Received August 7, 2020, accepted August 21, 2020, date of publication August 31, 2020, date of current version September 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3020501

On the Complexity and Performance of the Information Dispersal Algorithm

RICARDO MARCELÍN-JIMÉNEZ[1], JORGE LUIS RAMÍREZ-ORTÍZ[1], ENRIQUE RODRÍGUEZ DE LA COLINA[1], MICHAEL PASCOE-CHALKE[1], AND JOSÉ LUIS GONZÁLEZ-COMPEÁN[2]

[1]Department of Electrical Engineering, Universidad Autónoma Metropolitana, CDMX 09340, Mexico
[2]CINVESTAV, Tamaulipas 87130, Mexico

Corresponding author: Ricardo Marcelín-Jiménez (calu@xanum.uam.mx)

This work was supported by Consejo Nacional de Ciencia y Tecnología (CONACYT) of Mexico.

ABSTRACT The Information Dispersal Algorithm (IDA) has become a key component in several fault-tolerant massive storage systems. From a theoretical point of view, it is a linear transformation over a finite field on the vectors that make up a given file. Direct transformation adds redundancy, splitting the initial file into a new set of files called dispersals. The inverse transformation recovers the original file from a subset of dispersals. This piece of research demonstrates the impact of input and output (I/O) operations on direct and inverse transformations. Different alternatives to control the exchange of elements between RAM and disk were evaluated, which is the key operation to build a vector in memory and store its entries in a file. First, the impact of the working finite field was tested; second, the impact of the use of a buffer for exchange between the RAM and the hard disk, and finally, several instances of the algorithm with which to evaluate the impact of parallelism were simultaneously deployed. The results demonstrate that the combination of these factors may have an important effect on the speed of both direct and inverse procedures.

INDEX TERMS IDA, fault tolerance, reliability, massive storage.

I. INTRODUCTION

The information dispersal algorithm (IDA) [1] has drawn the attention of an increasingly large community of researchers from academia and industry alike. This may be due not only to the elegance of its approach, but also to the possibilities it opens up for communications and storage applications. In this latter area, in particular, IDA offers the opportunity to build massive storage systems in which the difference between raw or gross capacity and effective capacity is reduced, while supporting the same level of fault tolerance found in other copy or replica-based systems. Besides, being a parameterizable algorithm, it allows for the identification of different trade-offs between redundant information and the robustness of the solutions that are built.

On the other hand, to the extent that the algorithm has left the research laboratory to be incorporated into production systems, a more detailed study of its associated costs has also been initiated. However, our research revealed that many of these studies focus on the computational complexity of the algorithm for some particular working conditions, in which some kind of fast algorithm can be applied.

In contrast, this study initially addressed the importance held by the order of the working finite field, but the experiments demonstrated that this offer is only a possibility to control file reading and writing operations, which have the greatest impact on the costs involved. This paper presents the results obtained from a series of experiments in which the importance of exchange operations on the algorithm's performance has been recognized.

The rest of this document is organized as follows: Section II reviews works focused on IDA and its potential. Section III presents a description of the theoretical foundations of the algorithm is presented, as well as the analysis of the complexity of its operations. Section IV describes the agenda of experiments included in this study and analyzes the results. Finally, section V brings together the most outstanding findings and presents a set of conclusions, as well as future work directions.

II. RELATED WORK

After the original publication of the IDA, Rabin himself proposed extending the scope of his initial work and

The associate editor coordinating the review of this manuscript and approving it for publication was Cristian Zambelli.

suggested some applications for storage [2]. The use of coding techniques to optimize storage space has gained relevance in recent years, to the extent that systems of greater capacities are being built and the gap between raw capacity and effective capacity is being reduced [3].

Some authors also suggest the possibility of using the algorithm not only to optimize storage capacity, but also to lower energy consumption [4]. Another work proposes sending a file through a channel with delay problems, varying the parameters of the algorithm to send a quantity of redundant information that depends on the state of the channel [5].

Among the massive storage systems that use IDA or a similar approach, Scality [6], a commercial product for massive storage recently acquired by HPE, could be mentioned. There is also Cleversafe [7], a storage project recently acquired by IBM. Furthermore, the organization in charge of developing CEPH [8] announced the use of coding techniques similar to IDA to optimize large-scale storage. In turn, RedHat offers a version of CEPH that incorporates these features. In addition, the Babel system [9], which uses an IDA implementation of its own, is also worth mentioning.

In [10]–[12], there is a set of solutions for building efficient and flexible end-to-end cloud storage. Information integrity is preserved based on the IDA implementation embedded in parallel processing patterns. In [13], the authors propose the construction of a storage system mounted on a Hadoop platform, incorporating IDA as the coding technique.

