Data deduplication with edit errors

Laura Conde-Canencia
Lab-STICC, CNRS UMR 6285
Université Bretagne-Sud
Lorient, France
laura.conde-canencia@univ-ubs.fr

Tyson Condie
CS Department
University of California
Los Angeles, USA
tcondie@cs.ucla.edu

Lara Dolecek
ECE Department
University of California
Los Angeles, USA
dolecek@ee.ucla.edu

***Abstract*—In this paper we tackle the problem of file deduplication for efficient data storage. We consider the case where the deduplication is performed on files that are modified by edit errors relative to the original version. We propose a novel block-level deduplication algorithm with variable-lengths in the case of non-binary alphabets. Compared to hash-based deduplication algorithms where file deduplication depends on the content of the hash keys or to brute force methods that compare files symbol-by-symbol, our algorithm significantly reduces the number of symbol comparisons and achieves high *deduplication ratios*. We present a theoretical analysis on the cost of the algorithm compared to naive methods and experimental results to evaluate the efficiency of our deduplication algorithm.**

***Index Terms*—data deduplication, edit channel, insertions/deletions.**

I. Introduction

Data deduplication is an emerging technology that improves storage utilization and offers an efficient way of handling data replication in the backup environment. The principle is that redundant data blocks are removed and replaced with pointers to the unique data copy. Compared to classical data compression [1] [2], deduplication considers larger amounts of redundant data [3]. The value of deduplication is in reducing storage cost (power, cooling, floor space requirements, ...) and, obviously, the more effective the data reduction, the less disk capacity is needed for storage and the higher cost reductions can be achieved.

Data deduplication can operate at three different levels [4]: file, fixed-length block or variable-length block. A simple example of *file deduplication* would be the following: suppose the same 16 MB text file is stored in 10 folders, each copy for a different user. These 160 MB of disk space maintain the same 16 MB file. File deduplication ensures that only one complete copy is saved to disk. Subsequent replicas of the file are only saved as references that point to the saved copy. *Block-level deduplication* looks within a file and saves unique replicas of each block. Files can be broken into blocks of the same size (fixed-length block deduplication) or into blocks of various sizes depending on their contents (variable-length block deduplication). This last alternative allows the deduplication effort to achieve better deduplication ratios [5].

So far, deduplication techniques have been mostly based on hash algorithms. For each chunk (i.e., block of data that can be deduplicated), a hash value is used as a key into a hash table. If the hash table does not contain an entry with that key, the algorithm enters the complete chunk into the hash table. Then the next copies of the same chunk are replaced by the hash value. Hash algorithms may rarely produce the same hash value for two different chunks; this can lead to data loss.

The computer science community has given significant attention to the problem of deduplication [6] and large gains have been obtained for archival storage backup systems and primary storage systems. However, while there has been significant attention on the systems side, information/coding theoretic approaches have not yet been explored, except for the work in [7].

To the best of our knowledge, this paper is the first to tackle the question of deduplication under edit errors. We consider the problem of deduplication focusing on two files, X and Y, where file Y is an edited version of file X, with edit rates arbitrarily low [1] and the edits being insertions and deletions. Our approach is different to the one in [7] which focuses on substitution errors. We take advantage of the fact that data deduplication and synchronization are related problems; this observation was also made in [7]. Specifically, we consider prior work on file synchronization to propose a new block-level deduplication technique with two main goals: maximize the amount of deduplicated data compared to classical file- or block-level deduplication and reduce the number of comparisons (i.e., our algorithm must be superior to just comparing the two files until one difference is seen).

Consider the example in Fig. 1. The original file of size 16 MB is divided in blocks of sizes 4 MB (A and D), 2 MB (B1 and B2) and 1 MB (D1 to D4). Three copies of this file are manipulated by 3 different users: user 1 edits B1 and D2 (denoted as B1' and D2' after the edits), user 2 edits B2 ad D3 and user 3 does not edit the file. Without deduplication, the system needs 64 MB to store the four files; with a classical file-level deduplication technique, this amount would be reduced to 48 MB and, with a fixed block-level deduplication technique, to 32 MB (if blocks of size 4 MB) or to 22 MB (*idem* 2 MB). However, smaller blocks imply greater deduplication costs in terms of number of compared symbols[2] and system management.

In this context, we propose a deduplication technique that jointly optimizes the amount of deduplicated data (to reduce

[1]typically below 10^{-2}.

