●	○	●	○	●	○	●	○	●	○	●	○	●	○	●	○

Figure 3.2.

For $k = 2$, the construction of signatures can also be viewed as corresponding to a set of symmetries on a hypercube of dimension *m*. Figure 3.3 shows the 3-dimensional cube for C_2^8 in which the vertices correspond to pages in the file. One pair of signatures of the same level can be obtained by cutting the cube in two halves by a plane parallel to one of its sides. For instance, Figure 3.3 shows the symmetry producing the checking symbols c_{10} and c_{11} as formed by $\{P_0, P_1, P_2, P_3\}$ and $\{P_4, P_5, P_6, P_7\}$ respectively.

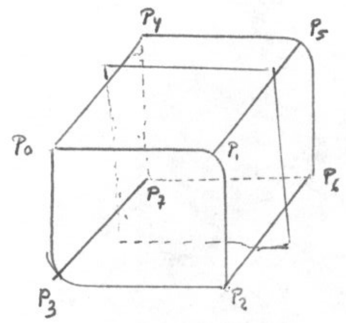

Figure 3.3

The comparison between two files generates a new matrix called the *syndrome matrix*.

Definition 2.1 [Fu86] *Syndrome matrix.* A syndrome matrix $A_k^n = [\alpha_{ij}]$, $0 \le i \le \log_k n - 1$, $0 \le j \le k - 1$, is a matrix of α_{ij}, such that

$$\alpha_{ij} = \begin{cases} 0 & \text{if } c^1_{ij} = c^2_{ij} \\ 1 & \text{if } c^1_{ij} \neq c^2_{ij} \end{cases}$$

where $[c^1_{ij}]$ and $[c^2_{ij}]$ are the checking matrices for the file in sites 1 and 2 respectively. •

By the use of a decoding algorithm and the syndrome matrix, a single differing page can be located. Here, we describe a simplified algorithm for the case $k = 2$, because our technique is based on this case. Since $k = 2$, there will be only two signatures per level. Also the syndrome matrix will have only two entries per row, i.e., α_{t0} and α_{t1}. Let $s^b_t(m)$ and $s^w_t(m)$ represent the pages included in the $t-th$ level. We call them the *black* and the *white* signature respectively. For instance for C^{16}_2 the signatures in level 1 are $s^b_1(4) = \{0,1,2,3,8,9,10,11\}$ and $s^w_1(4) = \{4,5,6,7,12,13,14,15\}$ respectively. The entries of the syndrome matrix will be called α^b_t and α^w_t (as opposed to α_{t0}, α_{t1}) for each level. The algorithm begins with the whole set of pages as the candidate solution. Then it uses each row of the syndrome matrix to try to filter out those pages for which there is no difference in the two copies. If both entries in the row are the same, no information can be gathered from this row. But if one of the two entries is a 1 and the other a 0, half of the pages can be filtered out of the solution. For instance, $\alpha^b_t = 1$ and $\alpha^w_t = 0$, then we can intersect the solution so far with the set $s_{t_b}(m)$, which contains only half of the pages. The complete algorithm is given in Figure 3.4. (N_r denotes the number of rows in the checking matrix.)

Algorithm **LOCATE**

```
Solution := $\{0, 1, ...., n-1\}$;
for $t := 0$ to $N_r$ do
  if $\alpha^b_t \neq \alpha^w_t$ then
    if $\alpha^b_t = 1$ then
      Solution := Solution $\cap\, s^b_t(m)$
    else
      Solution := Solution $\cap\, s^w_t(m)$;
```

Figure 3.4

Exploiting the idea of symmetries, it is possible to improve the technique. For instance, in the cube of Figure 3.3, one can use three new ''diagonal'' symmetries as shown in Figure 3.5 to generate new pairs of signatures. Next we prove that exploiting such symmetries one can locate two different pages in the file at a reasonable cost.