With regard to the reduction of the algorithm's complexity, a couple of works can be found that emphasize the improvement of the costs associated with arithmetic operations, using an approach reminiscent of the Fast Fourier Transform (FFT) [14], [15]. A couple of works that use an information theory perspective to study a generalization of IDA and the limits of these types of constructions [16], [17] are also worth mentioning. Authors in [18] propose a data recovery algorithm based on modular arithmetic resembling the IDA method applied to distributed storage. Results show significant benefits concerning processing time. Authors in [19] use a Reed-Solomon code which is closely related to IDA in order to improve data integrity and availability in cloud storage systems. This solution is also an effective alternative to data replication. In [20], the authors analyze IDA's limitations in supporting secure storage, proposing alternative schemes. Finally, in [21] the original Rabin's IDA is further modified to improve computational performance, security, and integrity by combining the All-Or-Nothing Transform with the Optimized Cauchy Reed-Solomon code.

III. THEORETICAL ASPECTS

Let F be an arbitrary file of size $8|F|$ (bits). Using IDA, F is transformed into n files, called *dispersals*, each of which has a size $8|F|/m$ (bits), such that any m of them are sufficient to recover the original file. Also $1 < m < n$. It is evident that the total amount of information that is produced is $8n|F|/m$ (bits). In other words, an excess of information equal to $8|F|(n/m - 1)$ (bits) is produced. The amount in parentheses is called the stretching factor or *stretching*, for short. Let $k = n - m$ be the *fault tolerance* or the number of missing dispersals that can be tolerated. Then, the same value of k can be achieved with different stretching values. Thus, it would seem interesting to tolerate a fixed number of faults with the least stretching possible. Later on, this paper will argue that as far as the robustness of a storage system based on this approach is concerned, this option is not necessarily the best.

A. THE LINEAR TRANSFORMATION

In practice, IDA comprises two complementary processes: *dispersion* and *recovery*. Dispersion uses a transformation matrix A over a finite field made up of n rows and m columns. Any combination of m out of n rows in A builds a set of linearly independent vectors. These vectors make up a square submatrix that is invertible. This property is achieved, for instance, using the construction of a Vandermonde matrix [14].

The input file F is also regarded as a succession of vectors $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_L$ over the same field as A. Each vector has m coordinates, i.e. has m dimensions. Using a linear transformation, which is the core of the dispersion process, each vector $\vec{b}_i,\ i = 1, \ldots, L$, is turned into a new vector $\vec{c}_i$ of n coordinates, according to the following operation,

$$A\vec{b}_i = \vec{c}_i. \tag{1}$$

Supposing m coordinates are selected from $\vec{c}_i$ in a completely arbitrary way in order to produce a new vector $\vec{d}_i$, this is equivalent to saying that $\vec{c}_i$ loses $k = n - m$ of its coordinates or, alternatively, is obtained from $\vec{b}_i$ through the following linear transformation,

$$B\vec{b}_i = \vec{d}_i, \tag{2}$$

in which B is built through choosing m rows of A, exactly from the same positions of the coordinates of $\vec{c}_i$ as contained in $\vec{d}_i$. From the properties of A, we know that the m rows in B are linearly independent. Therefore, B is invertible. Ultimately, $\vec{b}_i$ can be rebuilt according to the following operation,

$$\vec{b}_i = B^{-1}\vec{d}_i. \tag{3}$$

This transformation is referred to as the inverse algorithm, but it is better known as the recovery process.

This research investigates the impact of finite fields $GF(2^8)$ and $GF(2^{16})$ as working fields. The first case comprises the set of 8-bit strings, whereas, the second one comprises the set of 16-bit strings. The reason for focusing on these lengths is that the basic reading and writing operations in any computer are carried out by using elements with a length which is a multiple of 8 bits. For $GF(2^8)$, a polynomial generator $g(x) = x^8 + x^6 + x^5 + x^4 + 1$ was used, whereas, for $GF(2^{16})$ a generator $g(x) = x^{16} + x^{12} + x^3 + x + 1$ was used, and both are primitive polynomials.

Finally, it is worth mentioning that the elementary arithmetic operations on the selected working fields were implemented based on two complementary methods called the discrete logarithm and antilogarithm, respectively [22]. With this approach, any basic operation can be carried out in constant time.