[2]as will be detailed in Section IV-A

the storage needs) and the deduplication costs. For this, we introduce an algorithm that follows the divide-and-conquer approach of [8] [9] for file synchronization. However, as for deduplication we only need to detect (and not correct) the edit errors, no interactive communication is used in our algorithm.

The deduplication technique that we propose partitions the files into alternating components, called pivots and segments (as in [9]), with the length of pivots being much smaller than the length of segments. The core element of the proposed protocol is the Consecutive Pivot (CP) Matching Module which focuses on pivots to determine the size of the chunks by comparing only a reduced number of symbols; this number being much smaller than the size of the chunk (and, of course, the size of the file).

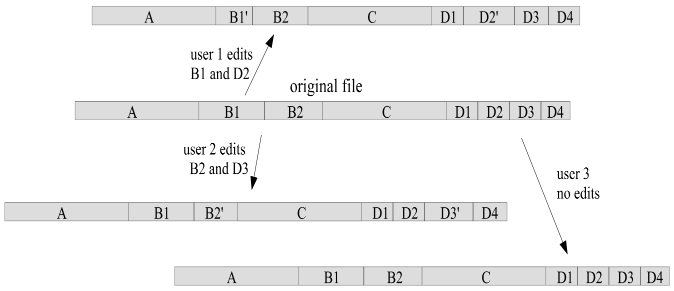

Fig. 1: Example of file partitioning for efficient deduplication.

The remainder of the paper is organized as follows: Section II presents the problem statement and notation. Section III describes the original deduplication technique as the Pivot Deduplication Algorithm. The core component of this algorithm is the CP Matching Module. A theoretical analysis on the cost of the Pivot Deduplication algorithm compared to brute force methods is provided in Section IV. In this Section we also present and discuss experimental results. Finally, Section V concludes the paper.

II. Problem statement and notation

Let X be an n-length file whose symbols are drawn from a non-binary alphabet according to an arbitrary distribution $\mu(x)$. Let Y be the output of an edit channel with input X, where insertions and deletions occur with probabilities β_I and β_D, respectively. As in [9], we consider the probability of an edit $\beta = \beta_I + \beta_D$ to be arbitrarily small, n to be large enough and the same edit channel model as in [9] with $\beta_I = \beta_D$. We denote by q the number of bits to represent a symbol of the non-binary alphabet.

For algorithmic convenience, we partition file X into substrings as $X = S_1, P_1, S_2, P_2, \ldots, S_{k-1}, P_{k-1}, S_k, P_k$, where S_i is a *segment* substring, P_i is a *pivot* substring and k is the number of both pivots and segments. The length of the *segment* substrings is L_S, the length of the *pivot* substrings is L_P and we assume the file length to be divisible by $L_S + L_P$.

A *block* is a substring that corresponds to the concatenation of one or several consecutive S_i, P_i pairs. A *chunk* is a block whose content is identical in X and Y and can thus be deduplicated. For $1 \leq i \leq j \leq n$, $X(i,j)$ denotes the substring $X(i), X(i+1), \ldots, X(j)$ of X. Note that $|X| = n$ and that the length of $X(i,j)$ is $l = j - i + 1$. Also, $n = \sum_{t=1}^{b} l_t$ where b is the number of blocks in a file, l_t is the length of block t and $t = 1, 2, \ldots, b$. Note that l_t can differ from one block to another (i.e., variable-length blocks).

We consider $L_S >> L_P$ and L_P to be small enough so that the probability that a pivot contains an edit is arbitrarily low, but long enough to include enough symbols to ensure a comparison that delivers enough information. Finally, the *deduplication ratio* is defined as the amount of deduplicated data divided by the total amount of data.

III. The Pivot Deduplication Algorithm

The goal of this algorithm is to identify the chunks in files X and Y based on a sequential comparison that only considers the pivots. This approach offers a reduced number of comparisons compared to a naive approach. With the assumption that the probability of an edit in a pivot is arbitrarily low, the background idea of the algorithm is that the edits in the segment preceding a pivot cause a mismatch in the pivot and deduplication cannot be executed for that block.

This Section is organized as follows: prior to the description of the Pivot Deduplication algorithm, some key definitions are provided. Then the core component of the algorithm, which is the CP Matching Module, is explained in details. The Section ends with some discussion on the events that affect performance of the algorithm.

