Searching on Non-Systematic Erasure Codes

Atthapan Daramas
Department of Computer Science
The University of Waikato
Hamilton, New Zealand
ad134@students.waikato.ac.nz

Vimal Kumar
Department of Computer Science
The University of Waikato
Hamilton, New Zealand
vimal.kumar@waikato.ac.nz

***Abstract*—Non-Systematic erasure codes provide confidentiality of data and can even provide strong security guarantees with appropriate parameter selection. So far however, they have lacked an elegant and efficient method for keyword search over the codes. While the obvious method of reconstruction before search can be too slow, direct search produces inaccurate results. This has been one of the barriers in the wider adoption of non systematic erasure codes in distributed and secure storage. In this paper we present an elegant solution to this problem by building an index data structure that we call the *Search Vector*, created from the generator matrix of the erasure code. We show that this method introduces a very small amount of delay in return of a high degree of accuracy in search results. We also analyse the security of the scheme in terms of information leakage and show that the information leaked from the index data structure is very small even when one assumes the worst case scenario of the attacker having access to the *Search Vector*.**

***Index Terms*—Keyword Search, Erasure Codes, Information Dispersal Algorithm, Information Retrieval, Non-Systematic Erasure Coding, Cloud Storage**

I. Introduction

Cloud storage is widely used to reduce the cost of data storage and for the convenience of being able to access data from anywhere over the internet. The convenience of using cloud storage, however, also has a trade-off. Clouds tend to be opaque when it comes to the techniques and technologies they use under the hood. This results in data owner's loss of control over data and the possibility of the loss of confidentiality of data as well. When trying to store data securely on the cloud, in addition to confidentiality, availability is another criteria that a data owner is usually concerned about. A data owner should use Cloud Service Providers (CSPs) that promise the highest availability versus cost ratio as well as employ mechanisms to increase availability of the data itself.

There are two methods to increase availability of the data, namely, replication and erasure coding. While replication is easier to implement, erasure coding provides higher availability gain and mean time to failure, and uses less bandwidth than replication [1]. The erasure coding schemes achieve availability by generating additional data fragments which can be used to reconstruct failed fragments. Examples of cloud storage services that can be configured to use erasure codes are IBM Spectrum Scale Erasure Code Edition [2], OceanStore [3], Red Hat Ceph Storage 3 [4], and MinIO [5] among others. The fragments of erasure-coded data need to be stored separately so that if one fragment fails others can be retrieved safely. The fragments can be stored separately on different CSPs or in different regions within the same CSP. For personal users, cloud services like Amazon S3, Microsoft Azure, RackSpace, etc. can be configured as storage services for erasure-coded fragments as described in [6], and [7]. When it comes to generating the fragments of erasure coded data, there are two approaches that are generally followed resulting into systematic, and non-systematic erasure coding. While the performance improvement of erasure coding has received considerable attention in the research literature [8], [9], the ability to search for keywords in erasure codes hasn't been studied much. Keyword search on systematic erasure codes is trivial since the data is stored in plaintext, however, searching on non-systematic erasure codes is not straightforward because the stored data is encoded. We demonstrate the hardness of directly searching on non-systematic erasure codes in section IV-B.

Contribution. In this paper we describe a technique to enable searching on non-systematic erasure codes. We first build an index data structure from the erasure code's generator matrix that we call the *Search Vector* and then show how the *Search Vector* can be used for searching on the erasure coded data. We also provide the proof of security of our technique and evaluate its performance in the paper.

The rest of the paper is structured as follows. We start with background and related work where erasure coding and searchable encryption are explained. We then describe the system model we will work with before discussing two inefficient methods of searching over non systematic erasure codes in section IV. In section V, we present our data structure for searching on non-systematic erasure codes called *Search Vector*. We also demonstrate how this data structure is used for searching. We then analyse the performance as well as the security of our search technique and present the results in section VI. Finally, we conclude the paper in section VII.

II. Background & Related Work

A. Erasure Coding

Erasure coding is used for error correction in data transmission and data storage. It can recover original data from an *erasure* scenario where data is lost during transmission or storage failure. The coding scheme splits a file into fragments. Additional fragments are then generated from the original data fragments. These fragments are collectively called *codewords*.

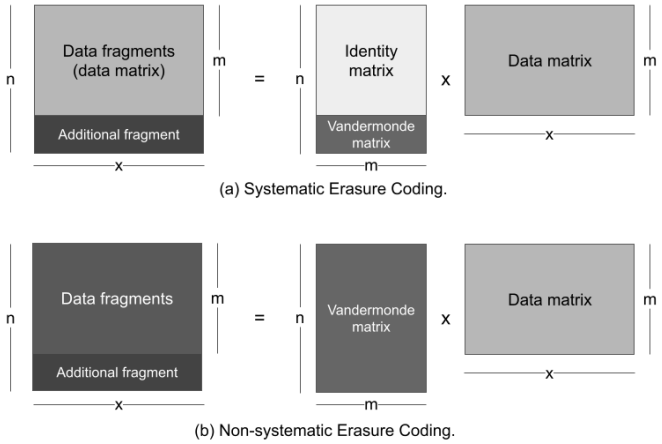

Fig. 1. Code Word Matrix Generation.