B. COMPLEXITY

Focusing first on the case of $GF(2^8)$, as the working field, for dispersion, the input file F is understood as a sequence of vectors. Each vector has m entries or coordinates and each entry has 8 bits. Therefore, if F has $|F|$ bytes, it can be decomposed into L_1 vectors such that

$$L_1 = |F|/m.$$

Next, each vector is multiplied by matrix A, which has n rows and m columns. Therefore, for each vector, n times m products and n times $m-1$ additions are calculated, which means that the following is obtained for the entire file

$$\begin{aligned} nmL_1 &= n|F| \text{ products,} \\ n(m-1)L_1 &\approx n|F| \text{ additions.} \end{aligned}$$

Finally, each vector $\vec{b}_i$ is transformed into a new vector $\vec{c}_i$, with n entries. The first coordinate of $\vec{c}_i$ is written in the first dispersal, the second coordinate in the second dispersal, and so on, for each of its n coordinates. Since there is a vector $\vec{c}_i$ for each vector $\vec{b}_i$ and each coordinate of $\vec{c}_i$ produces a writing operation, the total number to be executed is

$$nL_1 = n|F|/m \text{ writings.}$$

As for the readings, the analysis is reduced to the fact that the input file is read in 8-bit units, i.e., bytes, and therefore the total of operations that are carried out is

$$|F| \text{ readings.}$$

Instead, during recovery, the first step is the construction of matrix B of size $m \times m$, which has to be inverted. Inversion has an $O(m^3)$ complexity. To make up vector $\vec{d}_i$ of m entries, an 8-bit element is read from each of the surviving dispersals. Each recovered vector $\vec{b}_i$ comes from a vector $\vec{d}_i$ multiplied by the inverse of B. Therefore, if there are L_1 vectors $\vec{b}_i$, there will be the same number of vectors $\vec{d}_i$, each having m entries. In other words, the total of readings to carry out will be

$$mL_1 = |F| \text{ readings.}$$

The multiplication of matrix B^{-1} and vector $\vec{b}_i$ implies, for each row of the matrix, m individual multiplications and $m-1$ additions, therefore since there are m rows, m^2 products and $m(m-1)$ additions are obtained. However, knowing that there are as many vectors $\vec{d}_i$ as L_1, there will be a total number of

$$\begin{aligned} m^2L_1 &= m|F| \text{ products,} \\ m(m-1)L_1 &\approx m|F| \text{ additions.} \end{aligned}$$

Finally, each of the original vectors $\vec{b}_i$ with m 8-bit entries is recovered. Therefore, m writings were performed for each $\vec{b}_i$, and there are L_1 vectors of this type, which accounts for

$$mL_1 = |F| \text{ writings.}$$

As for $GF(2^{16})$, this time, input file F is also understood as a sequence of vectors. Each vector has m entries or coordinates, but now each entry has 16 bits. Therefore, if F has $|F|$ bytes, it can be decomposed into L_2 vectors such that

$$L_2 = |F|/2m.$$

From this point forward, the analysis only changes by the factor L_2, instead of L_1, which means that for dispersion the following overall number of operations are carried out,

$$\begin{aligned} nmL_2 &= n|F|/2 \text{ products,} \\ n(m-1)L_2 &\approx n|F|/2 \text{ additions,} \\ nL_2 &= n|F|/2m \text{ writings,} \\ &\quad |F|/2 \text{ readings.} \end{aligned}$$

As for recovery, there are

$$\begin{aligned} m^2L_2 &= m|F|/2 \text{ products,} \\ m(m-1)L_2 &\approx m|F|/2 \text{ additions,} \\ mL_2 &= |F|/2 \text{ writings,} \\ mL_2 &= |F|/2 \text{ readings.} \end{aligned}$$

To summarize, during dispersion, the cost of arithmetic operations does not depend on m, but only on n. As for I/O operations, a total of $|F|$ bytes are read, and $n|F|/m$ bytes are written, which means that outputs are very sensitive to the n/m ratio. That is, for a fixed n, the cost is maximum when $m \to 1$ and becomes marginal when $m \to n$. This decay comes from factor $1/m$.

Instead, during recovery, the cost of the arithmetic operations depends linearly on m only. Notice that when $m \to n$, stretching is reduced but it is necessary to process a larger number of dispersals to recover the original file. Meanwhile, for I/O operations, a total of $|F|$ bytes are read and the same number of bytes are written. Therefore, the I/O operations have a fixed cost.

IV. EXPERIMENTS AND RESULTS

This paper presents the experiments developed to assess the direct and inverse algorithms' performance and the impact of different factors on their execution times. These experiments were gathered into subsets called *families* in order to organize the results. A family of experiments is defined by all the possible combinations of parameters n and m, where $1 < m \leq 14$, $2 < n \leq 15$, and $1 < m < n$. Each family of experiments was executed for both $GF(2^8)$ and $GF(2^{16})$, thereafter referred to as IDA8 and IDA16, respectively. This design aims to contrast the effect of the working field order within each family. Each experiment corresponds to a particular combination of parameters n and m. Let $C_j = (m_j, n_j)$ be the j-th individual experiment. Then the set of all combinations is ordered lexicographically, that is, $C_1 = (3, 2)$, $C_2 = (4, 2)$, $C_3 = (4, 3), \ldots, C_{79} = (15, 2), \ldots, C_{91} = (15, 14)$, as it is described in Table 1. It is also important to point out that all of the families were carried out under the same environment, that is, the same hardware and software were used as reported in Table 2. Besides, the HDD was reformatted for each family of experiments and the file system cache was emptied at the beginning of each experiment.

