IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 13, 2020, accepted March 29, 2020, date of publication April 15, 2020, date of current version May 5, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2987999

Investigation of Optimal Data Encoding Parameters Based on User Preference for Cloud Storage

VIKAS CHOUHAN, (Member, IEEE), AND SATEESH K. PEDDOJU, (Senior Member, IEEE)
Department of Computer Science and Engineering, Indian Institute of Technology, Roorkee 247667, India

Corresponding author: Vikas Chouhan (vchouhan@cs.iitr.ac.in)

The work of Vikas Chouhan was supported in part by the Visvesvaraya Research fellowship from the Information Technology Research Academy (ITRA), Ministry of Electronics & Information Technology (MeitY), Government of India for his Ph.D. degree that includes the stipend and the support for his travel to attend research meetings or workshops, and other contingency expenses for a period of five years, and in part by the Indian Institute of Technology Roorkee, India, under Grant MIT-1100-CSE.

ABSTRACT The erasure encoding scheme creates multiple coded data and parity fragments to protect the data from the losses. Nowadays, most storage systems like cloud storage utilize the erasure coding scheme to attain superior data consistency, reliability, and availability. Most of the existing literature focuses on either the cost of recovery or overhead due to the redundant storage without considering the interests of the users, such as high reliability and lower storage cost. We believe that the storage service provider should choose an appropriate encoding scheme with optimal values of two encoding parameters, i.e., data fragments and parity fragments. The values of these encoding parameters depend on the size of the input data and the Quality of Service (*QoS*) requirements of the users, such as storage efficiency, availability, and recoverability. These parameters play a crucial role in providing higher reliability and lower storage costs. Therefore, in this paper, we investigate to identify optimal parameters to provide higher reliability and lower storage cost while considering the user's preferences. We present the analysis of the Reed-Solomon coding scheme from the perspective of storage overhead, the probability of data availability, data recoverability, and storage efficiency to identify the optimal values of encoding parameters. We performed the experiments on the Reed-Solomon encoding schemes, and results are reported.

INDEX TERMS Cloud computing, cloud storage, availability, storage cost, Reed-Solomon, erasure coding, reliability.

I. INTRODUCTION

Cloud storage service providers such as Dropbox, Microsoft, Google, and Amazon allow storing of the massive amount of data for both individual and enterprise customers in their datacenters [1], [2]. However, the storage systems may often experience the unavailability of customers data due to hardware and software failures. The erasure encoding scheme [3] creates multiple coded fragments of data and parity to protect the data from such losses. Nowadays, many Cloud Service Providers (CSP) employ Erasure Coding (EC) to reduce unexpected faults and minimize the data unavailability [4], [5]. However, the EC encoding scheme makes an encoding decision based on the static threshold value. The usage of such a static threshold-based encoding may suffer from storage overheads and fragility in a dynamic demand scenario.

The main focus of the work in this paper is to study how effectively to provide the services benefiting both CSP and customers. We believe that any cloud system is cost-effective and productive if CSP deploys optimal encoding parameters (as suggested in our approach). To maximize their service performance and availability, minimize the impact of service failures, and enhance the business continuity, the reliability and performance are two crucial issues in cloud storage that, in turn, will influence the service quality [6]. In case CSP tweaks the parameters, the storage systems may experience the unavailability of customer's data mostly during peak times, and may lead to loss of customers, which may, subsequently, impact the business metrics [6], [7]. In the absence of the optimal strategy, the total operational and storage costs may relatively increase for providing the services to the users.

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

And, it is more profitable if CSP adopts our strategy. For the expansion and leveraging of their business and improving users' trust, CSP should promise to incorporate the optimal strategy in the storage service, which helps in cost reduction.

A. MOTIVATION

CSPs store the user data in their datacenters. Since these datacenters are located geographically across the world, users often may encounter data unavailability. Moreover, the CSPs need to select the appropriate encoding parameters for different sized data to fulfill the dynamic demands of the users and to optimize the overall system costs while providing high reliability. The selection of suitable parameters is crucial, and the criteria for selecting these parameters depend on the size of input data and other Quality-of-Service (*QoS*) requirements such as storage efficiency, availability, recoverability. Keeping these points in mind, in this paper, we aim to select the optimal parameters to achieve data availability and recoverability.

B. CONTRIBUTIONS

The main contributions of this research are highlighted below:

- We propose an algorithm for investigating the optimal encoding parameters that meet the users' expectations.
- We explore the QoS requirements and summarize the findings according to the user's requirements and file size.
- We perform extensive experiments in real-time and discuss the performance analysis of the operations.

C. ORGANIZATION

The rest of the paper is organized as follows. The notations and function definitions are defined in Section II. Section III introduces erasure coding techniques and summarizes the related work. Section IV discusses the aspects of selecting the optimal encoding parameters for erasure coding. The proposed methodology is presented in the Section V. The results and analysis with experimental setup are presented in section VI. Finally, we conclude the paper in section VII.

II. PRELIMINARIES

In this section, Table 1 summarizes the notations used throughout the paper and then describes the set of function definitions that are used in our proposed work.

A. FUNCTION DEFINITIONS

- $\kappa \longleftarrow getFileCategory(F)$: This function takes the file F as input. It computes the size of F, and then determines and returns the file category (κ) based on its size, i.e, small, medium, and large.
- $\delta_{(\alpha,\beta)} \longleftarrow getEncodingPairs(\rho, \kappa)$: This function takes user preferences and file category as an input, and then determines and returns the optimal encoding pair(s) from the set of suitable encoding pairs (refer to Table 7).
- $(\alpha, \beta) \longleftarrow selectOptimalEncodingPair(\delta_{(\alpha,\beta)}, \kappa)$: This function determines and returns the optimal encoding pair from the set of suitable encoding pairs by calling Algorithm 1.
- $\delta_\gamma \longleftarrow executeErasureCoding(\alpha, \beta)$: This function calls the EC encoder with input α and β values to generate and return the set of fragments δ which contains $\gamma(=\alpha+\beta)$ number of encoded fragments.
- $(\delta_\alpha, \delta_\beta) \longleftarrow getSeparateFragments(\delta_\gamma)$: This function splits the input set δ_γ into two parts say δ_α and δ_β where δ_α is the set of α number of data fragments and δ_β is the set of β number of parity fragments.
- $count \longleftarrow getPairsCount(\delta_{(\alpha,\beta)})$: This function counts and returns the number of pairs existing in the set of encoding pairs.
- $(\alpha, \beta) \longleftarrow getPair(i, \delta_{(\alpha,\beta)})$: This function returns the i^{th} pair of the set $\delta_{(\alpha,\beta)}$.
- $(\alpha, \beta) \longleftarrow getAveragePair(\delta_{(\alpha,\beta)})$: This function computes the average number of $\gamma(=\alpha+\beta)$ values in the pairs from the set $\delta_{(\alpha,\beta)}$ and then selects (conflicts may be resolved on FCFS basis) and returns an encoding pair.
- $(\alpha, \beta) \longleftarrow getLargePair(\delta_{(\alpha,\beta)})$: This function computes the maximum number of $\gamma(=\alpha+\beta)$ values in the pairs from the set $\delta_{(\alpha,\beta)}$ and then selects (conflicts may be resolved on FCFS basis) and returns an encoding pair.

TABLE 1. Notations.

Symbol	Description
α	Number of Data Fragments
β	Number of Parity Fragments
γ	Total Number of Fragments
P_{avail}	Probability of Data Availability
η	Storage Efficiency
r	Encoding Rate
S_{oh}	Space Overhead
ψ	Single Data Fragment Size in Bytes
ρ	User Preference
F	Input File
κ	File Category
(α, β)	Encoding Pair
$\delta_{(\alpha,\beta)}$	Set of Encoding Pairs
δ_α	Set of Data Fragments
δ_β	Set of Parity Fragments
δ_γ	Set of Data and Parity Fragments
N	Total Number of Servers
M	Number of Currently Unavailable Servers

III. BACKGROUND AND RELATED WORK

In this section, first, we discuss the Erasure Coding (EC) scheme and its variants, and then we review the related works based on EC mechanisms.