In our technique, the matrix contains $K = \dfrac{m(m+1)}{2}$ black (and white) signatures, each one corresponding to one of the symmetries mentioned before. The construction of signatures is a recursive process. Before showing it in detail, we need two definitions.

Definition 3.1 *The shift function*

$$sh_n(i) = \begin{cases} i + \frac{n}{2} & \text{for } 0 \leq i \leq \frac{n}{2} - 1 \\ i - \frac{n}{2} & \text{for } \frac{n}{2} \leq i \leq n - 1 \end{cases}$$

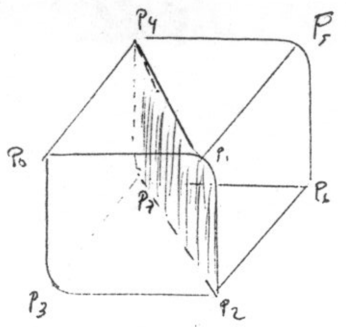

Figure 3.5

This function shifts page number i forward by $\frac{n}{2}$ if i is in the first half of the set of pages, or backward by $\frac{n}{2}$ if i is in the second half of the set. The function can also be applied to a set of pages P, giving a set of pages P' in which each element of P' is the shift of an element of P.

Definition 3.2 *The symmetry function*

$$sym_n(i) = n - 1 - i$$

This function returns the page number symmetric to i with respect to the imaginary line that divides the set of pages into two halves of $\frac{n}{2}$ elements each.

We are ready to describe the construction of the signatures. First, we define the signatures for $m = 1$ ($n = 2$) and then show how to construct the signatures for any m given the signatures for $m - 1$. Recall that $s_i^b(m)$ and $s_i^w(m)$ represent the pages included in the $i-th$ black and white signature for a given m, respectively.

For the case $m = 1$, we have 2 signatures, $s_0^b(1) = 0$ and $s_0^w(1) = 1$. This can be represented by Figure 3.6.

	Pages	
	0	1
$s_0(1)$	●	○

Figure 3.6. Signatures for the $m = 1$ case.

For the general case, the first black signature is defined as:

$$s_0^b(m) = \{0, 1, ..., 2^{m-1} - 1\}$$

The next $m - 1$ signatures are constructed by using the first $m - 1$ signatures of the matrix used to compute files of size $2^m - 1$, and copying them over the two halves of the set of pages. That is,

$$s_i^b(m) = s_{i-1}^b(m-1) \cup sh_n(s_{i-1}^b(m-1)) \quad i = 1, 2, ..., m-1$$

Finally, the last $\frac{m(m-1)}{2}$ black signatures are built by copying all signatures for the $m - 1$ case over the first half of the pages and placing the symmetric image over the other half. That is,:

$$s^b_{m+i}(m) = s^b_i(m-1) \cup sym_n(s^b_i(m-1))$$
$$i = 0,1,\ldots,\frac{m(m-1)}{2} - 1$$

The total number of rows is consequently:

$$N_r = 1 + m - 1 + \frac{m(m-1)}{2} = \frac{m(m-1)}{2}$$

The corresponding white signatures are, of course, made up of the rest of the pages in each case. As an example, consider the case $m = 2$. In this case,

$$s^b_0(2) = \{0,1\}$$
$$s^b_1(2) = s^b_0(1) \cup sh_4(s^b_0(1)) = \{0,2\}$$
$$s^b_2(2) = s^b_0(1) \cup sym_4(s^b_0(1)) = \{0,3\}$$

Graphically, we have Figure 3.7.

	Pages			
	0	1	2	3
$s_0(2)$	●	●	○	○
$s_1(2)$	●	○	●	○
$s_2(2)$	●	○	○	●

Figure 3.7.

In a similar fashion, for the case $m = 3$ one gets Figure 3.8.