TABLE 1. Combination of parameters (n, m) for each experiment C_j.

C_1:(3,2)	C_{19}:(8,5)	C_{37}:(11,2)	C_{55}:(12,11)	C_{73}:(14,8)
C_2:(4,2)	C_{20}:(8,6)	C_{38}:(11,3)	C_{56}:(13,2)	C_{74}:(14,9)
C_3:(4,3)	C_{21}:(8,7)	C_{39}:(11,4)	C_{57}:(13,3)	C_{75}:(14,10)
C_4:(5,2)	C_{22}:(9,2)	C_{40}:(11,5)	C_{58}:(13,4)	C_{76}:(14,11)
C_5:(5,3)	C_{23}:(9,3)	C_{41}:(11,6)	C_{59}:(13,5)	C_{77}:(14,12)
C_6:(5,4)	C_{24}:(9,4)	C_{42}:(11,7)	C_{60}:(13,6)	C_{78}:(14,13)
C_7:(6,2)	C_{25}:(9,5)	C_{43}:(11,8)	C_{61}:(13,7)	C_{79}:(15,2)
C_8:(6,3)	C_{26}:(9,6)	C_{44}:(11,9)	C_{62}:(13,8)	C_{80}:(15,3)
C_9:(6,4)	C_{27}:(9,7)	C_{45}:(11,10)	C_{63}:(13,9)	C_{81}:(15,4)
C_{10}:(6,5)	C_{28}:(9,8)	C_{46}:(12,2)	C_{64}:(13,10)	C_{82}:(15,5)
C_{11}:(7,2)	C_{29}:(10,2)	C_{47}:(12,3)	C_{65}:(13,11)	C_{83}:(15,6)
C_{12}:(7,3)	C_{30}:(10,3)	C_{48}:(12,4)	C_{66}:(13,12)	C_{84}:(15,7)
C_{13}:(7,4)	C_{31}:(10,4)	C_{49}:(12,5)	C_{67}:(14,2)	C_{85}:(15,8)
C_{14}:(7,5)	C_{32}:(10,5)	C_{50}:(12,6)	C_{68}:(14,3)	C_{86}:(15,9)
C_{15}:(7,6)	C_{33}:(10,6)	C_{51}:(12,7)	C_{69}:(14,4)	C_{87}:(15,10)
C_{16}:(8,2)	C_{34}:(10,7)	C_{52}:(12,8)	C_{70}:(14,5)	C_{88}:(15,11)
C_{17}:(8,3)	C_{35}:(10,8)	C_{53}:(12,9)	C_{71}:(14,6)	C_{89}:(15,12)
C_{18}:(8,4)	C_{36}:(10,9)	C_{54}:(12,10)	C_{72}:(14,7)	C_{90}:(15,13)
-	-	-	-	C_{91}:(15,14)

TABLE 2. Common experimental conditions.

O.S.	Centos 7
Processor	Core i7@3.2 GHz
RAM	64 GB
HDD	SATA, 8.0 TB, 7200 rpm

A. FAMILY 1

All the corresponding experiments were performed on a 120 MB file. A procedure called *pseudo IDA* was also developed. It consists of reading and writing the same amount of information involved in either the direct or inverse algorithms, but without performing any type of processing. In this way, the I/O times are measured exclusively. If they are subtracted from the total times, then the time taken by the execution of the arithmetic operations can be identified. A new performance measure called *processing speed* was also established. It was defined as the amount of information received (from either the source file or the surviving dispersals) expressed in megabits, divided by the time it takes to process this information in order to generate the output file(s) (dispersals or recovered file, respectively).

With regard to the time involved, arithmetic operations are practically negligible compared to R&W operations which turn out to be very expensive. In fact, it was discovered that arithmetic operations do not take more than 2 percent of the total time. It was also observed that working on 16 bits shows a small advantage over the 8-bit field, but this benefit vanishes compared to I/O times. As for the graph corresponding to the dispersion (direct) algorithm, a sawtooth pattern arises, which can be explained by the amount of information that is produced. During dispersion, a file of $|F|$ bits is read and the algorithm produces n dispersals of $|F|/m$ bits each, yielding a total of $n|F|/m$ bits to be written back to the disk. This means that a quantity of redundant information is generated. Meanwhile, for the recovery algorithm, the input is m dispersals of $|F|/m$ bits each, which totals $|F|$ bits, and the output is the recovered file, which also has $|F|$ bits. If the R&W operations have the greatest impact on the performance of both algorithms, the redundant information that must be written explains the aforementioned pattern, which only occurs in the case of the direct algorithm. Based on this analysis and for the sake of brevity, only the dispersion results will be shown (Fig. 1).