A. Some definitions

Let an r-length *edit pattern* $E = E_1, E_2, \ldots, E_r$ be defined so that the output Y of an edit channel with probabilities β_I and β_D is obtained from X as follows: for $1 \leq t \leq r$, where $n \leq r < \infty$,

- If $E_t = 0$, X_j is stored and the process moves on to symbol X_{j+1}. This occurs with probability $1 - \beta_D - \beta_I$.
- If $E_t = -1$, X_j is deleted and the process moves on to symbol X_{j+1}. *Idem* with probability β_D.
- If $E_t = 1$, a new symbol taken from distribution $\mu(x)$ is inserted. *Idem* with probability β_I.

The number of *net edits* is defined as $r_{ne} = \sum_{t=1}^{r} E_t$ and corresponds to the number of insertions minus the number of deletions.

Let $P_x = (p_{x,1}, p_{x,2}, \ldots, p_{x,L_P})$ be a pivot in X and $P_y = (p_{y,1}, p_{y,2}, \ldots, p_{y,L_P})$ the corresponding pivot in Y. We define the *symbol-matcher* operator as:

$$p_{x,i} \odot p_{y,j} = \begin{cases} 1 & \text{if } p_{x,i} = p_{y,j} \\ 0 & \text{if } p_{x,i} \neq p_{y,j} \end{cases} \tag{1}$$

and the *pivot-matcher* operator as

$$P_x \odot P_y = (p_{x,1} \odot p_{y,1}, p_{x,2} \odot p_{y,2}, \ldots, p_{x,L_p} \odot p_{y,L_p}). \tag{2}$$

We consider P_y to be a *good match* of pivot P_x if $P_x \odot P_y = (1, 1, \ldots, 1)$. Note that when P_y is a good match of P_x, $r_{ne} = 0$. But $r_{ne} = 0$ does not imply that there are no edits, as the number of deletions can be equal to the number

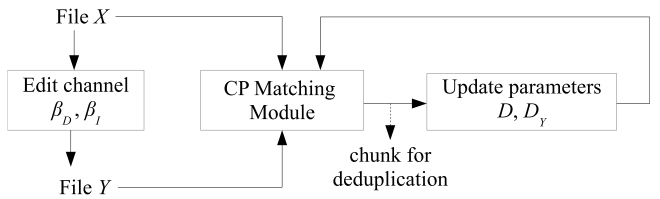

Fig. 2: Edit channel and principle of the Pivot Deduplication algorithm. File Y is an edited version of file X.

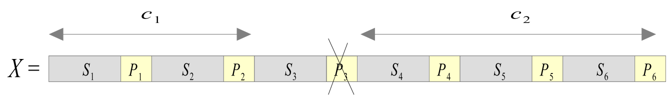
c_1
c_2
$X =$
S_1 P_1 S_2 P_2 S_3 P_3 S_4 P_4 S_5 P_5 S_6 P_6

Fig. 3: File X is divided in $k = 6$ segments and pivots: P_1 and P_2 find good matches in Y.

of insertions. This fact is later considered in the algorithm description.

Finally, we define a *shift* operator $T(P, u)$ that shifts pivot $P = (p_1, p_2, \dots, p_{L_P})$ $|u|$ positions to the right if $u > 0$ or to the left if $u < 0$. For example, for $u = 2$: $T(P, u) = (\Delta, \Delta, p_1, p_2, \dots, p_{L_P - 2})$, or $T(P, -3) = (p_4, p_5, \dots, p_{L_P}, \Delta, \Delta, \Delta)$, where Δ is a null value and $\forall p_i$ $p_i \odot \Delta = 0$.

B. *Principle of the Pivot Deduplication algorithm*

Fig. 2 illustrates the Pivot Deduplication algorithm with its different modules. File Y is the edited version of file X and both are inputs to the CP Matching Module. Each use of the CP Matching Module provides the number of consecutive matched pivots (to determine the size of the chunk for deduplication) and the number of net edits in the last segment (i.e., the one that stops the comparison). The CP Matching Module is explained in details in Section III-C.

The Pivot Deduplication algorithm can be summarized as follows:

- Consider substrings of files X and Y to start comparison at the first pivot.
- Use the CP Matching Module to determine the size of the chunk for deduplication and the number of net edits in the last segment.
- Update substrings of files X and Y to continue the comparison (with the CP Matching Module) until the end of the files is reached.

In other words, the protocol uses the CP Matching Module as many times as necessary until all the pivots in X are compared to the corresponding subsequences in Y. At the end of the protocol, the system knows which chunks can be deduplicated and which blocks contain edits and cannot be deduplicated.

The key parameters in the protocol are D, D_y and G, that correspond respectively to the first position of the pivot in file X from which the comparison starts, *idem* in file Y and the number of consecutive matched pivots (this number is provided by the CP Matching Module). The length of file Y is $n_y = n + r_{ne}$.