	Pages							
	0	1	2	3	4	5	6	7
$s_0(3)$	●	●	●	●	○	○	○	○
$s_1(3)$	●	●	○	○	●	●	○	○
$s_2(3)$	●	○	●	○	●	○	●	○
$s_3(3)$	●	●	○	○	○	○	●	●
$s_4(3)$	●	○	●	○	○	●	○	●
$s_5(3)$	●	○	○	●	●	○	○	●

Figure 3.8.

Using the algorithm of Figure 3.4, we now show two examples of how this construction locates two discrepancies between pages in the files. We will use in both the case $m = 3$.

For the first example, consider the case in which the files differ in pages 1 and 6. Notice that $sym_8(1) = 6$. According to Figure 3.8, both signatures in the first 3 rows are affected by these pages. In row 3, only the black signature is affected (both pages are covered by this signature). In rows 4 and 5 the white signature is the only affected. Thus, the syndrome matrix in this case is:

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

As we can see in the syndrome matrix, the only rows that are ''helpful'' in this case are the last three (in which α_t^b and α_t^w differ). The line $t = 3$ will reduce the target set to *{*0,1,6,7*}*. This set will be further reduced to *{*1,6*}* by rows 4 and 5. Since the last three rows in the checking matrix correspond to copying all signatures for the $m = 2$ case over the first half of the pages and placing the symmetric image over the other half, we can say loosely speaking, that the first half of the checking matrix is catching the discrepancy in page 1, while the second half is catching the discrepancy in page 6. Later we shall make this argument more formal.

Let us consider now a case in which the discrepancies appear over pages that are not symmetric. For instance, consider the case in which the files differ in pages 2 and 4.

The syndrome matrix in this case is:

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

The last three rows will reduce the target set to *{*2,3,4,5*}* which is the set formed by the pages that actually differ and their symmetric counterparts ($sym_8(2) = 5$ and $sym_8(4) = 3$). Fortunately, there is a matrix row among the first three, namely $t = 2$, in which the set *{*2,4*}* belong to the black signature, while the set *{*3,5*}* belong to the white one. (This translates in the syndrome matrix in a 1 in s_{t0} and a 0 in s_{t1}.) This row helps to reduce the target set to *{*2,4*}*. As we will see later the existence of this *discriminator* matrix row is no coincidence. The last $\frac{m(m-1)}{2}$ rows will always find a set composed of the actual pages and their symmetric counterparts, and there will always be a row among the first $m + 1$ which will filter the actual pages.

In what follows we make this arguments more formally. We begin with three lemmas that prove the existence of the discriminator row.

Lemma 3.1 Let $n = 2^m$ for $m \geq 2$. Let i and j be page numbers such that $0 \leq i < j \leq n - 1$. Then, there exists t with $0 \leq t \leq m - 1$, such that either

$$i \in s_t^b(m),\ j \in s_t^w(m) \qquad \text{(a)}$$

or

$$i \in s_t^w(m),\ j \in s_t^b(m) \qquad \text{(b)}$$

Proof: By induction on *m*:

Basis: For the case $m = 2$, the following table shows every possible pair for *i* and *j*, and the value of *t* for which the statement is satisfied. (See Figure 3.8.)

i,j	*t*
{0,1}	1
{0,2}	0
{0,3}	0,1
{1,2}	0,1
{1,3}	0
{2,3}	1

Inductive step: Conssider files with 2^m pages, and assume the hypotesis true for files of size $2^{m-1}, 2^{m-2}, \ldots$

Case 1: $0 \le i < j \le \frac{n}{2} - 1$

In this case, by the induction hypothesis, there exists a *t*, $0 \le t \le m-2$ such that either

$$i \in s_t^b(m-1),\ j \in s_t^w(m-1) \qquad \text{(a)}$$

or

$$i \in s_t^w(m-1),\ j \in s_t^b(m-1) \qquad \text{(b)}$$

without loss of generality, we can assume (a) is satisfied, then by construction of the checking matrix, we get

$$i \in s_{t+1}^b(m),\ j \in s_{t+1}^w(m)$$

Case 2: $\frac{n}{2} \le i < j \le n-1$. This case is analogous to case 1.