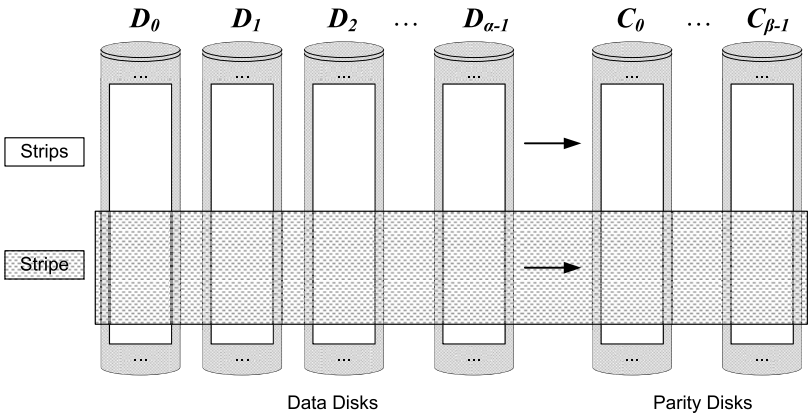

FIGURE 1. A typical storage system with EC.

A. ERASURE CODING

EC technique generates the input data D into α number of data fragments $(D_0, D_1, D_2, \ldots, D_{\alpha-1})$ and β number of parity fragments $(C_0, C_1, C_2, \ldots, C_{\beta-1})$ with a total of γ $(= \alpha + \beta)$ fragments. Maximum Distance Separable (MDS) [8], [9] property of the EC provides error recovery and data reconstruction in spite of any β number of unavailable fragments [10]. The recovery process uses the parity fragments to reconstruct the corrupted/unavailable fragments using rest of the data and parity fragments. We use the phrase *fragment unavailable* to indicate data loss, failure, corruption, or storage server unreachable.

The following section presents the widely used *Reed − Solomon* [11] and *Cauchy* Reed-Solomon [12] erasure codes that are used in this work for comparison and analysis of encoding pairs.

B. REED-SOLOMON (RS) CODES

Consider a typical storage system as shown in Figure 1 in which the symbol D_i denotes the data disks where $0 \leq i < \alpha$ and C_j denotes the parity disks where $0 \leq j < \beta$. It must satisfy the condition $\gamma \leq 2^w + 1$ [11] where every strip is a *w-bit* word having $w \in \{8, 16, 32, 64\}$. Each word is a number between 0 *and* $2^w - 1$. It applies $\gamma \times \alpha$ generator matrix which operates in a Galois Field $GF(2^w)$ to perform several operations like addition, subtraction, multiplication, and division on these words [9].

A Vandermonde matrix is used to construct the Generator Matrix (GM). It computes a codeword by multiplying GM with the α data words and β coding words as in Figure 2.

The recovery process performs the inverse operation and multiplication operation which are used for solving the set of independent linear equations. The addition operations in $GF(2^w)$ are done by performing bitwise *XOR*, but the multiplication operations are bit complex and expensive [9], [11].

C. CAUCHY REED-SOLOMON (CRS) CODES

CRS [12] replaces the Vandermonde matrices by the Cauchy matrices and also reduces the expensive multiplications

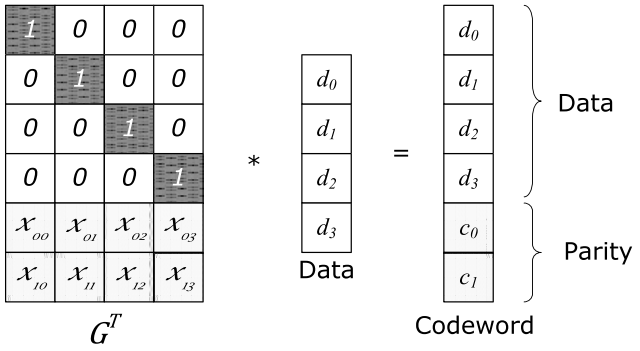

FIGURE 2. RS coding for $\alpha = 4$ and $\beta = 2$ [13].

operation of RS codes. CRS uses an additional *XOR* operations to minimize the number of multiplications of RS codes. This alteration converts the Generator matrix G^T from a $\gamma \times \alpha$ matrix of w-bit words to a $w\gamma \times w\alpha$ matrix of bits. CRS coding performs multiplication operation on all the strips rather than a single w-bit data words.

Each strip contains w fragments where w should satisfy the constraint $\gamma \leq 2^w$. The fragment size must be a multiple of the machine word size to achieve high performance. This encoding process involves only *XOR* operations. Therefore, the coding fragment is constructed as the *XOR* of all data fragments that have one bit in the coding fragment row of G^T. The process is depicted in Figure 3, which illustrates how the last coding fragment is created as the *XOR* of all data fragments identified by the last row of G^T [12].

D. RELATED WORK

Several techniques are proposed in the literature for efficient cloud storage. A three-way replication technique [2], [10] supports easy access and repairability in storage, but it increases storage overhead cost. It requires two-thirds of the raw storage capacity to store redundant data. Whereas, Erasure Coding technique minimizes the Mean Time To Data Loss (MTTDL) and redundancies in storage. See [14], [15], and [16] for more understanding.

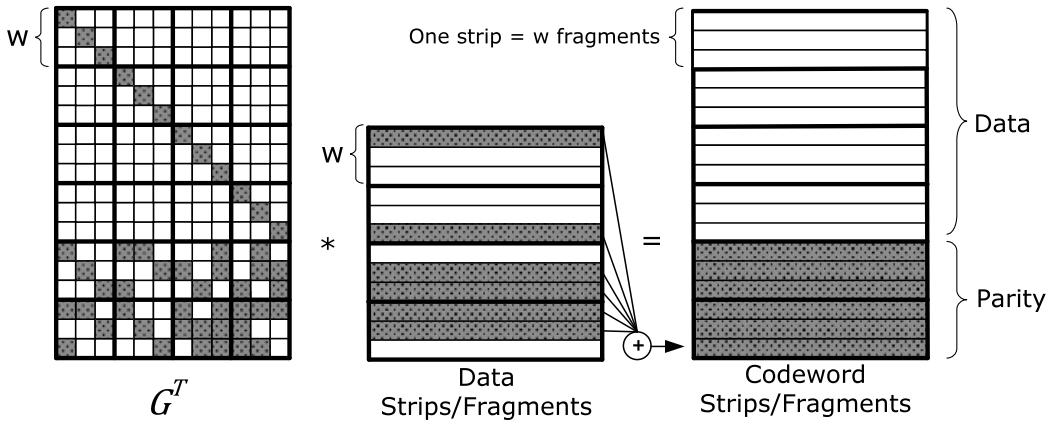

FIGURE 3. CRS coding example for $\alpha = 4$ and $\beta = 2$ [13].

Many authors proposed several EC schemes (e.g., [11], [12], [17], [18]) to achieve the fault tolerance and reliability in the storage systems. Schnjakin *et al.* [13] compared several EC schemes while providing their supporting libraries such as [19]–[21]. They determined an appropriate encoding algorithm for the Cloud-RAID system. Schuman and Plank [22] presented the encoding and decoding operations performance of a few erasure codes. While several authors evaluated the performance based on bandwidth and memory operations, others targeted to reduce the number of accesses to reconstruct the original data. Few others focused on some aspects such as read performance (e.g., [14], [23], [24]), coding performance (e.g., [22], [25]), fault tolerance (e.g., [26], [27]), proof of retrievability (e.g., [28]–[30]), and reliability (e.g., [31], [32]). Table 2 presents the summary of various works that employs erasure coding in their research for different purposes.

The reliability performance of erasure codes and its variant for several storage systems is studied in [14], [15], [33]–[37]. The reliability of erasure codes is evaluated in [16], [34], [38] based on Markov model. The estimation of Mean Time To Data Loss (MTTDL) for reliability analysis is studied in [16]. However, Greenan *et al.* [39] scrutinized the feasibility issues with the modeling of the modern storage systems for reliability analysis using the MTTDL and Markov model. Based on their study, they introduced the NOrmalized Magnitude of Data Loss (NOMDL) metric for reliability analysis. This metric is based on the failure and repair characteristics of the servers.

The work proposed in this paper, to achieve reliability, we rely on the data partitioning and fragment placement according to the encoding parameters of erasure codes. The partitioning of the input data is performed in order to minimize the impact of data loss. Each fragment is placed on a distinct server to maximize the tolerance against server failures.

In summary, we reviewed the most relevant literature related to erasure coding. However, most of the existing works have focused on the repair bandwidth, read latency, and storage overheads. But, our work focused on investigating optimal encoding parameters of erasure codes depending on various aspects like parameter selection and user preferences. To the best of our knowledge, the existing literature has not explored towards the identification of appropriate encoding parameters for efficient cloud storage. Therefore, our goal in this paper is to analyze and pick the optimal encoding parameters of erasure codes.

IV. ASPECTS OF PARAMETERS SELECTION

The optimal parameter value selection plays a crucial role to attain better storage efficiency, reliability, and availability in addition to the reduction in storage cost. In this section, we discuss the criteria to select the optimal encoding parameters for erasure coding.