The protocol can then be described as follows:

```
**Data:** Files $X$ and $Y$, parameters $L_S$ and $L_P$.
**Result:** Chunks for deduplication.
**Initialization:** $D = D_y = L_S + 1$;
**while** $D < n$ **do**
    Execute CP Matching Module with inputs: $X(D, n)$
      and $Y(D_y, n_y)$;
    CP Matching Module outputs: $G$ and $r_{ne,G+1}$ ;
    Chunk for deduplication:
      $Y(D_y - L_S, D_y - L_S + G(L_S + L_P)$;
    Update parameters:
      $D_{new} = D + (G + 1) * (L_S + L_P)$ and
      $D_{y_{new}} = D_{new} + r_{ne,G+1}$
**end**
```

Example Let file X be as in Fig. 3 where only pivot P_3 does not have a good match in Y. The first output of the CP Matching Module is $G = 2$, because $r_{ne,3} \neq 0$ (i.e., edits occurred in S_3). The first chunk is identified: $Y(1, 2(L_S + L_P))$. After updating parameters D and D_y, the second output of the CP Matching Module is $G = 3$ and the second chunk for deduplication is substring $Y(D_2 + r_{ne,3}, n_y) = X(D_2, n)$ of length $3(L_P + L_S)$, where D_2 is the updated D_{new} parameter after the second use of the CP Matching Module.

The *granularity* of the protocol is directly related to the value of L_S. We can introduce *iterations* to the protocol in order to reduce the granularity in a specific block (i.e., consider a smaller L_S value at each new iteration) to improve the amount of deduplicated data or to reduce the *false detections*, as described in Section III-D.

C. *Description of the CP Matching Module*

This module is the core component of the Pivot Deduplication algorithm and its goal is to determine G and $r_{ne,G+1}$, which are the number of consecutive good pivot matches in substrings $X(D, n)$ and $Y(D_y, n_y)$, and the number of net edits in the *last segment*, respectively. This last segment is the one that precedes the pivot $P_{x,G+1}$ (i.e., the one that has no good match in Y). The CP Matching Module is used as many times as necessary to determine the chunks in X and Y.

The first pivot considered in the comparison is $P_{x,1} = X(D, D + L_P - 1)$. If $Y(D_y, D_y + L_P - 1)$ is a good match of pivot $P_{x,1}$ then the search continues for pivot $P_{x,2} = X(D + L_P + L_S, D + 2L_P + L_S - 1)$ and substring $Y(D_y + L_P + L_S, D_y + 2L_P + L_S - 1)$... and so on until there is no good match. In a more general way, for $g = 1, \dots, G$ each pivot $X(D + g(L_P + L_S), D + (g+1)L_P + gL_S - 1)$ has a good match $Y(D_y + g(L_P + L_S), D_y + (g+1)L_P + gL_S - 1)$ and there is a chunk of size $c = G(L_S + L_P)$ for deduplication. Also, if $G = k$, then the entire file Y can be deduplicated.

The number of net edits in the segment S_{G+1} is calculated as:

$r_{ne,G+1} = -u > 0$ if
$P_{x,G+1} \odot T(P_{y,G+1}, u) = (\underbrace{1, \ldots, 1}_{L_P - u}, \underbrace{0, \ldots, 0}_{u})$
and $r_{ne,G+1} = -u < 0$ if
$P_{x,G+1} \odot T(P_{y,G+1}, u) = (\underbrace{0, \ldots, 0}_{u}, \underbrace{1, \ldots, 1}_{L_P - u})$.
Or, equivalently, $r_{ne,G+1} = -u$ if

$$|r_{ne,G+1}| = L_P - \sum_{j=1}^{L_P} p_{out,j} \tag{3}$$

where $p_{out,j}$ is an element of the binary vector $P_{x,G+1} \odot T(P_{y,G+1}, u)$ and $u < L_P$.