Case 3: $0 \le i \le \frac{n}{2} - 1$ and $\frac{n}{2} \le j \le n-1$.

In this case the statement is satisfied for $t = 0$. •

The following lemma shows that all of the the first *m* rows are discriminator rows if $i = sym_n(j)$.

Lemma 3.2 Let $i = sym_n(j)$, then row *t* is a discriminator for all *t*, $0 \le t \le m-1$.

Proof: By induction on *m*.

Basis: For the case $m = 2$, it is trivial to check that the first two rows are discriminators.

Inductive step: Consider true for files of size 2^{m-1} and smaller, and let the file be of size 2^m. Take any *t*, $0 \le t \le m-1$ and assume without loss of generality that $i \in s_t^b(m)$, and $i < \frac{n}{2}$. Then again by construction $i \in s_{t-1}^b(m-1)$ and by the inductive hypotesis we have $sym_{\frac{n}{2}}(i) \in s_{t-1}^w(m-1)$. Now by construction, we get $sh_n(sym_{\frac{n}{2}}(i)) \in s_t^w(m)$, but

$$sh_n(sym_{\frac{n}{2}}(i)) = \frac{n}{2} - 1 - i + \frac{n}{2} = n - 1 - i$$

$$= j$$

and the statement is true. ●

In the following lemma, we prove that for any pair of non-symmetric pages, there exists a t, such that both belong to the same signature.

Lemma 3.3 Let i and j be page numbers, with $i \neq sym_n(j)$. There exists a t, $0 \leq t \leq m-1$ such that either

$$i,j \in s_t^b(m) \tag{a}$$

or

$$i,j \in s_t^w(m) \tag{b}$$

Proof: Let $j' = sym_n(j) \neq i$. Using lemma 3.1, there exists a t, $0 \leq t \leq m-1$, such that (without loss of generality), $i \in s_t^b(m)$ and $j' \in s_t^w(m)$. But according to Lemma 3.2, for such t we have $j \in s_t^b(m)$. ●

The last part of this section proves that using the checking matrix we have described, we are able to locate discrepancies between two pages in the files. We begin with a theorem that proves that any page for which the copies differ, will appear in the output of algorithm LOCATE.

Theorem 3.1 If the two copies differ in page i, then algorithm LOCATE will have i in its output.

Proof: For every row of the syndrome matrix, the entry that corresponds to page i will be 1. Then, the other entry is either a 1, in which case the row will not be used by the algorithm, or is a 0, in which case the page i cannot be filtered out.●

The next two theorems prove that using out technique, up to two different pages can be located.

Theorem 3.2 Two copies of the same file differ only in page i iff the algorithm LOCATE returns $\{i\}$.

Proof:

If. Assume files differ only in page i. The proof is by induction on m.

Basis: For the $m = 2$ case, it is trivial to verify the theorem case by case.

Inductive step: Assume true for every file of size 2^{m-1} or less. Consider a file of size 2^m. Page i is in one of the two halves of the file, assume without loss of generality that it is the first half. By induction, it is easy to see that using the checking matrix for $m-1$ and applying the algorithm LOCATE to the first half of the file, we would get $\{i\}$ as the output. This means that the last $\dfrac{m(m-1)}{2}$ rows of the matrix for m give as a solution $\{i,i'\}$, where $i' = sym_n(i)$. Now, the first row acts as a discriminator for i,i' and we get the right result.

Only if: Assume that the output of the algorithm is $\{i\}$. If the files were identical, the algorithm would output the empty set. Thus, there is at least one difference. Furthermore, by Theorem 3.1, it is easy to see that the copies can only differ in page i. ●

Theorem 3.3 Two copies of the same file differ in pages i and j iff the algorithm LOCATE returns $\{i,j\}$.