An erasure coding scheme with *n* codewords generated from *m* fragments of original data is called an *m-of-n* erasure coding scheme. Such a coding scheme can tolerate up to *(n - m)* of lost fragments. In case of data loss or storage failure, the surviving code words can be used to reconstruct the lost data. Well-known erasure coding schemes include, Reed-Solomon Erasure coding, flat XOR, and Rabin's Information Dispersal Algorithm (Rabin's IDA).

In Reed-Solomon and Rabin's IDA, the additional fragments are generated from multiplication between fragments of original data and a generator matrix. First, the original data is split into *m* parts and rearranged into an $m \times x$ matrix called the data matrix where $x = \lceil total\ file\ size/m \rceil$. Then, the data matrix is multiplied with an $n \times m$ generator matrix. A common generator matrix is a combination of $m \times m$ identity matrix and $(n - m) \times m$ Vandermonde or Cauchy matrix. Given generator matrix *A*, and data matrix *D*, a code word matrix *E* can be generated as shown in eq. (1).

$$\underset{n\times x}{\mathrm{E}} = \underset{n\times m}{\mathrm{A}} \cdot \underset{m\times x}{\mathrm{D}} \tag{1}$$

In eq. (1), the mentioned generator matrix results in what is called the systematic erasure code. Fig. 1(a) shows one notable characteristic of systematic erasure code that is the first *m* rows of the code word matrix are actually the rows of the data matrix. An example of this type of erasure code is Reed-Solomon erasure coding. On the other hand, as shown in Fig. 1(b) a generator matrix that is generated from $n \times m$ Vandermonde or Cauchy matrix will result in a non-systematic erasure coding, e.g., Rabin's IDA erasure coding. Since the generator matrix does not contain an identity matrix, the first *m* rows of code word matrix are not the data matrix in non-systematic erasure coding.

In systematic erasure codes as depicted in Fig. 1(a), the first *m* codewords are the same as the data matrix, leading to the data being stored in cleartext. In a non-erasure scenario therefore, no additional computation is required when data needs to be used. On the other hand, non-systematic erasure codes as shown in Fig. 1(b) are not in cleartext and so require additional computation to convert the fragments back to data. For this reason systematic erasure codes have enjoyed considerable popularity over non-systematic erasure codes. Non-systematic erasure codes, however, provide the significant benefit of confidentiality due to their randomised nature. As described in [10] and [11], careful selection of parameters in the code generation process can even provide strong confidentiality guarantees. The usage of non-systematic codes therefore has been explored in secure distributed storage such as in [12] and [13].

B. Searchable Encryption

Due to being randomised, searching on these codes is not as straightforward and has been a major drawback in their wider uptake. In this paper, we tackle the problem of searching on non-systematic erasure codes. This problem has not been studied before and there is a lack of authoritative literature on it. A related problem that has been studied well in the literature is of Searchable Encryption (SE). SE techniques allow a user to store encrypted data on an untrusted server while maintaining search functionality. The techniques can be classified into two types, sequential, and index-based search.

Sequential search was first introduced by Song et. al in [14] and refers to direct search on the encrypted data. As the name suggests, when searching for a keyword this technique performs sequential search on every file in the storage, which results in $O(n)$ search time where *n* is the total number of files. On the other hand, an index-based approach builds an index by extracting keywords from the files before encrypting and storing the data [15], [16]. The survey by Poh et al. [17] describes that the index can be built using three different approaches; direct index, inverted index, and tree-based index. The notable advantage of index-based search is that it is faster than sequential approach. However, this approach may have higher computational complexity on file updates. In order to achieve accurate results, the keywords index must be synchronised with the files and therefore, the index must be updated after any file is updated.

We borrow the concepts of sequential search as well as index from SE. Our technique as explained in section V creates an index type data structure from the original data. The underlying concept of our proposed technique is that the search keyword will be transformed into a suitable form for searching. Then, the transformed keyword will be used for sequential search on the data structure.

III. System Model

The system model we use in this paper is presented in Fig. 2. We assume that the data owner wants to store a file in cloud storage(s). The file is encoded using an *m-of-n* erasure coding scheme to generate *n* codewords. Each codeword is then stored in a separate storage.

IV. Problem Formulation

When considering searching on non systematic erasure codes one can think of two straightforward methods; reconstruction before search and direct search. In this section, we

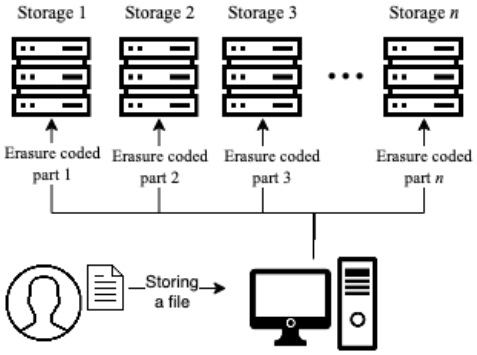

Fig. 2. System Model

TABLE I
AVERAGE SEARCHING TIME FROM FILES RECONSTRUCTION.