The basic aspects of parameter selection to provide the quality of service to the users expectations depend on the number of fragments created, the size of each fragment, and the file size. Hence, we incorporate, primarily, the input file size and user preference to investigate the optimal value of the encoding parameters to achieve the *QoS* requirements. We classify the input files into three different categories based on their sizes such as *small*(256 *KB*), *medium*(512 *KB or* 1024 *KB*), and *large*(256 *MB or* 512 *MB*). Typically, the size or the number of categories can be variant.

A. USER PREFERENCES

The users specify their preferences in terms of availability of the resources, efficiency of the performance, the recoverability of the data, and acceptable overhead of storage. These preferences, predominantly, influence the cost of the storage for the users. So, the users notify the current demand of these preferences as HIGH, AVERAGE, or LOW. For example, if the user's preference for availability is HIGH, then probably, it will influence the storage cost. If the user's preference for the efficiency is HIGH, then probably, it will influence the storage overhead. So, we experimented with different possible cases for the user preferences with the combinations

TABLE 2. Summary of existing work that employs erasure coding.

Main Study	Reference	Approach/Methodology
Coding Performance	Schuman *et al.* [22]	Analysed the encoding/decoding performance for $(6, 2)$, $(12, 4)$, $(14, 2)$, & $(10, 6)$ pairs. Also examined the effect of packet size on encoding rate.
	Plank *et al.* [25]	Discussed the read latency for $(10, 6)$ pair values and showed the encoding/decoding performance for $(6, 2)$, $(14, 2)$, & $(12, 4)$ pair values.
Fault Tolerance and Reliability	Hafner *et al.* [31]	Discussed the storage reliability by applying variations on standard reliability model. Also, compared the LDPC code and Weaver codes to the MDS model.
	Woitaszek *et al.* [26]	Introduced a fault-tolerant storage system based on Massive Arrays of Idle Disks. Employed Tornado Codes is employing to achieve fault tolerance.
	Wylie *et al.* [27]	Introduced a MEL metric for XOR-based erasure codes to determine fault tolerance.
	Greenan *et al.* [32]	Introduced redundancy placement algorithms to maximize the reliability using XOR-based erasure codes.
	Park *et al.* [40]	Introduced a version of LDPC that reduces the number of repair blocks and storage overhead. The reliability is examined using the MTTDL metric.
Read Performance and Latency	Huang *et al.* [14]	Focused on reduction in bandwidth and input/outputs requirements, and extended the reliable storage with a lesser amount of overhead and low read latencies by using LRC Codes.
	Liang *et al.* [41] and Liang *et al.* [42]	Minimized the expected delay for static strategies using parallel connections and fixed FEC codes by adjusting the number of fragments according to workload level.
	Rashmi *et al.* [23]	Reduced the amount of data read during reconstruction using piggybacked RS codes.
	Rashmi *et al.* [43]	Introduced erasure codes that optimized the amount of disk I/O, storage, and bandwidth during the reconstruction phase by transforming certain classes of MSR.
	Li *et al.* [24]	Introduced the Beehive codes that can trim down network transfers and disk input/outputs significantly compared to the existing erasure codes.
Proof of Retrievability	Shacham *et al.* [28]	Addressed the problem of remote data integrity verification with proof of security against arbitrary adversaries using BLS signatures and pseudorandom functions. Used erasure codes to split the files into blocks.
	Juels et al. [44] and Bowers et al. [29]	Used Erasure coding and spot-checking mechanism to ensure data possession and retrievability on the external storage system.
	Wang *et al.* [45]	Introduced an integrity auditing mechanism for distributed storage using homomorphic token and erasure coding to ensures data correctness and error localization.
	Xu *et al.* [30]	Introduced an efficient and secure proof of retrievability mechanism that supports remote data auditing. Used Erasure coding to encode the files.
	Henry *et al.* [46]	Introduced a data integrity protection scheme to address the problem of remote data integrity verification in cloud storage. Used FMSR codes to protect the storage against data corruption and demonstrate trade between performance and security for a few encoding parameters.
	Ren *et al.* [47]	Proposed a dynamic proof of retrievability scheme to allow public auditability and communication efficient recovery from data corruption. Adapted network coding and erasure codes to encode data blocks in order to tolerate data corruption and supporting communication efficient data recovery.

Note: Abreviations are define as follows: BLS: Boneh–Lynn–Shacham, FEC: Forward Error Correction, FMSR:Functional Minimum Storage Regenerating, LDPC:Low-density Parity-check, LRC: Local ReconstructionCodes, MDS: Maximum Distance Separable, MEL: Minimal Erasures List, MSR: Minimum storage regenerating, MTTDL: Mean Time To Data Loss, PoR: Proofs of Retrievability, RS: Reed–Solomon.

of the current demand including HIGH, AVERAGE, and LOW.

To investigate the effectiveness of the encoding parameters and to satisfy the current demand of the users, we study the performance of EC in Section VI based on the decisive factors discussed in the following section.

B. DECISIVE FACTORS

(i) **Data Availability**

The probability of data availability (P_{avail}) [10] can be computed using Equation 1.

$$P_{avail} = \sum_{i=0}^{\gamma-\alpha} \frac{\binom{M}{i}\binom{N-M}{\gamma-i}}{\binom{N}{\gamma}}. \quad (1)$$

In this equation, $\binom{M}{i}$ denotes the number of ways in which we can organize inaccessible fragments on unavailable servers; $\binom{N-M}{\gamma-i}$ is the number of ways in which we can organize accessible fragments on available servers, and $\binom{N}{\gamma}$ is the total number of ways in which we can organize the γ fragments on total servers. Generally CSPs store two absolute copies that can provide availability of data with the probability 0.99 [10]. Assume that $N = 10^5$, and $M = 4500$, applying EC with $\gamma = 24$ fragments and $r = 1/2$ computes the probability of availability,P_{avail}, to be 1. In this paper, we analyzed the availability for various (α, β) pairs to maximize availability.

(ii) **Rate of Encoding**

The data encoding rate (r) [10] can be calculated as the number of data fragments (α) divided by the total erasure code fragments (γ), where $\gamma > \alpha$.

$$r = \frac{\alpha}{\gamma}. \quad (2)$$

The number of redundant fragments β represent the fault tolerance capability of the encoding parameters. Moreover, this encoding rate increases the storage cost by a factor of $f = \frac{1}{r}$, called as boost factor.

(iii) **Storage Efficiency**

The storage efficiency (η) can be calculated as the ratio of the number of data fragments (α) and the total erasure code fragments (γ). In this paper, we evaluated the efficiency of various (α, β) pairs to analyze the performance of EC.

$$\eta = \frac{\alpha}{\gamma} \times 100\% = r \times 100\%. \quad (3)$$

(iv) **Recoverability**

The optimal number of redundant fragments is another important parameter that provides recoverability. The EC encoding generates γ fragments and can recover actual data from any α fragments. Therefore, it can tolerate up to the $\beta = \gamma - \alpha$ number of unavailable fragments. We used the recoverability metric to maximize data reliability and minimize the impact of data loss.

(v) **Storage Overhead**

The EC encoding includes certain fragments overhead for the data recovery purpose. This overhead is caused by the padding of β fragments. The estimated storage overhead is computed by the ratio of the number of redundant fragments and the number of data fragments.

$$Estimated\ Storage\ Overhead = \frac{\beta}{\alpha} \times 100\%. \quad (4)$$

However, the erasure coding generates γ fragments of the same size corresponding to the given data size. Hence, the resulting fragment size is the multiplication of one fragment size and γ. Practically, using these fragments size and data size, we can compute the space overhead which is computed as:

$$S_{oh} = (\frac{\psi \times \gamma}{DataSize} - 1) \times 100\%. \quad (5)$$

In our experiments, we used Equation 5 to measure the storage overhead.

V. PROPOSED METHODOLOGY

This section presents the investigation procedure of the optimal encoding parameters, followed by the fragments generation procedure.

In this approach, a user is expected to provide the input file along with the preferences to the CSP for the selection of optimal encoding parameters and generation of coded fragments, which is shown in Figure 4. The following steps show the corresponding sequence of operations.

1) The CSP, first, sends the file as an input to both the investigation process module and EC block. The *investigation process* module, initially, calculates the size of the input file, and then it uses the preferences to decide the encoding (α, β) pair.
2) The EC block gets the input optimal pair values from the investigation process to encode the input file.
3) The EC scheme encodes the input file based on the selected encoding (α, β) pairs to generates the α number of data and β number of parity fragments.