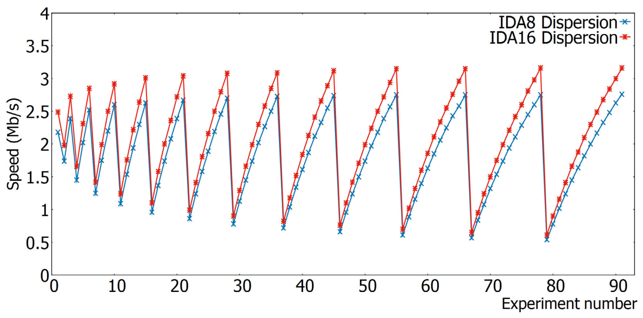


FIGURE 1. Family 1, dispersion processing speed.

B. FAMILY 2

From the preceding results, a bottleneck related to the transformations involved in (1) and (3) was found, corresponding to the dispersion and recovery processes, respectively. In both cases, before the matrix-vector multiplication is carried out, it is necessary to assemble an input vector by reading each of its entries from a file, which is stored in a disk and, finally, this multiplication produces an output vector the entries of which must be written into a file, back to the disk. This bottleneck can be explained as a consequence of the difference between the speed of arithmetic operations and the speed of the disk and RAM exchange. Instead of exchanging one single entry at a time, it was decided to upload/download a chunk of the file using two intermediate buffers in RAM. A swap buffer was implemented in order to bring file chunks from the disk and another one to take file chunks to disk. This way, vector manipulation is separated into two decoupled steps: vector assembly/disassembly on RAM and chunk exchange between buffers and disk. All the corresponding experiments for both IDA8 and IDA16 were performed, but this time such an important improvement on the processing speed was achieved that a bigger file was used to measure the involved times more accurately. A 1 GB file was therefore chosen.

Results show that the buffers have an important impact on the total times of direct and inverse algorithms so that larger files can be processed in less time, compared to the previous family of experiments. In this scenario, arithmetic operations have a comparable duration in relation to I/O operations. In this set of experiments, the difference between using 8 or

16 bits as vector entries is more important and it is clear how IDA16 outperforms IDA8. The processing speed grows by a factor of 100 (roughly) compared to the speed measured in the preceding family.

The chunk size has an optimal value that is around 100kB. This can be explained as a trade-off (as can be conjectured) between two conflicting operations involved in the disk: reading/writing or carrying information. A very small chunk size implies poor reading and fast carrying. Meanwhile, a very large chunk size implies the contrary. It should be mentioned that the sawtooth patterns are now observed in dispersion as well as recovery. This is evidence of the weight or importance of arithmetic operations that are now found in the same proportion as I/O operations (Fig. 2).

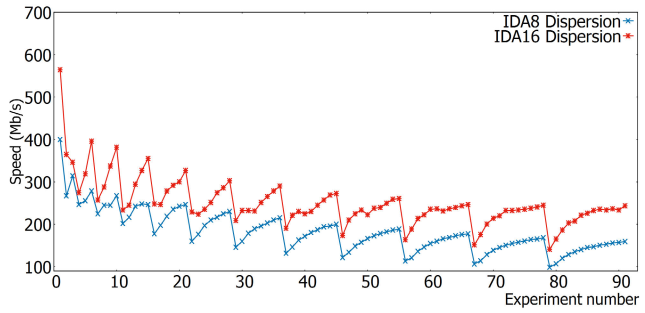


FIGURE 2. Family 2, dispersion processing speed for the optimal chunk size.

C. FAMILY 3

In this set of cases, all the corresponding experiments were performed on the same file as the preceding family. Nevertheless, there was a need to evaluate the impact of the underlying HW which provides a 12-cores CPU and can implicitly support parallelism. That is, as many concurrent executions of the algorithm as i times the number of cores ($i = 1, 2, 4$) were launched. This approach was found to reduce the idle times associated with I/O operations. The difference in speed between 8 and 16 bits appeared to be minimal when m was rather small. Also, the processing speed increased again by a factor of 3 - 5 times, compared to the results of the preceding set of experiments. Finally, the processing speed did not change significantly when the number of concurrent instances of the algorithm was increased from 12 to 48 threads (Fig. 3).

D. ANALYSIS

In all the cases that were addressed, the cost of inverting the matrix involved in the recovery process had a negligible impact on the overall performance of the inverse algorithm. This can be explained by the fact that despite the complexity of matrix inversion, which is a cubic function of m, it cannot be compared with the massive volume of the other arithmetic operations that, as was seen, hinge on the size of the source file.