Example Let us consider an alphabet of $2^q = 8$ symbols and $L_P = 5$. Let the beginning of file X be

$$\text{S}_{x,1}, P_{x,1}, S_{x,2}, P_{x,2} = \underbrace{7, \ldots, 4, 5, 1}_{L_S}, \underbrace{2, 3, 1, 4, 2}_{L_P}, \underbrace{3, 2, 9, \ldots, 7, 2, 1, 5}_{L_S}, \underbrace{3, 2, 1, 4, 5}_{L_P} \tag{4}$$

and the corresponding edit pattern:
$E = \underbrace{0, 0, \ldots, 0}_{L_S + L_P} \underbrace{-1, +1, 0, 0, \ldots, 0, -1, -1}_{L_S}, \underbrace{0, \ldots, 0}_{L_P}$
so that the beginning of file Y is $\underbrace{7, \ldots, 4, 5, 1}_{L_S}, 2, 3, 1, 4, 2, \underbrace{2, 4, 9, \ldots, 7, 2, 3, 2}_{L_S}, 1, 4, 5, 3, \ldots$

First pivot-matching is $P_{x,1} \odot (2, 3, 1, 4, 2) = (1, 1, \ldots, 1)$, so $r_{ne,1} = 0$. Second pivot-matching is $P_{x,2} \odot T((1, 4, 5, 3, 1), 2) = (3, 2, 1, 4, 5) \odot (\Delta, \Delta, 1, 4, 5)$, so $r_{ne,2} = -2$. The CP Matching Module outputs $G = 1$ and $r_{ne,G+1} = -2$.

D. Edits in pivots and false detections

There are three kinds of events that affect the performance of the Pivot Deduplication algorithm in terms of deduplication ratio and data loss:

- Edits in a pivot: even if L_P is chosen to be short enough so that the probability of edits in a pivot is arbitrarily low, this event may occur. The algorithm is able to detect it but cannot recover from it to continue the search in the remaining data in the file. This event does not cause data loss but reduces the deduplication ratio.
- Insertion in the last position of pivot: in practice this is considered by the algorithm as an edit in the segment that follows the pivot. This does not cause data loss but may reduce the deduplication ratio; the reduction occurs only if that segment had no edits.
- False detection: if the number of net edits in a segment is zero but there are edits (i.e., the number of deletions equals the number of insertions). This causes data loss that can be avoided by introducing iterations to the algorithm.

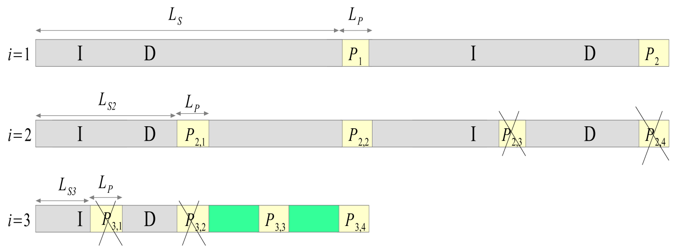

Fig. 4: Pivot Deduplication algorithm with 3 iterations. I represents an insertion and D a deletion.

E. Iterations in the Pivot Deduplication algorithm

False detections can be avoided by adding iterations to the algorithm. The first iteration ($i = 1$) provides chunks with a minimum size of $L_S + L_P$ and discards the segments with edits. Iteration $i = 2$ only considers the chunks from iteration $i = 1$ as inputs for the CP Matching Module, with a smaller granularity: $L_{S_2} = \frac{(L_S + L_P)}{2} - L_P$ and $L_{P_2} = L_P$. In general, for iteration i the CP Matching Module uses parameters: $L_{S_i} = \frac{L_S + L_P}{2^{i-1}} - L_P$, and $L_{P_i} = L_P$ with $i \geq 2$, and L_S, L_P being the parameters of the first iteration.

Each iteration provides new chunks or discards blocks. Fig. 4 shows an example where at $i = 1$ the protocol outputs a chunk of size $2(L_S + L_P)$ that corresponds to false detections ($r_{ne} = 0$ for both segments but there are edits). Iteration 2 then considers segments of length L_{S_2}, discards two of the four segments and outputs a chunk of size $2(L_{S_2} + L_P)$. Iteration 3 considers segments of length L_{S_3}, discards two of the 4 blocks and outputs a chunk of size $2(L_{S_3} + L_P)$.

IV. Theoretical analysis and experimental results

A. Theoretical analysis

In order to evaluate the benefits of the Pivot Deduplication algorithm compared to naive or brute force methods (BFM), we derive expressions for the average number of compared symbols in a n-length file for both techniques. This number will be considered as the *cost* of the algorithm, which is a function of the edit rate $\beta = \beta_I + \beta_D$. The BFMs compare data symbol-by-symbol. In our study we consider file- and fixed-block-level deduplication [4].