The selection of optimal encoding parameters and the generation of fragments are divided into two phases, as discussed below.

A. PHASE I: INVESTIGATION PROCEDURE

In this phase, an optimal encoding pair is selected among the set of suitable pairs which is shown in Algorithm 1.

Algorithm 1 takes the set of encoding pairs $\delta_{(\alpha,\beta)}$ and file category κ as an input, and it returns the optimal encoding pair (α, β). The algorithm performs the following operations:

i. Initially, it calls the function *getPairsCount*, which takes $\delta_{(\alpha,\beta)}$ as an input and returns the number of pairs exists in the input set.
ii. In case of only a single pair exist in the set $\delta_{(\alpha,\beta)}$ or file belongs to the small size category then it returns first pair of the input set by calling $getPair(1, \delta_{(\alpha,\beta)})$ function.

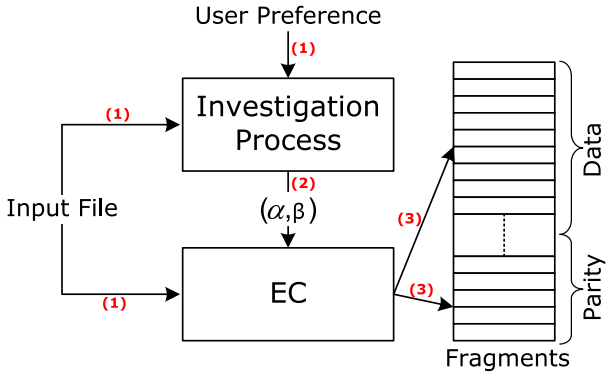

FIGURE 4. Selection of optimal encoding parameters and generation of coded fragments where numbers in red represents the sequence of operations.

Algorithm 1 Selection of Optimal Encoding Pair

Input : *Set of Encoding Pairs* $\delta_{(\alpha,\beta)}$, *File Category* κ
Output: *Encoding Pair* (α, β)

```
1  begin
2      count = 0
3      Let count denotes the number of elements in the Set δ(α,β)
4      count ⟵ getPairsCount(δ(α,β))
       /* Returns number of pairs in the input
          set                                          */
5      if (count == 1 || κ == "small") then
6          return getPair(1, δ(α,β))
           /* Returns 1st pair of the Set δ(α,β)
              */
7      end
8      else if (count == 2) then
9          return getPair(2, δ(α,β))
           /* Returns 2nd pair of the Set δ(α,β)
              */
10     end
11     else if (count == 3) then
12         if (κ == "medium") then
13             return getPair(2, δ(α,β))
               /* Returns 2nd pair of the Set
                  δ(α,β)                               */
14         end
15         else
16             return getPair(3, δ(α,β))
               /* Returns 3rd pair of the Set
                  δ(α,β)                               */
17         end
18     end
19     else if (count > 3) then
20         if (C == "medium") then
21             return getAveragePair(δ(α,β))
22         end
23         else
24             return getLargePair(δ(α,β))
25         end
26     end
27 end
```

iii. In case of only two pairs exist in the set $\delta_{(\alpha,\beta)}$ then it returns second pair of the input set by calling *getPair*$(2, \delta_{(\alpha,\beta)})$ function.

Algorithm 2 Generation of Data and Parity Fragments

Input : *Preference* ρ, *File* F
Output: *Set of Data and Parity Fragments* $(\delta_\alpha, \delta_\beta)$

```
1  begin
2      Let κ denotes the category of the file i.e. small,
       medium, and large
3      κ ⟵ getFileCategory(F) /* Determines and
          returns the file category                    */
4      δ(α,β) ⟵ getEncodingPairs(ρ, κ) /* returns the
          set of encoding pairs                        */
5      (α, β) ⟵ selectOptimalEncodingPair(δ(α,β), κ)
          /* Call Algorithm 1                          */
6      δγ ⟵ executeErasureCoding(α, β) /* returns
          the set of γ fragments; where
          γ = α + β                                    */
7      (δα, δβ) ⟵ getSeparateFragments(δγ)
8      return (δα, δβ)
9  end
```

iv. In case of only three pairs exist in the set $\delta_{(\alpha,\beta)}$ and file belong to the medium size category then it returns second pair of the input set by calling *getPair*$(2, \delta_{(\alpha,\beta)})$ function, otherwise (file belongs to large size category) it returns the third pair of the input set by calling *getPair*$(3, \delta_{(\alpha,\beta)})$ function.

v. In case of a set $\delta_{(\alpha,\beta)}$ contains more than three pairs and file belong to the medium size category then it returns average pair from the input set by calling *getAveragePair*$(\delta_{(\alpha,\beta)})$ function, otherwise (file belongs to large size category) it returns the large pair of the input set by calling *getLargePair*$(\delta_{(\alpha,\beta)})$ function.

B. PHASE II: FRAGMENT GENERATION PROCEDURE

Phase II executes Algorithm 2 which generates the coded fragments based on user's preference and input file.

Algorithm 2 takes the user's preference ρ and file F as an input, and returns the pair which contains a set of data and parity fragments. The algorithm performs the following operations:

i. It calls the function *getFileCategory* with an input file F to get the file category κ.
ii. Then it calls the function *getEncodingPairs* function with preference ρ and file category κ as an input to get the set of encoding pairs $\delta_{(\alpha,\beta)}$.
iii. Then it selects the optimal encoding pair $(\alpha, \beta) \in \delta_{(\alpha,\beta)}$ by calling *selectOptimalEncodingPair* function.
iv. Then it calls the function *executeErasureCoding* with input α and β values to generates the set of γ fragments.
v. Now, the *getSeparateFragments* function is executed which separates the set of data and parity fragments $(\delta_\alpha, \delta_\beta)$ from the set δ_γ.
vi. Finally, the algorithm returns the pair $(\delta_\alpha, \delta_\beta)$ containing the set of data and parity fragments.

TABLE 3. Probabilities of data availability at $M = 10\%$.

Encoding	P_{avail} at $N = 10^5$	P_{avail} at $N = 10^6$	P_{avail} at $N = 10^7$	$Avg\ P_{avail}$
EC(3,2)	0.991441944	0.991441944	0.991440194	0.991441361
EC(5,3)	0.994977763	0.994977763	0.994975861	0.994977129
EC(6,4)	0.998366216	0.998366216	0.998365178	0.99836587
EC(8,3)	0.981471552	0.981471552	0.98146587	0.981469658
EC(8,4)	0.995673403	0.995673403	0.995670931	0.995672579
EC(10,3)	0.965849252	0.965849252	0.965840276	0.96584626
EC(10,4)	0.990775023	0.990775023	0.990770311	0.990773452
EC(12,6)	0.998829374	0.998829374	0.998828022	0.998828923
EC(12,8)	0.999940286	0.999940286	0.999940156	0.999940243
EC(12,10)	0.99999755	0.99999755	0.999997541	0.999997547
EC(12,12)	0.999999915	0.999999915	0.999999914	0.999999915

VI. RESULTS AND ANALYSIS

In this section, first, we discuss the experimental setup with packages and libraries, followed by several performance analysis and results.

A. EXPERIMENTAL SETUP

The experimental setup comprises of a desktop machine with Intel® *Core*^{TM} i7-3770 CPU @ 3.40GHz processor of 8 cores, 8 GB RAM, installed with Ubuntu 14.04 64-bit operating system and *Python*2.7 − *dev* package, and other connected python libraries. We have employed the Erasure Coding module using *PyECLib* − 1.2.0 library with the *liberasurecode* − *dev* and *libjerasure* − *dev* packages.

B. DATASETS

We construct testing datasets for analyzing the cost of the operations. These datasets contain multiple files with a set of distinct file flavors. A typical file sets are considered having different sizes, 256*KB*, 512*KB*, 1024*KB*, 256*MB*, and 512*MB*. Further, we determine the total of 66 pair values that satisfy the conditions: $1 < \alpha \leq 12$ and $1 < \beta \leq \alpha$. For practical consideration, we conduct the tests on the erasure coding with a set of following selected (α, β) pair values: {(3, 2), (5, 3), (6, 4), (8, 3), (8, 4), (10, 3), (10, 4), (12, 6), (12, 8), (12, 10), (12, 12)}. These specific pairs are selected to analyze the Reed-Solomon codes which are arranged in chronological order according to the number of data fragments. In our experiment, we exploit *Reed* − *Solomon* schemes such as RS and CRS, and further present the comparison between them. However, RS scheme use the *liberasurecode* − *dev* package and Jerasure RS & CRS use the *libjerasure* − *dev* package for evaluation of the results.