Total file size (MBs)	**16.1**	**32.4**	**48.7**	**65.5**
Files Reconstruction time (s)	4.000	7.587	11.193	14.723
Search time (s)	0.017	0.034	0.043	0.053
Total time usage (s)	4.017	7.621	11.236	14.776
Search time percentage	0.43%	0.44%	0.38%	0.36%

discuss these two methods and explain the issues that render them impractical.

A. Computational Overhead

As described in section II-A, non-systematic erasure coding schemes store all data in encoded code words. The simplest method for searching therefore is to reconstruct the files from the code words and search on the reconstructed file.

We performed a small experiment to illustrate the issues with this approach. In the experiment we erasure coded 4 datasets of varying sizes with varying number of files and performed search after reconstruction. The erasure coding scheme that is used in the experiments is *3-of-4* erasure coding. Table I shows the average time for reconstruction and searches for specific keywords on local storage. As expected the average total time spent increases with the increase in file size. However, the most important finding is that the time spent searching never exceeds 0.44% of the total time spent. Barring that very small amount of time spent searching, the overhead is entirely due to the reconstruction process. Additionally, if the erasure coded data was stored on the cloud, to maintain confidentiality and avoid information leakage, the reconstruction process should be performed on the user-side. This would necessitate transfer of the entire data to the user and the resulting overhead would add to the inefficiency caused by reconstruction. Therefore, although reconstructing the files before searching is simple and provides accurate searching results, the time overhead shows that the method is very inefficient.

B. Inaccurate Results

Another potential method for keyword search is to directly search on the stored data with an encoded keyword. There are two limitations to this approach. Firstly, the size of the keyword should be multiplicative of m since a single byte in the stored data is encoded from m bytes of original data. The m parameter is derived from *m-of-n* erasure coding that is used for encoding the files. A search keyword that does not obey this constraint will result in loss of data in encoded keywords. In the worst case when the length of the keyword is shorter than m, it will result in the whole keyword being discarded and consequently the inability to perform search.

Fig. 3 demonstrates the difficulty of searching using this method. Let '$b_1b_2b_3b_4b_5b_6$' be the data that will be stored, and *3-of-4* erasure code is used. Fig. 3(a) shows the process of encoding the data where the left side of equation is the stored erasure coded. Fig. 3(b) demonstrates keyword encoding method where the keyword is '$b_2b_3b_4b_5$'. Since this keyword is not multiplicative of 3, it results in the last column which contain 'b_5' being discarded. Therefore, the keyword is reduced to '$b_2b_3b_4$'.

The second constraint is that a single keyword can be encoded into multiple patterns. The byte sequence in the original file can affect the encoded data. The previous example shows that the search keyword is, in fact, encoded in the form of '$x_1b_2b_3b_4b_5x_2$' where x_1 and x_2 are b_1 and b_6 in the stored data respectively. Therefore, every possible sequence must be generated to account for all possible patterns. With the information that *3-of-4* erasure code is used in the example, the keyword will need to be padded to be multiplicative of 3. The pattern generation results in three patterns, including, '$b_2b_3b_4b_5xx$', '$xb_2b_3b_4b_5x$', and '$xxb_2b_3b_4b_5$' where x represents a padding byte. These three patterns will be encoded and sent to each storage, resulting in 12 patterns being generated. In general, the number of total encoded keywords can be formulated as $(m \times n)$ where m and n are from *m-of-n* erasure coding.

For the second constraint, let $\mathcal{K}$ be a set of encoded keywords; $\mathcal{K} = \{k_{11}, k_{12}, ..., k_{1m}, k_{21}, ..., k_{nm}\}$. Once the encoded keywords are generated, each of n storages will receive m encoded keywords from $\mathcal{K}$, i.e., k_{1*} will be sent to the first storage, k_{2*} will be sent to the second storage, and so on. Then, each storage provider searches its erasure-coded data with received encoded keywords. Finally, the results from

$$\begin{bmatrix} b_1+b_2+b_3 & b_4+b_5+b_6 \\ b_1+2b_2+4b_3 & b_4+2b_5+4b_6 \\ b_1+3b_2+9b_3 & b_4+3b_5+9b_6 \\ b_1+4b_2+16b_3 & b_4+4b_5+16b_6 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{bmatrix} \times \begin{bmatrix} b_1 & b_4 \\ b_2 & b_5 \\ b_3 & b_6 \end{bmatrix}$$

(a)

$$\begin{bmatrix} b_2+b_3+b_4 \\ b_2+2b_3+4b_4 \\ b_2+3b_3+9b_4 \\ b_2+4b_3+16b_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{bmatrix} \times \begin{bmatrix} b_2 & b_5 \\ b_3 & ? \\ b_4 & ? \end{bmatrix}$$

Discarded

(b)

Fig. 3. Keyword Byte Discarded.