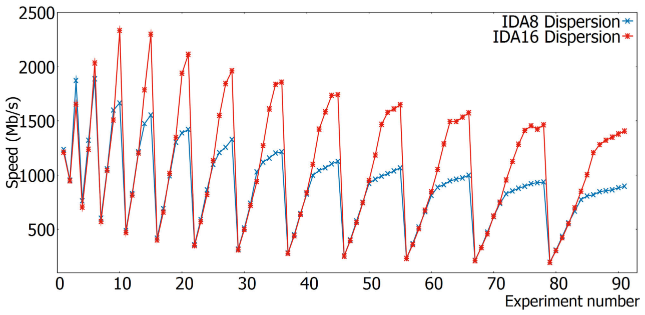


FIGURE 3. Family 3, dispersion processing speed for 12 threads.

The most expensive case for dispersion occurred for a fixed value of n and $m = 2$, which at the same time induced the (local) minimum value on the cost of recovery and vice versa. That is, for a fixed value of n and $m \rightarrow n$, the cost of dispersion reached its local minimum, while the recovery obtained its local maximum. This implies that, from a landscape view, the overall most expensive case on dispersion always happened under the combination (15, 2), whereas the most expensive case of recovery always happened under the combination (15, 14). Furthermore, we must consider that there are alternative ways to address the costs of the algorithm since this study shows the performance for each value of $n \in [3, 15]$ and the corresponding possibilities for m. For an IDA-based storage system, a value of n as big as 15 does not seem convenient, as will be argued later. Indeed, a value of between 5 and 10 can provide all the advantages of the algorithm without reaching the limit of the costs that has been demonstrated.

Finally, it seems pertinent to think about what could be considered a good combination of m and n. For a fixed $k = n - m$, it is possible to have different combinations of n and m but, although $m \rightarrow n$ would seem a good option because the stretching is significantly reduced, the probability that k or more faults occur increases with n.

Consider, for example, two instances of IDA with parameters (5, 3) and (15, 13), respectively. Both offer the same tolerance, but the second case also provides a lower cost in redundancy (their stretching values being 0.666 vs 0.153, respectively). The second case offers a better stretching, but the probability that there are two or more failures as a function of n should be considered. This is obtained by modeling the probability of faults with a binomial distribution in which p is the probability that a storage component crashes. Given n components, the probability that two or more faults occur must be calculated, but this is the complementary probability that zero or one failures occur. The result is a function that grows with the value of n. The conclusion is that although it seems interesting to increase n, while bounding the number of failures that can be tolerated $(n - m)$, the downside is that this criterion can lead to greater risk. It is also important to consider, as has already been established, that a large value of n implies longer processing times. Therefore, alternative $m \rightarrow n$, in which n takes a very large value (e.g. > 10) does not seem very convenient, since the cost of processing rises but, above all, the risk in which the storage system is set also rises, especially when their components age and increase their probability to fail.

V. CONCLUSION

I/O operations play the most important role in IDA performance and can be improved in several ways: i) by defining a vector's coordinates in units greater than 8 bits in order to accelerate data exchange; ii) by building a swap device that decouples RAM usage from HD (e.g., using a buffer or a solid state disk); and iii) by using some form of parallelism. A measure called processing speed was defined in order to compare the effect of each of these mechanisms in an objective and fair way. The aggregate effect of all the aforementioned factors is a speed that increases 500 times on average, from the first to the last family of experiments. A method to characterize the cost of I/O exchange and separate it from the cost of arithmetic operations was built. It was found that there is an optimal value in the size of the chunk exchanged between disk and swap buffers that does not depend on the working field, file size, or parallelism, but seems to be determined by the HD profile. Findings apply to all algorithms involving linear transformation (e.g., FFT).

As far as I/O operations are concerned, dispersion costs strongly depend on the combination of parameters m and n, which may produce an excess of redundant information to be stored, whereas recovery always works with the same amount of data, regardless of these parameters. This explains why this approach has a deeper effect on dispersion, but nonetheless is still important for recovery.

The results obtained in the experiments are perfectly explained through the complexity calculations presented previously. However, there is a scale factor that can be extremely relevant, which is determined by the technology that supports the processing and the exchange of vectors that make up the input and output files.

It is important to mention that all the elementary arithmetic operations carried out in this work were based on discrete logarithm and antilogarithm methods, which grant each basic operation a fixed execution time. This was a key requirement for the study. With this approach, two tables were constructed based on the powers of the generator element of the working field. The most serious drawback was the size of its supporting data structures, since the main interest lies in the atomic strings with a length which is a multiple of 8 bits. Therefore, for IDA8 each table had 2^8 entries and for IDA16, 2^{16} elements. The possibility of working with a higher order field was not considered due to the involved complexity of constructing its corresponding tables. Even though there exist alternative computational methods, which are commonly used in cryptography, for instance, they are beyond the initial scope.

Finally, this study provides the designer with criteria to choose the combination of working conditions that best suits the needs for a particular application. At the same time, this piece of research has demonstrated that there is a large area of opportunity related to I/O operations. Future research will study the impact of the file system and its influence on the size of the swap buffer. It is also worth considering the impact of an alternative disk unit, such as a solid state disk (SSD).