C. DATA AVAILABILITY ANALYSIS

The data availability computation is shown in Equation 1 and the corresponding results are presented in the Tables 3 and 4.

We analyze the data availability for various input (α, β) pair values with $N = \sum_{i=2}^{9} 10^i$ number of total existing servers and $M\%$ of currently unavailable servers where $M = \{10,20,30,40,50\}$. Table 3 demonstrates the data availability for different values of $N = \{10^5, 10^6, 10^7\}$ and the average availability at $M = 10\%$. Since the ranges are overlapping, we have taken the average of N values to see how they get impacted due to M with {10, 20, 30, 40, 50}, which is summarized in Table 4. We choose a large number of unavailable servers to determine availability. The probability of data availability for $N = 10^5, 10^6$, and 10^7 is nearly the same. Hence, we deduce that the data availability is unrestrained from the total number of existing servers, and further, it achieves the equilibrium state. The relatively better availability is achieved at $M = 10\%$ which lies between 0.9658393 *and* 0.9999999.

Figure 5 summarizes the data availability probability of erasure encoding parameters at $M = 20\%$ corresponding to the total number of servers, and the results display the higher availability at $EC(12, 12)$ & $EC(12, 10)$.

D. RELIABILITY ANALYSIS

The important consideration in cloud storage are the data recoverability metric, minimizing the impact of data loss, and server failures. The increase in the number of redundant fragments increases the recoverability of the corrupted fragments to build a durable system. Each fragment can be placed on a distinct server to maximize the tolerance against server failures. To recover the corrupted fragments, it requires to retrieve α number of fragments from the storage. To capture the probability of data loss and normalized magnitude of data loss, we use a discrete-event simulator SimECD [48] with its default configuration of 4TiB disk capacity per node. The simulation is based on the generation of failure and repair events in a production datacenter. It measures the probability of data loss (PDL) based on the number of permanently lost

TABLE 4. Average probabilities of data availability at $M = \{10\%, 20\%, 30\%, 40\%, 50\%\}$.

Encoding	P_{avail} at $M = 10\%$	P_{avail} at $M = 20\%$	P_{avail} at $M = 30\%$	P_{avail} at $M = 40\%$	P_{avail} at $M = 50\%$
EC(3,2)	0.991441361	0.942083226	0.836923704	0.682562419	0.5
EC(5,3)	0.994977129	0.943723538	0.805901368	0.594088026	0.363277422
EC(6,4)	0.99836587	0.967211589	0.849739473	0.633106769	0.376948818
EC(8,3)	0.981469658	0.838868552	0.569562339	0.296275985	0.113272226
EC(8,4)	0.995672579	0.927452872	0.723661942	0.438174249	0.193837504
EC(10,3)	0.96584626	0.747330759	0.420599102	0.168568077	0.046135246
EC(10,4)	0.990773452	0.870170908	0.584202331	0.27924614	0.089772022
EC(12,6)	0.998828923	0.948740146	0.721706513	0.37426759	0.118927388
EC(12,8)	0.999940243	0.9900231	0.886684325	0.595603758	0.251707197
EC(12,10)	0.999997547	0.998411184	0.961266945	0.771966538	0.415899476
EC(12,12)	0.999999915	0.999783478	0.988503737	0.885754276	0.580596899

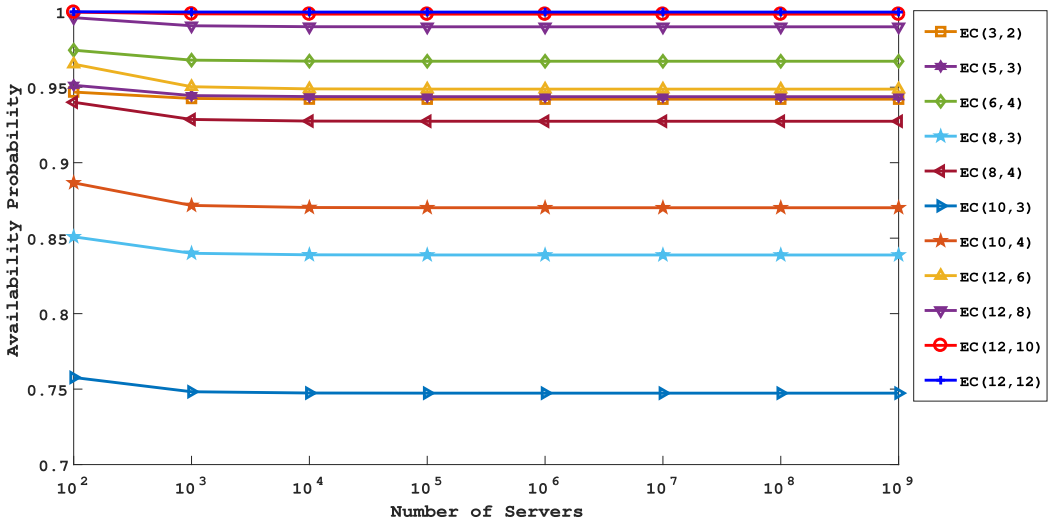

FIGURE 5. Availability probability of erasure encoding parameters at $M = 20\%$.

fragments over an expedition period (10 years in our case). The lowest PDL represents more tolerance capacity against data corruption. It also uses Normalized the Magnitude of Data Loss (NOMDL) by measuring the expected amount of data loss normalized to the storage capacity. Figure 6 shows the graph of reliability metric for various α and β value pairs of RS codes. Since NOMDL values in the graph are very small relative to PDL values, we scale the NOMDL by 10^4 in the graph.

The redundant fragment increases the capacity of data recovery and decreases the impact of data loss or server failures. The pairs, i.e., (3, 2), (8, 3), and (10, 3) has limited number of redundant fragments. That may lead to high chances of inaccessibility due to failures. As well, (8, 3) and (10, 3) has the lowest probability of data availability, according to Table 4. As can be seen in Figure 6, the pairs (3, 2), (8, 3), and (10, 3) has relatively higher PDL than others. According to Figure 6 and the number of parity fragments, we observe that the pairs $RS(12, 12)$ & $RS(12, 10)$ has lowest PDL, NOMDL and highest recoverability among all pairs, so that these pairs offer better data reliability and high accessibility.

E. ENCODING EFFICIENCY AND STORAGE COST ANALYSIS

We plot the estimated storage efficiency and overheads for various input (α, β) pair values as shown in Figure 7. According to the Figure 6 and Figure 7, we say that the $EC(12, 12)$ achieves high reliability and $EC(10, 3)$ provides the high efficiency and low storage overhead.

F. SPACE OVERHEAD ANALYSIS

We analyze the space overhead of the RS and CRS schemes based on various input (α, β) values. We perform experiments on various file samples of different sizes.

TABLE 5. Execution time (in μs) of encoding operation for various input file samples.

File Size	Encoding	EC(3,2)	EC(5,3)	EC(6,4)	EC(8,3)	EC(8,4)	EC(10,3)	EC(10,4)
256KB	RS	0.000618625	0.00092001	0.001655364	0.000631142	0.00088861	0.000734711	0.001140428
	JRS	0.000218558	0.00038228	0.000547814	0.000410056	0.00057323	0.000304675	0.000365877
	CRS	0.000294709	0.00039835	0.000526285	0.000363278	0.000479364	0.000380564	0.000439334
512KB	RS	0.000764203	0.001098275	0.002268505	0.001317883	0.002169204	0.00130105	0.002039313
	JRS	0.000323796	0.000824356	0.000780511	0.000544667	0.000748587	0.000589418	0.000776958
	CRS	0.000576115	0.000710917	0.000800323	0.000531721	0.000691319	0.000519681	0.000805807
1024KB	RS	0.001792097	0.002600622	0.004019737	0.002826858	0.004017878	0.002647471	0.003940034
	JRS	0.000635409	0.001713705	0.001589227	0.001290488	0.00162518	0.001227736	0.001866364
	CRS	0.000592828	0.000844622	0.001381993	0.001063609	0.001016593	0.001023817	0.001202488
256MB	RS	0.647053719	0.838044906	1.217429709	0.88321588	1.193140149	0.87514658	1.174454641
	JRS	0.453764892	0.523606157	0.665286803	0.519930339	0.67131319	0.520850301	0.666269755
	CRS	0.363608599	0.441983724	0.539226699	0.429107428	0.527471685	0.447753906	0.524421239
512MB	RS	1.291857266	1.68828423	2.416339636	1.766760206	2.389678979	1.741206002	2.357754946
	JRS	0.785832167	1.042118382	1.32170186	1.025963497	1.309180379	1.03348577	1.317791724
	CRS	0.709969497	0.891177607	1.073768687	0.846559644	1.034380603	0.837551093	1.029846501

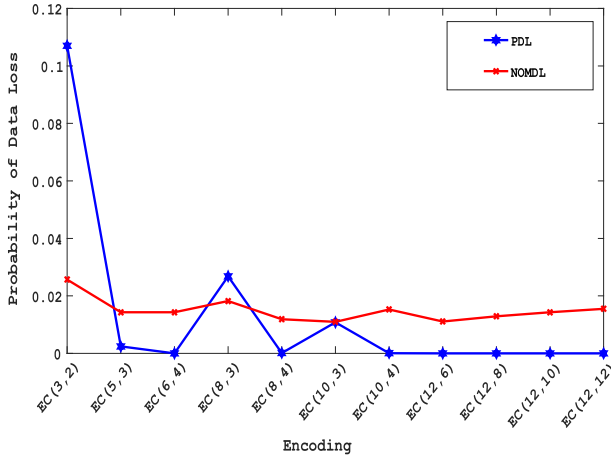

FIGURE 6. Reliability metric for various α & β value pairs of RS codes.