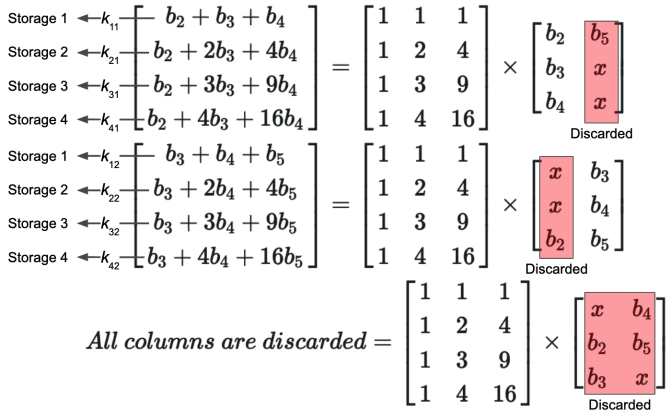

Fig. 4. Encoded Keywords Generation.

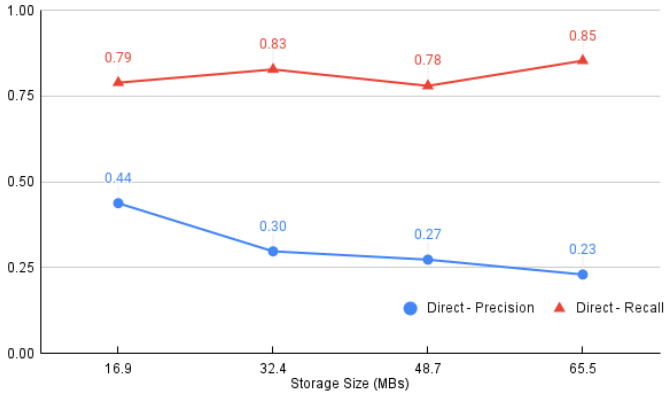

Fig. 5. Precision and Recall of Direct Searching on Erasure-coded Data.

each storage are compared. A document contains searching keyword if and only if one or more of the encoded patterns exist in all storages. i.e. all k_{i*} must exist in the document where $1 \leq i \leq n$ and $* \subseteq \{1 \leq j \leq m\}$. Note that, this constraint when combined with the former and will also result in loss of information in searching. Fig. 4 demonstrates encoded keywords generation and the problem. The keyword that will be searched on storage is $b_2b_3b_4b_5$. Each of the equations is a pattern that keyword could be encoded in the storage (the second constraint). Since some of the columns contain a padding byte *x*, the column should be discarded (the first constraint). This results in the last combination not being encoded since both of the columns contain *x*. Each row of the encoded keyword is finally sent to data storage. The document contains the keyword if and only if all storages contain one or more encoded keyword patterns (equation).

The loss of information in encoded keyword leads to inaccurate search results. The metrics, precision and recall are generally used to demonstrate the degree of inaccuracy of the results. First, the results from search are categorised into 4 groups: True Positive (*TP*: document retrieved with search keyword), True Negative (*TN*: document not retrieved without search keyword), False Positive (*FP*: document retrieved without search keyword), and False Negative (*FN*: document not retrieved with search keyword). Then, precision and recall are calculated from:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

Fig. 5 shows the precision and recall when using direct search on erasure-coded data. The data used in this experiment is the same as in the previous experiment. The horizontal axis shows the size of original files in MBs. The vertical axis shows value of precision and recall calculated from the search results. The circle and triangle data points represent precision and recall respectively. It is noticeable that the precision gradually decreases with the size of data to be searched on, while the recall values are more stable. This shows that direct search on erasure-coded data can retrieve most of the documents that contain the search keyword but, at the same time, it also retrieves a large number of irrelevant documents that do not contain the search keyword. As a result, this method may not be useful for accurate searching, but may be somewhat acceptable when a user want to retrieve as much data as possible.

To summarize, in this section, we described two methods to search on erasure coded data, reconstruction before search, and direct search. The former method has the advantage of correctly retrieving the results, but incurs significant delay. On the other hand, the latter method does not perform as well in the information retrieval aspect. In the next section, we propose a data structure and a technique that can be used for searching on non-systematic erasure coded data.

V. Proposed Technique

A. Data Structure for Searching

The proposed data structure is called the Search Vector, *SV*. It is an encoded vector created from original data (*D*) and erasure code's generator matrix (*A*). The purpose of having this data structure is to eliminate the reconstruction process, and avoid direct searching on erasure-coded data. To generate the Search Vector, first we create the key vector $\overrightarrow{K}$ by taking the product of the row vector $\overrightarrow{1}$ and the generator matrix *A*. $\overrightarrow{K}$ is a row vector with the same length as the number of columns in the generator matrix.