ACKNOWLEDGMENT

The authors wish to express their gratitude to the anonymous reviewers whose valuable comments helped to improve the quality of this work.

REFERENCES

[1] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. ACM*, vol. 36, no. 2, pp. 335–348, Apr. 1989.

[2] M. O. Rabin, "The information dispersal algorithm and its applications," in *Sequences: Combinatorics, Compression, Security, and Transmission*, vol. 1, 1st ed. New York, NY, USA: Springer, 1990, pp. 406–419.

[3] O. T. Lee, S. D. M. Kumar, and P. Chandran, "Erasure coded storage systems for cloud storage—Challenges and opportunities," in *Proc. Int. Conf. Data Sci. Eng. (ICDSE)*, Cochin, India, Aug. 2016, pp. 23–25.

[4] A. Afianian, S. S. Nobakht, and M. B. Ghaznavi-Ghoushchi, "Energy-efficient secure distributed storage in mobile cloud computing," in *Proc. 23rd Iranian Conf. Electr. Eng.*, Tehran, Iran, May 2015, pp. 740–745.

[5] A. Bestavros, "An adaptive information dispersal algorithm for time-critical reliable communication," in *Network Management and Control*, vol. 2, 1st ed. Boston, MA, USA: Springer, 1994, pp. 423–438.

[6] P. Speciale, "Scality RING: Scale out file system & Hadoop over CDMI," in *Proc. 7th Storage Developer Conf.*, Santa Clara, CA, USA, 2014, pp. 15–18.

[7] A. Cleversafe, "Paradigm shift in digital assest storage," Cleversafe Whitepaper, 2008. [Online]. Available: https://www.evaluatorgroup.com/document/object-storage-cleversafe-dsnet/

[8] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, Seattle, WA, USA, 2006, pp. 307–320.

[9] M. Quezada-Naquid, R. Marcelín-Jiménez, and J. L. González-Compeán, "Babel: The construction of a massive storage system," *Int. J. Web Services Res.*, vol. 13, no. 4, pp. 36–53, Oct. 2016.

[10] D. Carrizales, D. D. SÃĺnchez-Gallegos, H. Reyes, J. L. González-Compeán, M. Morales-Sandoval, J. Carretero, and A. Galaviz-Mosqueda, "A data preparation approach for cloud storage based on containerized parallel patterns," in *Proc. Int. Conf. Internet Distrib. Comput. Syst.*, Naples, Italy, 2019, pp. 478–490.

[11] M. Santiago-Duran, J. L. Gonzalez-Compean, A. Brinkmann, H. G. Reyes-Anastacio, J. Carretero, R. Montella, and G. T. Pulido, "A gearbox model for processing large volumes of data by using pipeline systems encapsulated into virtual containers," *Future Gener. Comput. Syst.*, vol. 106, pp. 304–319, May 2020.

[12] J. L. Gonzalez-Compean, V. Sosa-Sosa, A. Diaz-Perez, J. Carretero, and J. Yanez-Sierra, "Sacbe: A building block approach for constructing efficient and flexible end-to-end cloud storage," *J. Syst. Softw.*, vol. 135, pp. 143–156, Jan. 2018.

[13] J. Ling and X. Jiang, "Distributed storage method based on information dispersal algorithm," in *Proc. 2nd Int. Symp. Instrum. Meas., Sensor Netw. Autom. (IMSNA)*, Toronto, ON, Canada, Dec. 2013, pp. 624–626.

[14] Y. D. Lyuu, "Information dispersal," in *Information Dispersal and Parallel Computation*, 1st ed. Cambridge, U.K.: Cambridge Univ. Press, 2004, ch. 2, pp. 8–22.

[15] S.-J. Lin and W.-H. Chung, "An efficient (n, k) information dispersal algorithm based on fermat number transforms," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 8, pp. 1371–1383, Aug. 2013.

[16] P. Béguin and A. Cresti, "General information dispersal algorithms," *Theor. Comput. Sci.*, vol. 209, nos. 1–2, pp. 87–105, Dec. 1998.

[17] A. De Santis and B. Masucci, "On information dispersal algorithms," in *Proc. IEEE Int. Symp. Inf. Theory*, Lausanne, Switzerland, Jun. 2002, p. 410.

[18] M. Deryabin, N. Chervyakov, A. Tchernykh, V. Berezhnoy, A. Djurabaev, A. Nazarov, and M. Babenko, "Comparative performance analysis of information dispersal methods," in *Proc. 24th Conf. Open Innov. Assoc. (FRUCT)*, Moscow, Russia, Apr. 2019, pp. 67–74.

[19] M. Singh and S. Singh, "A framework for cloud storage system based on information dispersal algorithm," *Int. J. Recent Technol. Eng.*, vol. 7, no. 6C, pp. 145–148, 2019.