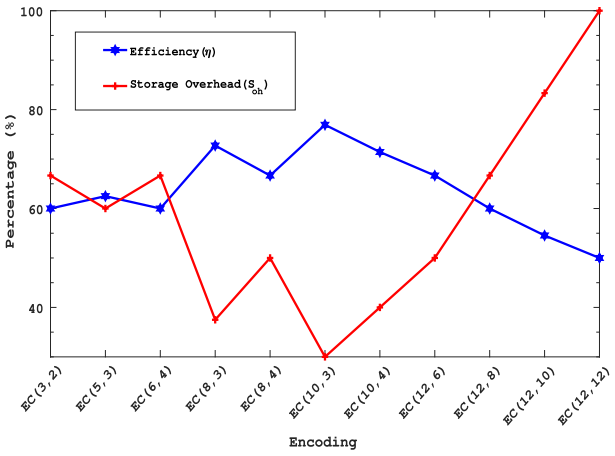

FIGURE 7. Estimated storage efficiency and overheads.

The Figure 8(a) and Figure 8(b) show the space overhead for the input files of size 256KB and 512MB respectively. The CRS scheme is relatively expensive as compared to the RS encoding for the smaller file as shown in Figure 8(a). The space overheads of both the schemes are nearly the same for the larger file, however, little higher overheads for the smaller file. The reason lies in the fact that the encoding library includes 80$byte$ header to each fragment.

Figure 9 represents the overall space overhead based on the implementation results for various input (α, β) values to RS encoding. The results show that $RS(10, 3)$ and $RS(12, 12)$ has the lowest and the highest space overhead respectively.

G. EXECUTION COST ANALYSIS

In this subsection, we analyze the execution cost for the sample files of size 256KB, 512KB, 1024KB, 256MB, and 512MB using Python *timeit* module. We perform each encoding operation 1000 times to evaluate the average execution time which is summarized in Table 5.

We observe from the Table 5 that all the operations require extremely less time, between 0.00021 to 2.41μs, for executing the encoding operation of all the sample files. Further, we analyze the execution cost of encoding operations for the sample files and illustrate the behavior of the small and large file in Figure 10(a) and Figure 10(b) respectively. We observe

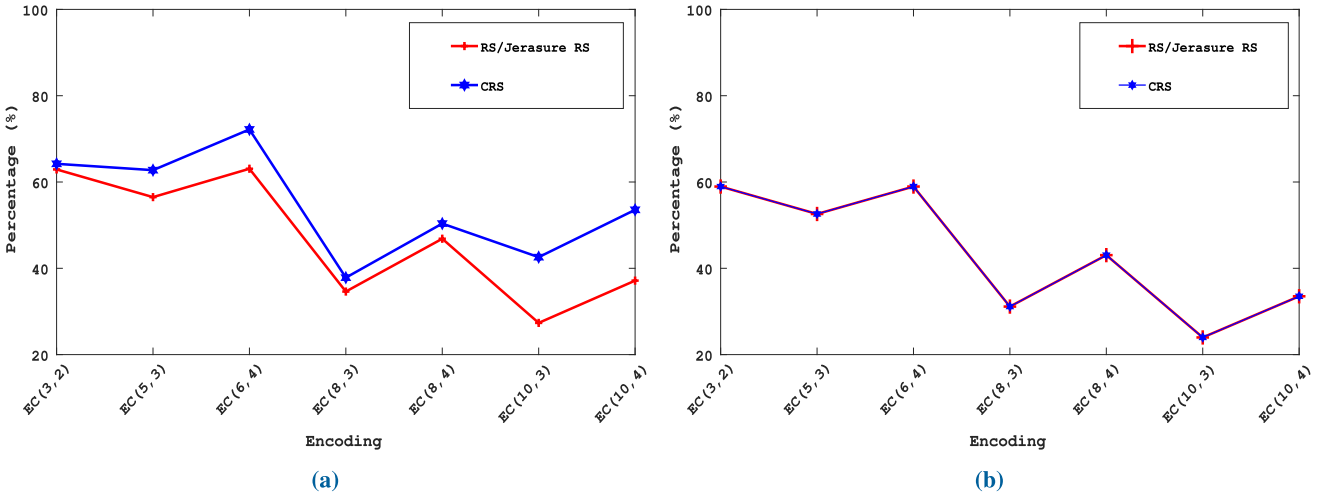

FIGURE 8. (a) Space overheads for the input file 256KB. (b) Space overheads for the input file 512MB.

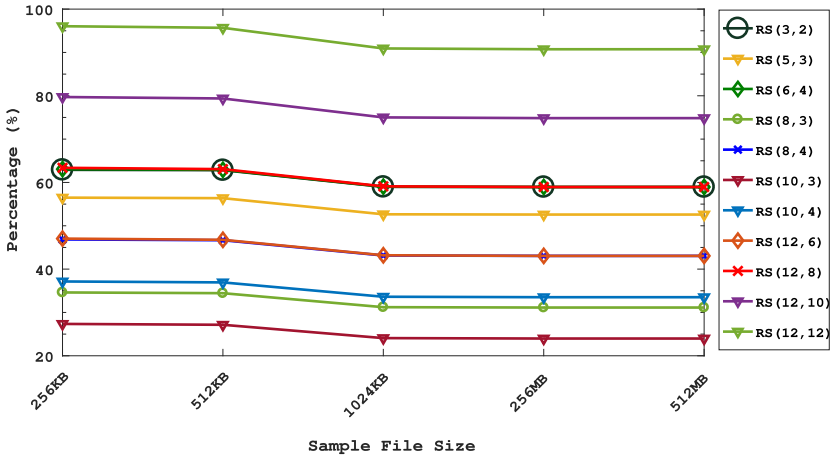

FIGURE 9. Space overheads for various input file samples.

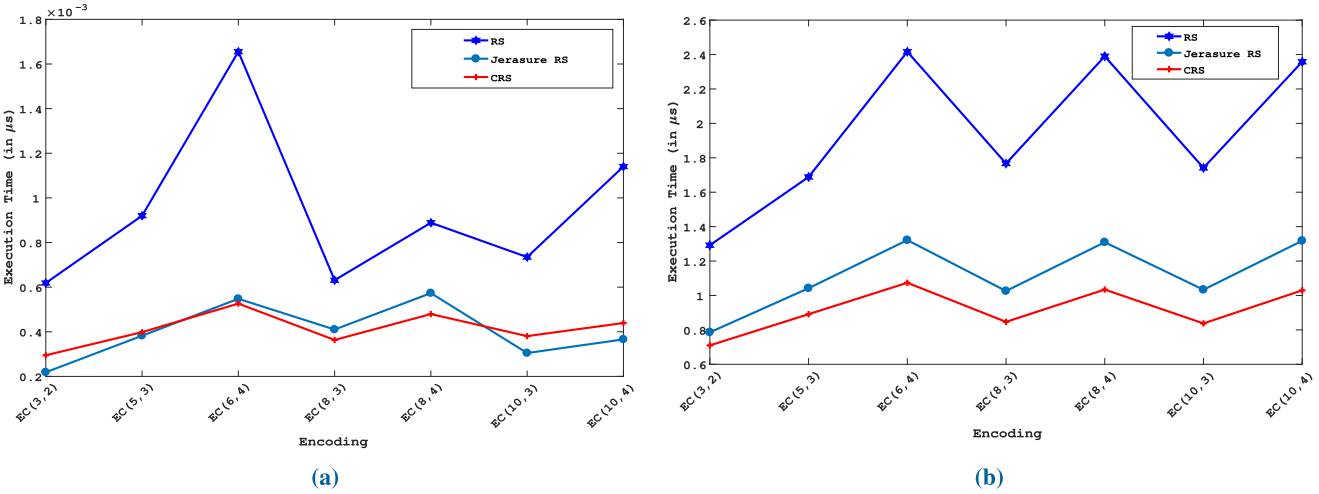

FIGURE 10. (a) Execution time of encoding for the input file 256KB. (b) Execution time of encoding for the input file 512MB.