Proof:

If. Assume files differ in pages i and j. Proof is by induction on m.

Basis: For the case $m = 2$, it is trivial to verify the theorem case by case.
Inductive step: Assume true for files of size less than or equal to 2^{m-1}. Consider a file of size 2^m.

Case 1: $i = sym_n(j)$. Without loss of generality assume $0 \leq i \leq \frac{n}{2} - 1$. By induction, if we apply the algorithm to the first half of the file using the checking matrix for $m - 1$, we get i as the output. Now, by construction, the last $\frac{m(m-1)}{2}$ rows of the checking matrix will give us i,j as output. Since i and j belong to different signatures in the first m rows (Lemma 3.2), this will be the final output.

Case 2: $i \neq sym_n(j)$. Let $i' = sym_n(i)$ and $j' = sym_n(j)$. Assume, without loss of generality that $0 \leq i \leq \frac{n}{2} - 1$. By induction, if we apply the algorithm using the matrix for $m - 1$ to the first half of the file, the output would be $\{i,j'\}$. Applying it to the second half, the output would be $\{i',j\}$. Thus, $\{i,j,i',j'\}$ will be the result given by the last $\frac{m(m-1)}{2}$ rows of the checking matrix. Using Lemma 3.3 we know that there exists a t, $0 \leq t \leq m - 1$ such that both i and j belong to the same signature, say without loss of generality, $\{i,j\} \subseteq s_t^b(m)$. By Lemma 3.2, we know that for this t, $\{i',j'\} \subseteq s_t^w(m)$. Therefore, the syndrome matrix entries for this t are $\alpha_t^w = 1$ and $\alpha_t^b = 0$, and the final solution will be $\{i,j\}$.

Only if: Assume that the algorithm gives $\{i,j\}$ as the solution, but this is not the set of pages in which the copies differ. There are two cases:

Case 1: The copies differ only in one page. But by Theorem 3.2, the output should have a single page, a contradiction.
Case 2: The copies differ in more than one page. But according to Theorem 3.1, if the copies differed in page $k \neq i,j$, this page would be in the output. So, the only pages in which they can differ are i and j. ●

4. CONCLUSIONS

In this paper we have presented a new technique for low cost file comparision. We proved that using a checking matrix of $\frac{m(m+1)}{2}$ rows, up to two differing pages can be located. Thus by sending $m(m+1)$ signatures, that is, $O(log^2(N))$, we are able to double the amount of pages that can be located with respect to the technique proposed in Fuchs et al., which uses $O(log(N))$ signatures. We believe that the extra cost paid in sending signatures is more than compensated by the ability to identify double differences quickly. As we mentioned in section 2, we believe there are many practical situations in which it is important to identify two differences.

It is still an open question to try to generalize the approach to identify three or more failures. It is conceivable that finding different sets of symmetries one may devise a scheme that identifies a larger number of differing pages. A reasonable conjecture would be that sending $O(log^m(N))$ signatures, one could identify up to m differences in the copies.

REFERENCES.

[Fu86] W.K. Fuchs, K. Wu and Abraham J. Low-Cost Comparison and Diagnosis of Large Remotely Located Files, *Proc. Fifth Symposium on Reliability in Distributed Software and Database Systems*, January 1986, pp. 67-73

[Me83] J. Metzner A Parity Structure for Large Remotely Located Replicated Data Files *IEEE Transactions on Computers*, Vol. C-32, No. 8, August 1983.

[Pi86] F. Pittelli and H. Garcia-Molina Database Processing with Triple Modular Redundancy, *Proc. Fifth Symposium on Reliability in Distributed Software and Database Systems*, January 1986, pp. 95-103

[Pi87] F. Pittelli and H. Garcia-Molina Recovery in a Triple Modular Redundancy Database System, *Proc. Seventh International Conference on Distributed Computing Systems*, Berlin, September 1987.