[20] L. Yao, J. Lu, J. Liu, D. Wang, and B. Meng, "A secure and efficient distributed storage scheme SAONT-RS based on an improved AONT and erasure coding," *IEEE Access*, vol. 6, pp. 55126–55138, 2018.

[21] H. Lahkar and C. R. Manjunath, "Towards high security and fault tolerant dispersed storage system with optimized information dispersal algorithm," *Int. J. Adv. Res. Comput. Sci.*, vol. 5, no. 6, pp. 286–291, 2014.

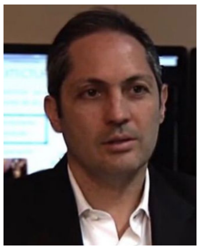
[22] F. J. MacWilliams and N. J. A. Sloane, "Finite fieds," in *The Theory of Error Correcting Codes*, 1st ed. Amsterdam, The Netherlands: Elsevier, 1977, ch. 4, pp. 93–124.

RICARDO MARCELÍN-JIMÉNEZ was born in Mexico City, Mexico, in 1965. He received the B.S. degree in electronics engineering from the Autonomous Metropolitan Universidad-Iztapalapa (UAM-I), Mexico City, in 1987, the M.S. degree in computer engineering from the National Polytechnic Institute (CINVESTAV-IPN), Mexico, in 1992, and the Ph.D. degree in computer science from the National Autonomous University of México (UNAM), Mexico, in 2004.

He is currently a Full Researcher and Professor with the Department of Electrical Engineering, UAM-I. He is the author of three books, more than 50 articles, and one invention. His research interests include the theory and practice of distributed computing, especially issues related to coordination and fault tolerance.

Dr. Marcelín-Jiménez is a Level I member of the CONACYT's National Research System (SNI), Mexico.

JORGE LUIS RAMÍREZ-ORTÍZ was born in Mexico City, Mexico, in 1973. He received the B.S. degree in electronics engineering and the M.S. degree in information technologies from the Autonomous Metropolitan Universidad-Iztapalapa (UAM-I), Mexico City, in 2002 and 2011, respectively. Since 2004, he has been working as a private consultant, involved in the design and construction of massive storage systems. His research interests include distributed computing and software engineering. For both degrees, he received a medal for academic merit, awarded to the Most Outstanding Student.



ENRIQUE RODRÍGUEZ DE LA COLINA was born in Mexico City, Mexico, in 1968. He received the B.S. degree in electronics engineering from the Autonomous Metropolitan Universidad-Iztapalapa (UAM-I), Mexico City, in 1994, the M.S. degree in computer science from the Autonomous Metropolitan Universidad-Azcapotzalco (UAM-A), Mexico City, where he was awarded the University Merit Medal, in 1998, the Diploma degree in management skills from the Autonomous Technological Institute of Mexico (ITAM), in 2001, and the Ph.D. degree in engineering in photonic communications systems from the University of Cambridge, U.K., in 2009.

He collaborated as a Postdoctoral Associate Researcher with the Broadband Communications Group and Distributed Systems, University of Girona, Spain. He has been a Professor-Researcher in the area of networks and telecommunications with the Department of Electrical Engineering, UAM-I, since August 2010. He is currently a coordinator of the postgraduate studies in sciences and information technologies at UAM-I. Prior to joining academia, he worked in the industry for more than 15 years as an engineer, project manager, and consultant for various telecommunications companies. His areas of research interest include cognitive radio networks, community intranetworks, mission-critical wireless communications systems, and high-capacity networks, including optical, and satellite networks. He is a Level I member of CONACYT's National Research System (SNI).

MICHAEL PASCOE-CHALKE was born in Mexico City, Mexico, in 1970. He received the B.S. degree in mechanical-electrical engineering and the M.S. and Ph.D. (Hons.) degrees in electrical engineering from the National Autonomous University of Mexico (UNAM), in 1997, 2005, and 2010, respectively.

He is currently an Associate Professor with the Metropolitan Autonomous University-Iztapalapa (UAM-I), Mexico City. His research interests include wireless communications, location systems, and cognitive radio networks. He is a Level I member of the CONACYT's National Research System (SIN).

JOSÉ LUIS GONZÁLEZ-COMPEÁN was born in Ciudad Valles, San Luis Potosí, Mexico, in 1973. He received the Ph.D. degree in computer architecture from the UPC Universitat Politecnica de Catalunya, Barcelona, in 2009.

He was a Visiting Professor with the Universidad Carlos III de Madrid, Spain, and a Researcher with CINVESTAV, Mexico. His line of research includes cloud-based storage systems, distributed computing, software construction models, and infrastructure-agnostic storage and processing solutions for edge cloud environments. His research interests include the design of fault-tolerant systems, adaptability and availability strategies, task scheduling, and storage virtualization.

Dr. González-Compeán is a Level I member of CONACYT's National Research System (SNI).