TABLE 6. Performance analysis of *EC* for various input (α, β) pair values.

Encoding	Availability	Recoverability	Efficiency	Storage Overhead
EC(3,2)	High	Average	Average	Average
EC(5,3)	High	Average	Average	Average
EC(6,4)	High	Average	Average	Average
EC(8,3)	Average	Low	High	Low
EC(8,4)	High	Average	High	Average
EC(10,3)	Low	Low	High	Low
EC(10,4)	Average	Low	High	Low
EC(12,6)	High	Average	High	Average
EC(12,8)	High	Average	Average	Average
EC(12,10)	High	High	Average	High
EC(12,12)	High	High	Average	High

TABLE 7. Suitable encoding pairs based on input file size and user preference.

Input File Size	User Preference	Encoding Pair Values
Small	Preference-I	$(3, 2)$
	Preference-II	$(8, 3)$
	Preference-III	$(8, 4)$
	Preference-IV	$(12, 10)$
Medium	Preference-I	$(5, 3)$ or $(6, 4)$
	Preference-II	$(8, 3)$
	Preference-III	$(8, 4)$
	Preference-IV	$(12, 10)$
Large	Preference-I	$(12, 8)$
	Preference-II	$(10, 4)$
	Preference-III	$(12, 6)$
	Preference-IV	$(12, 12)$

that the RS scheme takes more computation time compared to the other schemes, and the Jerasure RS and CRS schemes require almost equal computation time. Hence we conclude that the Jerasure RS and CRS encoding scheme is computationally better than the RS schemes.

The EC encoding operations performance for various input (α, β) pair values on the basis of availability, recoverability, efficiency, and storage overhead is summarized in Table 6. We perform the availability analysis at $M = 20\%$ and we assume the availability to be *Low* at 0.7, *average* at 0.8, and *high* at 0.9. We observe the data recoverability based on the percentage of additional fragments. We assume the recoverability as *low* if *percentage* $<$ 50, *average* if 50 $\leq$ *percentage* $<$ 80, and *high* if 80 $<=$ *percentage*. Similarly, we assume the efficiency to be *low* if *efficiency* $<$ 50, *average* if 50 $\leq$*efficiency*$<$ 65, and *high* if 65 $<=$ *efficiency*. Further, we assume storage overhead as *low* if *overhead* $<$ 50, *average* if 50 $\leq$ *overhead* $<$ 70, and *high* if 70 $<=$ *overhead*.

We achieve high availability and high efficiency at $EC(8, 4)$ and $EC(12, 6)$ with average recoverability and storage overhead. Further, we achieve high efficiency and lower storage overhead at $EC(8, 3)$ and $EC(10, 4)$ with average availability. Moreover, the $EC(12, 8)$ achieve high availability with average efficiency, recoverability, and storage overhead. Hence, either of the pair, $(8, 4)$ or $(12, 6)$, is the suitable choice for designing the highly available, efficient, and fault-tolerant storage system with average storage overhead. Based on the aforementioned findings, the input preference, as classified below, define the user requirements for the encoding of the file.

- *Preference-I*: High availability with average efficiency, recoverability, and storage overhead.
- *Preference-II*: High efficiency and lower storage overhead with an average availability.
- *Preference-III*: High availability and high efficiency with average recoverability and storage overhead.
- *Preference-IV*: High availability and high recoverability with average efficiency.

We have applied the following rules based on Algorithm 1 to assign the Encoding pair values to different files depending upon the number of available encoding pair values:

- *Case-I:* If only one pair is available, then assign that pair to all the files irrespective of their size.
- *Case-II:* If two pairs are available, then assign the smaller pair value to small and medium size files; and higher pair value to large size file.

- *Case-III:* If three pairs are available, then assign the smaller pair value to small files; medium pair to medium size files and higher pair value to large size file.
- *Case-IV:* If more than three pairs are available, then assign the smaller pair value to small files; higher pair value to large files; and any one of the remaining pair value to medium size files depending upon the first come first serve basis.

Table 7 illustrates the above discussed rules to assign the encoding pair values. Suppose for given input file and user preference, we get the multiple (α, β) pair values as (3, 2), (5, 3), (6, 4), (12, 8) for encoding a particular file. Therefore, we select the minimum number of γ $(= \alpha + \beta)$ fragments from the offered pairs for small file, average number of γ fragments for medium file, and maximum number of γ fragments for large file. Thus, we choose the encoding pair values (3, 2), {(5, 3) or (6, 4)}, and (12, 8) for small, medium, and large files respectively.

VII. CONCLUSION

The CSPs provide Storage-as-a-Service to the end-users. To prevent data unavailability due to server failures and software or hardware faults, they use the EC concept. The encoding scheme provides a durable and recoverable system with high reliability and low storage cost based on the dynamic selection of encoding parameters as per the user preferences. Our findings provide direction towards the selection of optimal encoding pairs of erasure coding suitable to the specific user's requirements. The investigation revealed that the probability of availability achieved up to 0.9999999 when ten percent of the servers were unavailable. It is also observed that the encoding operations require extremely less time, between 0.00021 to 2.41μs, for all the sample files. On the other hand, we explored the *QoS* requirements and summarized the findings according to the user's requirements and file sizes. $RS(12, 12)$ & $RS(12, 10)$ provided high reliability and also offered better data recovery and high accessibility. However, the results showed that both $RS(12, 12)$ & $RS(12, 10)$ have the highest space overhead. Though, we achieved high availability and high efficiency at $EC(8, 4)$ and $EC(12, 6)$ with average recoverability and storage overhead. Further, we achieved high efficiency and low storage overhead at $EC(8, 3)$ and $EC(10, 4)$ with average availability. Also, $EC(12, 8)$ achieved high availability with average efficiency, recoverability, and storage overhead. Hence, either of the pairs, (8, 4) or (12, 6), is the suitable choice for designing the highly available, efficient, and fault-tolerant storage system with average storage overhead, on a specific user's specifications. These choices may change based on the user's preferences. The results indicate that utilizing the optimal selection of encoding values. We can significantly reduce the unnecessary space overhead and computation overhead, and fulfill the user *QoS* requirements while providing higher availability and reliability. Essentially, reducing the overall system cost.

REFERENCES

[1] H.-Y. Lin and W.-G. Tzeng, "A secure erasure code-based cloud storage system with secure data forwarding," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 6, pp. 995–1003, Jun. 2012, doi: 10.1109/TPDS.2011.252.

[2] P. Li, X. Jin, R. J. Stones, G. Wang, Z. Li, X. Liu, and M. Ren, "Parallelizing degraded read for erasure coded cloud storage systems using collective communications," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 1272–1279.

[3] N. Alon, J. Edmonds, and M. Luby, "Linear time erasure codes with nearly optimal recovery," in *Proc. IEEE 36th Annu. Found. Comput. Sci.*, Oct. 1995, pp. 512–519.

[4] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, and Y. Xu, "Windows azure storage: A highly available cloud storage service with strong consistency," in *Proc. 23rd ACM Symp. Operating Syst. Princ. (SOSP)*, New York, NY, USA, 2011, pp. 143–157, doi: 10.1145/2043556.2043571.

[5] D. Ford, F. Labelle, F. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proc. 9th USENIX Symp. Operating Syst. Design Implement.*, 2010, pp. 1–14.

[6] B. Yang, F. Tan, and Y.-S. Dai, "Performance evaluation of cloud service considering fault recovery," *J. Supercomput.*, vol. 65, no. 1, pp. 426–444, 2013.

[7] M. R. Mesbahi, A. M. Rahmani, and M. Hosseinzadeh, "Reliability and high availability in cloud computing environments: A reference roadmap," *Hum.-Centric Comput. Inf. Sci.*, vol. 8, no. 1, Dec. 2018.

[8] C. Suh and K. Ramchandran, "Exact-repair MDS codes for distributed storage using interference alignment," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2010, pp. 161–165.

[9] F. J. MacWilliams and N. J. A. Sloane, *The Theory Error-correcting Codes*, vol. 16. Amsterdam, The Netherlands: Elsevier, 1977.