$$\overrightarrow{K} = \overrightarrow{1}.A$$

$$k_i = (\sum_{j=1}^{n} a_{j(i \mod m)})$$

Then, the keys set is recursively used for multiplication with bytes in original data which results in a vector as long as the data itself which we call the Search Vector. The Search Vector's elements can be written as:

$$sv_i = (\ b_i \sum_{j=1}^{n} a_{j(i \mod m)}\ ) \tag{4}$$

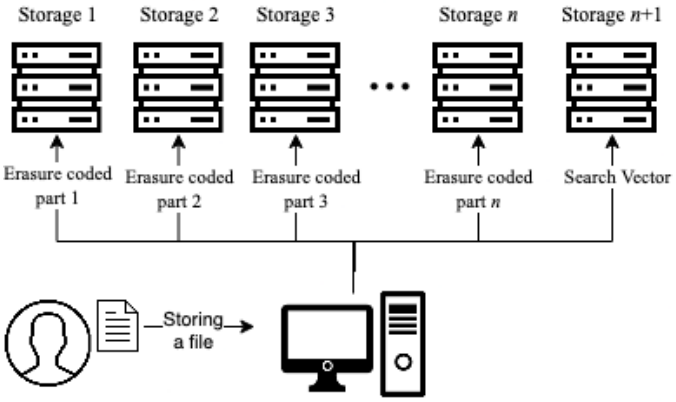

Fig. 6. System Architecture with Search Vector.

where:

$$\begin{aligned} sv_i &= \text{element of search vector at position } i \\ b_i &= \text{original byte data to be processed at position } i \\ m &= \text{parameter } m \text{ from erasure coding} \\ n &= \text{parameter } n \text{ from erasure coding} \\ a_{j(i \mod m)} &= \text{element of matrix } A \\ &\quad \text{at row } j \text{ column } (i \mod m) \end{aligned}$$

Below we demonstrate the process of generating the Search Vector through an example. We assume that *2-of-3* erasure coding is used, and the data matrix D consists of 4 elements, '$b_1b_2b_3b_4$'.

$$\begin{aligned} A &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \\ \overrightarrow{K} &= \begin{bmatrix} (a_{11}+a_{21}+a_{31}) & (a_{12}+a_{22}+a_{32}) \end{bmatrix} \\ &= \begin{bmatrix} k_1 & k_2 \end{bmatrix} \\ D &= \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \end{bmatrix} \\ SV &= \begin{bmatrix} b_1 \cdot k_1 & b_2 \cdot k_2 & b_3 \cdot k_1 & b_4 \cdot k_2 \end{bmatrix} \end{aligned}$$

Note that, these operations take place over Galois Field. Also note that the first SV key can be 0 if the Vandermonde matrix has even rows ($n \mod 2 = 0$). This is because elements in the first column in Vandermonde matrix are all 1s.

Reflecting back on the system model, the Search Vector data structure can be stored on a separate storage device/provider as depicted in Fig. 6.

B. Searching using Search Vector

Our approach for search using the Search Vector is similar to direct search. First, the search keyword is encoded to generate all possible patterns in which the keyword could appear in the Search Vector. Next, encoded keywords are used for search. For ease of implementation, we assume that one file will contain one Search Vector. We also assume that the searcher knows the key vector $\overrightarrow{K}$. As an example of searching technique, assume that *2-of-3* erasure coding is used. To generate all possible patterns for searching, $\overrightarrow{K}$ is rotated by left-shifting the elements inside by one. The overflowing element will be placed at the end of the vector. Each such left shift creates a new row. Thus, the first row of $\overrightarrow{K}$ for searching is the $\overrightarrow{K}$ vector itself, and each following row is the result of left-shifting the row above. Therefore, the $\overrightarrow{K}$ for searching matrix will be of size $m \times m$ where m is the parameter m from erasure coding. The generation of $\overrightarrow{K}$ for searching matrix is shown below:

$$\begin{aligned} \overrightarrow{K} &= \begin{bmatrix} (a_{11}+a_{21}+a_{31}) & (a_{12}+a_{22}+a_{32}) \end{bmatrix} \\ &= \begin{bmatrix} k_1 & k_2 \end{bmatrix} \\ \overrightarrow{K} \text{ for searching} &= \begin{bmatrix} k_1 & k_2 \\ k_2 & k_1 \end{bmatrix} \end{aligned}$$

The rotation of $\overrightarrow{K}$ generates all possible encoding keys that could be used to generate *SV*. Since the *SV* is stored in a single storage, the total amount of possible patterns is *m*. Then, each row of $\overrightarrow{K}$ for searching is used to encode search keyword. Let the search keyword be $q_1q_2q_3$. Since the length of the query is larger than the columns of $\overrightarrow{K}$ matrix, the keys are recursively used to mimic the SV generation. Therefore, two possible encoded keywords are generated as follows:

$$\text{possible patterns} = \begin{bmatrix} q_1 \cdot k_1 & q_2 \cdot k_2 & q_3 \cdot k_1 \\ q_1 \cdot k_2 & q_2 \cdot k_1 & q_3 \cdot k_2 \end{bmatrix}$$

After the possible patterns matrix, *P*, is generated, each row of *P* will be used to test whether it appears in the Search Vector. If one of the rows in *P* appears in the Search Vector, the original text contains the searching keywords.