The file-level BFM compares on average:

$$N_F = \sum_{i=1}^{n-1} i\beta(1-\beta)^{i-1} + n[\beta(1-\beta)^{n-1} + (1-\beta)^n] \tag{5}$$

symbols, considering that the comparison stops as soon as an edit is detected. In case of block-level deduplication, the average number of compared symbols is

$$N_B = k_B \sum_{i=1}^{L_B - 1} i\beta(1-\beta)^{i-1} + L_B[\beta(1-\beta)^{L_B - 1} + (1-\beta)^{L_B}] \tag{6}$$

where k_B is the number of blocks in the n-length file and L_B is the length of a block. In this case, we consider that when

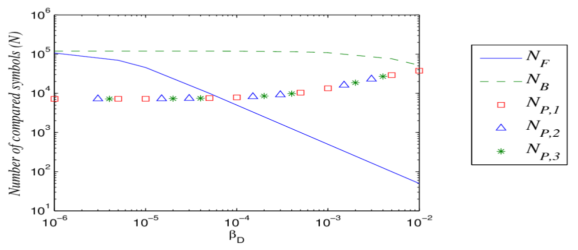

Fig. 5: Number of compared symbols (N) as a function of β_D; $n = 120000$, $L_S = 94$, $L_P = 6$ (in symbols), $k = k_B = 1200$. Subscripts are: F for file-level and B for block-level naive methods; P, i for Pivot Deduplication with i iterations.

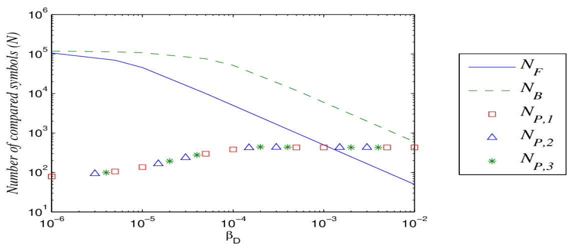

Fig. 6: Number of compared symbols (N) as a function of β_D; $n = 120000$, $L_S = 9994$, $L_P = 6$ (in symbols), $k = k_B = 12$. Same subscripts as in Fig. 5.

an edit is detected, the comparison continues at the beginning of the following block.

For the Pivot Deduplication algorithm, the number of compared symbols for the first iteration is:

$$N_{P,1} = L_P k(1-\beta)^{L_S} + (L_P)^2 k[1-(1-\beta)^{L_S}] \qquad (7)$$

where the first term represents the case of $r_{ne} = 0$ without edits [3] and the second term represents the other cases. Note that $(L_P)^2$ symbols are compared to solve (3).

When considering 2 iterations, the number of compared symbols is:

$$N_{P,2} = N_{P,1} + 2L_P k \sum_{i=1}^{L_S/2} \binom{L_S}{2i} (\beta_D)^{2i}(1-\beta)^{L_S-2i} \qquad (8)$$

where the second term represents false detections. Each segment with a false detection involves the comparison of two pivots (i.e., $2L_P$ symbols) in the second iteration. For further iterations, an equivalent equation can be written. However, it can be shown that $N_{P,1} \approx N_{P,2} \approx N_{P,3} \ldots$.

Figures 5 and 6 show the evolution of the Pivot Deduplication algorithm cost function for two different values of k and k_B, and a fixed file size of 1.2×10^5 symbols. Obviously, the benefits of the Pivot Deduplication algorithm in terms of cost increase with smaller values of k and β. For example, the cost of the Pivot Deduplication algorithm is a thousand times smaller than the one of the BFM (block-and file-level) for $\beta_D = 10^{-6}$ and $k = 12$. In the cases considered, the cost of the block-level BFM is higher than the equivalent Pivot Deduplication algorithm [4]. Also, iterations in the Pivot Deduplication algorithm do not introduce any significant additional cost.

B. *Experimental results*

We evaluate performance of the Pivot Deduplication algorithm and the BFM in terms of data deduplication ratios and false detections through simulation. For this, we generate 1000 pairs of files X and Y, where symbols in X are independent identically distributed according to an arbitrary distribution $\mu(x)$ with an alphabet of size $2^q = 64$. File Y is an edited version of X with the edit channel model described in Section III-A and we consider a file size $n = 120000$ symbols (i.e., 720 million bits are simulated to obtain each value in a curve).

Fig. 7 shows the evolution of the data deduplication ratio as a function of β_D (note that we consider $\beta_D = \beta_I$ in the channel model) for the Pivot Deduplication algorithm, the file-level BFM and the block-level BFM for different block sizes. The Pivot Deduplication algorithm simulations consider 1 to 3 iterations and 3 different values of L_S. Note that, except for the case $L_S = 9994$, no significant improvement of the deduplication ratio is obtained with iterations. However, iterations are crucial to reduce the false detection ratio (i.e., amount of data in blocks identified as false detections). Fig. 8 shows reduction factors of up to 1.5 orders of magnitude in the false detection ratio from iteration 1 to 3, for the three different L_S values.