VI. Evaluation

The evaluation is divided into three parts, latency analysis, information retrieval analysis, and security analysis. First, latency in searching is evaluated and compared with the searching methods presented in IV . Then we compare the correctness of results retrieved by our proposed method with those presented in section IV. Finally we discuss the security implications of our technique. The data source that is used for the first two analyses is from [18]. The original purpose of the dataset was for binary sentiment classification for movie reviews. The dataset is separated into 4 categories, positive feedback for training, negative feedback for training, positive feedback for testing, and negative feedback for testing. Each category of data has a different size aiding us in experimenting with varying data size. The experiments are carried out with *3-of-4* erasure coding scheme. In total we have tested 500 queries for each category, or 2,000 queries in total.

A. Latency Analysis

To reflect the problem of traditional search techniques presented in section IV-A, the evaluation begins with comparing search time usage.

Fig. 7 shows the average of total searching time for one query. On the horizontal axis we show the size of the data

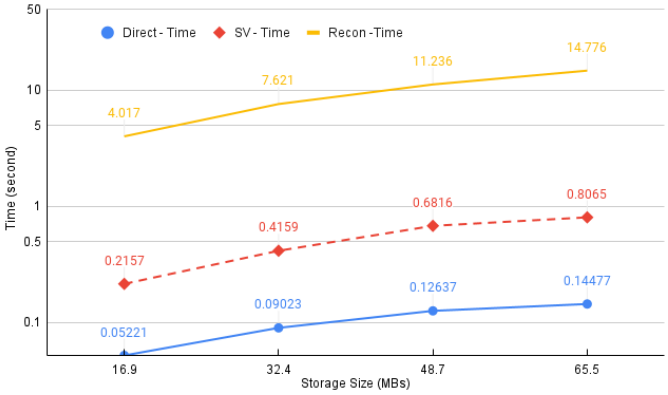

Fig. 7. Time Usage for Searching.

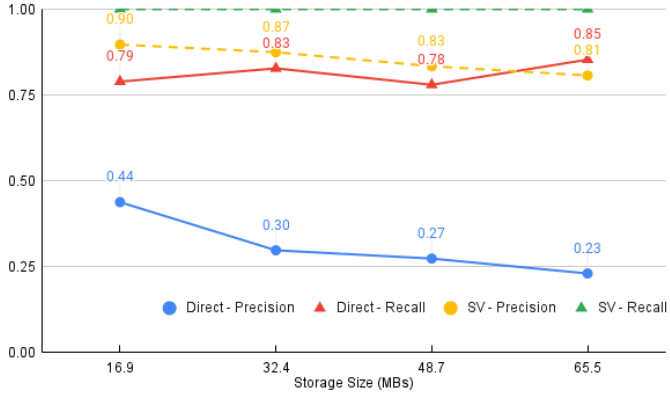

Fig. 8. Precision and Recall with Different Searching Methods.

stored in the system. The vertical axis represents time used for searching in seconds. Please note that this axis is non linear so that we can accommodate data curves in widely different ranges. Overall, we can see that as expected, each search methods' time usage increases with the size of data. The proposed technique is noticeably faster than the reconstruct-before-search technique. Our technique, however, incurs more latency than direct search. It needs to be noted that these results exclude the communication time between user and data storage since the experiments have been carried out on local storage. As mentioned in section IV-A, factoring in communication time will greatly affect the results of reconstruct-before-search as the files are required to be downloaded to local storage. The addition of communication time will result in a slight increment to the results of direct search since the encoded keywords are needed to be sent to multiple storages. Finally, the addition of communication time in our proposed technique would result in an even smaller increment than direct search since the encoded keywords are sent to only one storage, where the search vector is stored.

B. Information Retrieval Analysis

The information retrieval analysis focuses on two metrics, precision, and recall. Precision of the searching method refers to the correctness of the results. The higher the precision, the fewer are the incorrect results that have been retrieved, i.e., lesser number of documents returned that do not contain the keyword. Recall is important when looking at the total amount of correct results that should be retrieved from the searching method. Higher recall means that the searching method can retrieve a larger number of documents that contain the search keyword. On the other hand, lower recall means a high number documents that contain the keyword have been missed.

Fig.8 shows precision and recall of direct search compared with our proposed technique. The horizontal axis shows the size of data stored in the system in MBs. The vertical axis shows the precision and recall values. The solid and dashed lines represent direct search and our proposed technique respectively. The circle and triangle data points represent precision and recall in both search methods. Overall, our proposed technique shows much better precision than direct search. Even though precision in both methods decreases gradually with size, our Search Vector *SV* based technique has a lower decreasing rate than direct search. In the case of recall all tests using the *SV* returned the value 1 . This means, *SV* technique is able to retrieve all possible documents that contain the keyword. The non-perfect precision of *SV* technique could be due to Galois Field operations that create collisions in *SV* generation. However, this effect does not affect the recall values of our proposed technique. On the other hand, both the precision and recall of direct search are generally very low due to the effects of data loss mentioned in section IV-B. We do not show the precision and recall values of reconstruction before search in Fig.8 as the values would always be 1. Table II summarizes the search results.

TABLE II
SEARCH RESULTS IN INFORMATION RETRIEVAL CONTEXT

Scheme	TP	FP	FN	Precision	Recall
Reconstruction	4927	0	0	1	1
Direct Search	4012	10330	915	0.28	0.81
SV	4927	906	0	0.84	1

C. Security Analysis

As described in [11], given a secure Vandermonde matrix is used for generation, erasure-coded data is secure as long as the adversary cannot obtain more than m parts. Therefore, in this security analysis we focus our attention on the security of the Search Vector. Search Vector *SV* is created using the original data. We assume that as described in Fig. 6 the *SV* is stored in a separate storage. We further assume that an adversary is able to get hold of the *SV* and its goal is to find as much information about the original data as possible. We should therefore analyse the possibility of information leakage through the Search Vector.

We measure the information leakage by *Mutual Information* between the original message and *SV*. Shannon [19], describes mutual information as the information shared between two sources. It can also be interpreted as the amount of information

in one source (message) that can be obtained from another source (Search Vector). A high value of mutual information means a high possibility of information leakage. As mutual information can be derived from the entropy of a message, i.e., an uncertainty of a message, the following equation describes calculation of message entropy.

$$H(M) = -\sum_{x \in M} p(x)\ log\ p(x) \tag{5}$$

where:

M = Message, the original text
$H(M)$ = Entropy of the message M
x = Byte that appears in the message M
$p(x)$ = Probability that x will appear in M

Then, the mutual information between two sources can be calculated from entropy of each source, and the conditional entropy between them. In our context, *M* refers to the original data, and SV refers to the Search Vector.

$$I(M; SV) = H(M)\ -\ H(M|SV) \tag{6}$$

$$I(M; SV) = \sum_{m,sv} p(m, sv)\ log\ \frac{p(m, sv)}{p(m)p(sv)} \tag{7}$$

where:

SV = Cipher text, our Searching Vector
$I(M; C)$ = Mutual information between M and SV
$H(M|SV)$ = Conditional entropy of M given SV
m, sv = Byte that appears in M and SV respectively
$p(m, sv)$ = Probability of m appears with sv
$p(m)$ = Probability of m appears
$p(sv)$ = Probability of sv appears

From eq. (6), the lower bound and upper bound of mutual information can be described as in eq. (8).

$$0 \leqslant I(M; SV) \leqslant H(M) \tag{8}$$

The lower bound refers to zero mutual information, that is the case when the two message sources are independent. In this case $H(M|SV)$ in eq. (6) is equal to the entropy of the message, $H(M)$ i.e., the uncertainty of the message does not reduce given the SV. On the contrary, the upper bound occurs when $H(M|SV)$ in eq. (6) is equal to zero, i.e., given SV, the uncertainty of message reduces to zero (absolute certainty). The upper bound also means that Search Vector is fully dependent on the message.

The total entropy of our dataset mentioned earlier in section IV was calculated to be 39.1837 bits. Recall that the experiments used *3-of-4* erasure coding scheme. The mutual information between the original text and the SV comes out to be 4.2223 bits. Therefore, with a known SV (3-of-4 erasure coding which is used in all experiments), the uncertainty of the original data, $H(M|SV)$, is reduced to 34.9615 bits according to eq. (6)

TABLE III
THE SMALLEST AND LARGEST MUTUAL INFORMATION WITH GENERATION PARAMETERS

	Value	I_{NORM}	*m-of-n Erasure Coding*
Smallest Mutual Information	3.5333	0.0902	*8-of-10*
Largest Mutual Information	4.4162	0.1127	*2-of-3*
Average Mutual Information	3.9863	0.1017	-

We further computed the mutual information of SV that is generated from 9 schemes of erasure coding, *2-of-3*, *3-of-4*, *4-of-5*, *4-of-6*, *6-of-7*, *6-of-8*, *6-of-9*, *8-of-10*, and *10-of-12*. The average mutual information between the original data using the above mentioned schemes and SV is 3.9863 bits. This is the average information leakage from the data without any knowledge about the parameters *m* and *n*. To compare the correlation of SV with the original data, normalised mutual information is calculated as shown below.

$$I_{NORM}(M; SV) = \frac{I(M; SV)}{H(M)} \tag{9}$$

Table III shows the smallest, largest and average mutual information between *SV* and original data along with the erasure coding scheme used to obtain that. The smallest mutual information, indicating the least leakage of information from *SV*, is calculated from *8-of-10* erasure coding. On the other hand, the highest information leakage from *SV* takes place when using *2-of-3* erasure coding. The table also shows the normalised mutual information for each case. We can see from the normalised information leakage that the *SV* and the original data has very small correlation. Therefore, we can conclude that our proposed scheme is secure and does not provide an attacker much information about the data even when the attacker obtains the *SV*.