The highest data deduplication ratios are obtained with the BFM with the lowest granularities ($L_B = 100$ and $L_B = 1000$, Fig. 7). However, for edit probabilities below 10^{-4}, the cost of the BFM is more than 10 times higher than the one of Pivot Deduplication algorithm for $L_B = 100$ (Fig. 5) and about 100 times higher for $L_B = 1000$ (not explicitly shown in any Figure, but easily computed from (6) and (7) or extrapolated from Figures 5 and 6). The file-level BFM provides very poor data deduplication ratios, even close to zero for $\beta_D = 10^{-6}$. Note that this performance would be the one of hash algorithms operating at file-level and that smaller file sizes would show better data deduplication ratios. The false detection ratios introduced with the Pivot Deduplication algorithm (Fig. 8) show that with *enough* iterations false detection ratios tend to zero, as the Pivot Deduplication algorithm tends to BFM because L_S becomes arbitrarily small (i.e., $L_P \approx L_B$).

We now consider the influence of the pivot length on the deduplication ratio. In Section III-D, edits in pivots were identified as one kind of event that significantly affects deduplication ratio in the Pivot Deduplication algorithm. As discussed, the algorithm can detect these edits but cannot recover from them, so after an *edited pivot* the rest of file is discarded. In Fig. 9 we show data deduplication ratios for

[3]For simplicity, we have omitted the case $r_{ne} = 0$ with edits because the contribution is very small compared to the first term in the equation and can thus be neglected.

[4]We consider that BFM and Pivot Deduplication algorithm are equivalent when $L_B = L_P + L_S$.

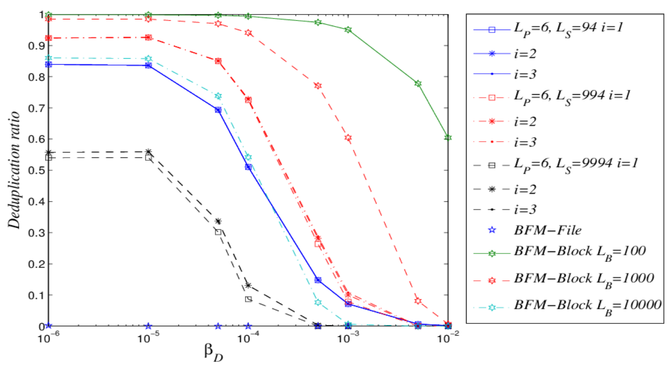

Fig. 7: Data deduplication ratio as a function of β_D (or β_I). Pivot Deduplication (for 3 different L_S and iterations $i = 1, 2$ and $i = 3$) is compared to file- and block-BFM.

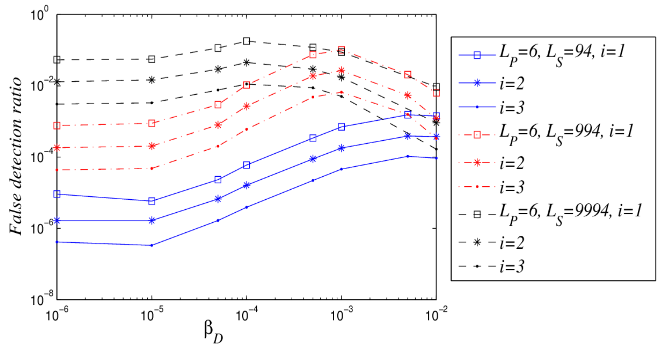

Fig. 8: False detection ratio as a function of β_D (or β_I) for $L_S = 94, 994$ and 9994, $L_P = 6$. Iterations from 1 to 3.

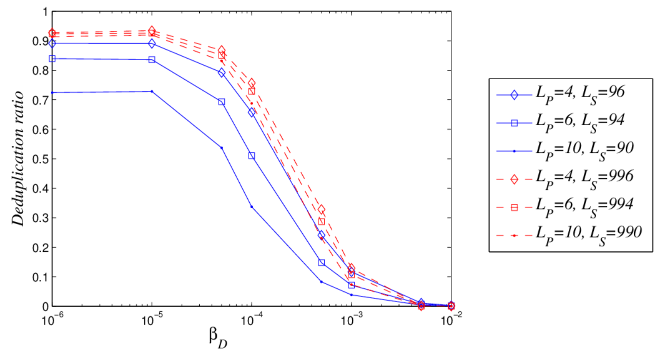

Fig. 9: Effect of L_P in the data deduplication ratio. Values of L_P are 4, 6 and 10.