We further explored the impact of the *m* and *n* parameters of the erasure coding scheme on information leakage. To do this we computed the correlation between the parameters and the mutual information between original data and *SV*.

$$Correlation_{x\ MI} = \frac{\sum(x_i - \overline{x})(MI_i - \overline{MI})}{\sqrt{\sum(x_i - \overline{x})^2(MI_i - \overline{MI})^2}} \tag{10}$$

where:

x_i = i-th erasure coding scheme's parameter
$\overline{x}$ = Average of the erasure coding scheme parameters
MI_i = Mutual information between original data and SV
$\overline{MI}$ = Average mutual information

We used eq. (10) to calculate the correlation for m and n and found that both the parameters have high negative correlation with mutual information. Correlation between m and MI is -0.9265 and correlation between n and MI is -0.9122. This indicates that as the values of m and n increase, the mutual information between the SV and original data decreases. It follows that, to achieve lower information leakage, one needs to use large values of m and n for the erasure coding scheme.

VII. Conclusion and Future Work

Erasure coding, in particular the systematic type, has been a popular method for increasing availability of data. Non-systematic erasure coding schemes on the other hand provide in built confidentiality of data but have not enjoyed that popularity, in part due to the lack of an efficient mechanism to search over erasure coded data. In this paper we presented a novel technique to search over non systematic erasure codes. Our technique relies on a data structure that we call the *Search Vector* which is built using the generator matrix. We evaluated the technique in terms of the latency it introduces and its precision and recall. The results show that our technique is only slightly worse than direct search in terms of speed and is highly reliable in terms of precision and recall. We also analysed the security of the technique in terms of the information leakage from the *Search Vector*. The analysis shows that while it is possible that a small amount of information leaks through the *Search Vector*, the leakage is small and can be improved by fine tuning the erasure code parameters, m, and n.

In future this work can be expanded by conducting a more thorough evaluation of the latency caused by this scheme when data is stored on the clouds. It will also be interesting to perform information retrieval analysis to evaluate the relationship between the number of files, file size, and accuracy and recall. Finally, it will also be of interest to investigate information leakage, if any, through user issued queries.

References

[1] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 328–337.

[2] "Introducing ibm spectrum scale erasure code edition," Apr 2021. [Online]. Available: https://www.ibm.com/docs/en/spectrum-scale-ece/5.1.0?topic=introduction-spectrum-scale-erasure-code-edition

[3] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer *et al.*, "Oceanstore: An architecture for global-scale persistent storage," *ACM SIGOPS Operating Systems Review*, vol. 34, no. 5, pp. 190–201, 2000.

[4] "Product documentation for red hat ceph storage 3." [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/3/

[5] MinIO, "Minio erasure code quickstart guide." [Online]. Available: https://docs.min.io/docs/minio-erasure-code-quickstart-guide.html

[6] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Presented as part of the 2012 {USENIX} Annual Technical Conference ({USENIX}{ATC} 12)*, 2012, pp. 15–26.

[7] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "Depsky: dependable and secure storage in a cloud-of-clouds," *Acm transactions on storage (tos)*, vol. 9, no. 4, pp. 1–33, 2013.

[8] T. Miyamae, T. Nakao, and K. Shiozawa, "Erasure code with shingled local parity groups for efficient recovery from multiple disk failures," in *10th Workshop on Hot Topics in System Dependability (HotDep 14)*, 2014.

[9] X. Zhang, Y. Cai, Y. Liu, Z. Xu, and X. Dong, "Nade: nodes performance awareness and accurate distance evaluation for degraded read in heterogeneous distributed erasure code-based storage," *The Journal of Supercomputing*, pp. 1–30, 2019.

[10] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. ACM*, vol. 36, no. 2, p. 335348, Apr. 1989. [Online]. Available: https://doi.org/10.1145/62044.62050

[11] M. Li, "On the confidentiality of information dispersal algorithms and their erasure codes," *arXiv preprint arXiv:1206.4123*, 2012.

[12] J. A. Garay, R. Gennaro, C. Jutla, and T. Rabin, "Secure distributed storage and retrieval," *Theoretical Computer Science*, vol. 243, no. 1, pp. 363–389, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397598002631

[13] P. Morales-Ferreira, M. Santiago-Duran, C. Gaytan-Diaz, J. Gonzalez-Compean, V. J. Sosa-Sosa, and I. Lopez-Arevalo, "A data distribution service for cloud and containerized storage based on information dispersal," in *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, 2018, pp. 86–95.

[14] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*. IEEE, 2000, pp. 44–55.

[15] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 965–976.

[16] P. Van Liesdonk, S. Sedghi, J. Doumen, P. Hartel, and W. Jonker, "Computationally efficient searchable symmetric encryption," in *Workshop on Secure Data Management*. Springer, 2010, pp. 87–100.

[17] G. S. Poh, J.-J. Chin, W.-C. Yau, K.-K. R. Choo, and M. S. Mohamad, "Searchable symmetric encryption: designs and challenges," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, pp. 1–37, 2017.

[18] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, 2011, pp. 142–150.

[19] C. E. Shannon *et al.*, "Coding theorems for a discrete source with a fidelity criterion," *IRE Nat. Conv. Rec*, vol. 4, no. 142-163, p. 1, 1959.