$L_P = 4, 6, 10$ and $k = 120, 1200$. The highest deduplication ratio is obtained for $L_P = 4$ and $k = 120$, which are the values that minimize the probability of edit in a pivot. Note that this probability can be written as: $1 - (1 - \beta)^{L_P k}$, so it grows with L_p for a given k. However, as introduced in Section II, the value of L_p must be a tradeoff, i.e., short enough to minimise the probability of edits in a pivot and long enough so that r_{ne} can be computed as in (3). We can consider that $L_P > 2r_{ne}$ is a sufficient condition to compute (3). On the other hand, $P(r_{ne} = i) = \binom{L_S}{2i}(\beta_D)^{2i}(1-\beta)^{L_S-2i}$ decreases very rapidly with i. For example, for $L_S = 94$ and $\beta_D = 10^{-3}$, $P(r_{ne} = 1) \approx 10^{-3}$, $P(r_{ne} = 2) \approx 10^{-6}$ and $P(r_{ne} = 3) \approx 10^{-10}$. Furthermore, for $\beta_D = 10^{-6}$, $P(r_{ne} = 1) \approx 10^{-9}$ and $P(r_{ne} = 2) \approx 10^{-18}$. So we can conclude that $L_P = 4$ constitutes a reasonable tradeoff.

C. On the efficiency of the Pivot Deduplication algorithm

The Pivot Deduplication algorithm constitutes a robust deduplication solution in the context of edit errors (specially for edit probabilities below 10^{-4}) and real time data reduction applications, where the rapidity of the protocol is a priority. For k values in the order of 10, the Pivot Deduplication algorithm reduces the cost of the BFM by factors up to 1000, but with *poor* deduplication ratios and non negligible false detection ratios (around 10^{-3}). However, for k values in the order of 1000 the Pivot Deduplication algorithm reduces the cost of the BFM by a factor of 10 for edit probabilities below 10^{-3}, with false detection ratios under 10^{-4}. Considering that additional iterations can be performed without significant additional cost, these false detection ratios can be reduced until they meet some given criteria.

V. Conclusion

This paper was dedicated to data deduplication in the context of edit errors. We proposed an algorithm that efficiently performs file deduplication. This algorithm focuses on pivots in the file and its parameters can be adjusted to minimize the cost or maximize performance in terms of data deduplication ratio and false detections. We derived theoretical expressions to compare the proposed algorithm to brute force methods and we presented experimental results to evaluate the data deduplication ratios for different values of parameters. The proposed algorithm is particularly interesting for low edit rates and real time data reduction.

References

[1] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, May 1977.

[2] J. Cleary and I. Witten, “Data compression using adaptive coding and partial string matching,” *IEEE Transactions on Communications*, vol. 32, no. 4, pp. 396–402, Apr 1984.

[3] A. El-Shimi, R. Kalach, A. Kumar, A. Ottean, J. Li, and S. Sengupta, “Primary data deduplication—large scale study and system design,” in *Annual Technical Conference*, Boston, MA, 2012, pp. 285–296.

[4] A. Venish and K. S. Sankar, “Study of chunking algorithm in data deduplication,” *Proc. of Int. Conf. on Soft Computing Systems, Advances in Intelligent Systems and Computing, Springer India*, 2016.

[5] “Quantum white book: Effectiveness of variable-block vs fixed-block deduplication on data reduction: A technical analysis,” www.quantum.com, Tech. Rep., 2015.

[6] L. L. You, K. T. Pollack, and D. D. E. Long, “Deep store: an archival storage system architecture,” in *21st International Conference on Data Engineering (ICDE’05)*, April 2005, pp. 804–815.

[7] U. Niesen, “An information-theoretic analysis of deduplication,” in *IEEE Int. Symp. on Information Theory (ISIT)*, June 2017, pp. 1738–1742.

[8] S. M. S. T. Yazdi and L. Dolecek, “A deterministic polynomial-time protocol for synchronizing from deletions,” *IEEE Transactions on Information Theory*, vol. 60, pp. 397–409, 2014.

[9] F. Sala, C. Schoeny, N. Bitouze, and L. Dolecek, “Synchronizing files from a large number of insertions and deletions,” *IEEE Transactions on Communications*, vol. 64, no. 6, pp. 2258–2273, June 2016.