[10] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Proc. 1st Int. Workshop Peer–Peer Syst.* London, U.K.: Springer-Verlag, 2002, pp. 328–338. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687814

[11] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, Jun. 1960.

[12] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," Int. Comput. Sci. Inst., Berkeley, CA, USA, Tech. Rep. TR-95-048, Oct. 1999.

[13] M. Schnjakin, T. Metzke, and C. Meinel, "Applying erasure codes for fault tolerance in cloud-RAID," in *Proc. IEEE 16th Int. Conf. Comput. Sci. Eng.*, Dec. 2013, pp. 66–75.

[14] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Proc. USENIX Annu. Tech. Conf.*, Boston, MA, USA, 2012, pp. 15–26.

[15] D. Ford, F. Labelle, F. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proc. 9th USENIX Conf. Oper. Syst. Design Implement.*, Vancouver, BC, Canada, 2010, pp. 61–74.

[16] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: Novel erasure codes for big data," *Proc. VLDB Endowment*, vol. 6, no. 5, pp. 325–336, Mar. 2013.

[17] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An optimal scheme for tolerating double disk failures in RAID architectures," in *Proc. 21 Int. Symp. Comput. Archit.*, 1994, pp. 245–254.

[18] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *Proc. 3rd USENIX Conf. File Storage Technol.*, Berkeley, CA, USA, 2004, p. 1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1973374.1973375

[19] A. Partow. *Schifra Reed–Solomon ECC Library*. Accessed: Apr. 13, 2018. [Online]. Available: http://www.schifra.com/downloads.html

[20] J. Plank, S. Simmerman, and C. Schuman, "Jerasure: A library in C/C++ facilitating erasure coding for storage applications-Version 1.2," Tech. Rep., Jan. 2008.

[21] Z. O'Whielacronx. *Efficient, Portable Erasure Coding Tool*. Accessed: Apr. 13, 2018. [Online]. Available: https://pypi.python.org/pypi/zfec

[22] C. D. Schuman and J. S. Plank, "A performance comparison of open-source erasure coding libraries for storage applications," Univ. Tennessee, Knoxville, TN, USA, Tech. Rep. UT-CS-08-625, 2008.

[23] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A 'hitchhiker's' guide to fast and efficient data reconstruction in erasure-coded data centers," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 331–342, Aug. 2014. [Online]. Available: http://doi.acm.org/10.1145/2740070.2626325
[24] J. Li and B. Li, "Beehive: Erasure codes for fixing multiple failures in distributed storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 5, pp. 1257–1270, May 2017.
[25] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn, "A performance evaluation and examination of open-source erasure coding libraries for storage," in *Proc. Fast*, vol. 9, Feb. 2009, pp. 253–265.
[26] M. Woitaszek and H. M. Tufo, "Tornado codes for MAID archival storage," in *Proc. 24th IEEE Conf. Mass Storage Syst. Technol. (MSST)*, Sep. 2007, pp. 221–226.
[27] J. J. Wylie and R. Swaminathan, "Determining fault tolerance of XOR-based erasure codes efficiently," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2007, pp. 206–215.
[28] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Berlin, Germany: Springer, 2008, pp. 90–107.
[29] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: Theory and implementation," in *Proc. ACM Workshop Cloud Comput. Secur. (CCSW)*, 2009, pp. 43–54.
[30] J. Xu and E.-C. Chang, "Towards efficient proofs of retrievability," in *Proc. 7th ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS)*, 2012, pp. 79–80.
[31] J. L. Hafner and K. Rao, "Notes on reliability models for non-MDS erasure codes," IBM, Armonk, NY, USA, Res. Rep. RJ10391, 2006.
[32] K. M. Greenan, E. L. Miller, and J. J. Wylie, "Reliability of flat XOR-based erasure codes on heterogeneous devices," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. With FTCS DCC (DSN)*, 2008, pp. 147–156.
[33] D. A. Patterson, G. Gibson, and R. H. Katz, *A Case for Redundant Arrays Inexpensive Disks (RAID)*, vol. 17, no. 3. New York, NY, USA: ACM, 1988.
[34] W. A. Burkhard and J. Menon, "Disk array storage system reliability," in *Proc. 23rd Int. Symp. Fault-Tolerant Comput.*, 1993, pp. 432–441.
[35] J. G. Elerath and M. Pecht, "Enhanced reliability modeling of RAID storage systems," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2007, pp. 175–184.
[36] K. Rao, J. L. Hafner, and R. A. Golding, "Reliability for networked storage nodes," in *Proc. Int. Conf. Dependable Syst. Netw. (DSN)*, May 2006, pp. 237–248.
[37] M. Baker, M. Shah, D. S. H. Rosenthal, M. Roussopoulos, P. Maniatis, T. Giuli, and P. Bungale, "A fresh look at the reliability of long-term digital storage," *ACM SIGOPS Operating Syst. Rev.*, vol. 40, no. 4, pp. 221–234, Oct. 2006.
[38] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence, "FAB: building distributed enterprise disk arrays from commodity components," *ACM SIGARCH Comput. Archit. News*, vol. 32, no. 5, pp. 48–58, 2004.
[39] K. M. Greenan, J. S. Plank, and J. J. Wylie, "Mean time to meaningless: MTTDL, Markov models, and storage system reliability," in *Proc. HotStorage*, 2010, pp. 1–5.
[40] H. Park, D. Lee, and J. Moon, "LDPC code design for distributed storage: Balancing repair bandwidth, reliability, and storage overhead," *IEEE Trans. Commun.*, vol. 66, no. 2, pp. 507–520, Feb. 2018.
[41] G. Liang and U. C. Kozat, "FAST CLOUD: Pushing the envelope on delay performance of cloud storage with coding," *IEEE/ACM Trans. Netw.*, vol. 22, no. 6, pp. 2012–2025, Dec. 2014.
[42] G. Liang and U. C. Kozat, "TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2014, pp. 826–834.
[43] K. Rashmi, P. Nakkiran, J. Wang, N. B. Shah, and K. Ramchandran, "Having your cake and eating it too: Jointly optimal erasure codes for i/o, storage, and network-bandwidth," in *Proc. 13th USENIX Conf. File Storage Technol.*, 2015, pp. 81–94.
[44] A. Juels and B. S. Kaliski, Jr., "PORs: Proofs of retrievability for large files," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 584–597.
[45] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward secure and dependable storage services in cloud computing," *IEEE Trans. Services Comput.*, vol. 5, no. 2, pp. 220–232, Apr. 2012, doi: 10.1109/TSC.2011.24.
[46] H. C. H. Chen and P. P. C. Lee, "Enabling data integrity protection in Regenerating-Coding-Based cloud storage: Theory and implementation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 407–416, Feb. 2014.
[47] Z. Ren, L. Wang, Q. Wang, and M. Xu, "Dynamic proofs of retrievability for coded cloud storage systems," *IEEE Trans. Services Comput.*, vol. 11, no. 4, pp. 685–698, Jul. 2018.
[48] M. Zhang, S. Han, and P. P. C. Lee, "SimEDC: A simulator for the reliability analysis of erasure-coded data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2836–2848, Dec. 2019.

VIKAS CHOUHAN (Member, IEEE) received the Master of Technology degree in computer science and engineering from the Indian Institute of Technology, Roorkee, India, where he is currently pursuing the Ph.D. degree in computer science and engineering. His research interests include cloud computing, cloud security and privacy, and data security. He served as a program committee member for many international conferences. He was one of the recipients of the Visvesvaraya Research Fellowship for the Ph.D. Program. He is the Publicity Co-Chair of the IEEE MASS, in 2020, SLICE, in 2019, and DSEA, in 2018.

SATEESH K. PEDDOJU (Senior Member, IEEE) is currently an Associate Professor with the Indian Institute of Technology, Roorkee, India. He is a coauthor of the book *Security and Storage Issues in the Cloud Environment* and co-editor of the book *Cloud Computing Systems and Applications in Healthcare*. He has publications with reputed journals, such as the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, the *IEEE Potentials*, *MTAP*, *WPC*, *IJIS*, and *PPNA*, and conferences, such as ACM MobiCom, the IEEE TrustCom, the IEEE MASS, the ACM/IEEE ICDCN, and ISC. His research interests include cloud computing, ubiquitous computing, and security. He is a Senior Member of ACM. He was a recipient of the Cloud Ambassador Award from AWS Educate, the IBM SUR Award, the Microsoft Educate Award, the University Merit Scholarship, the Best Teacher Award in his previous employment, the Best Paper/Presentation Awards, and the Travel Grants. He received grants from NMHS, MEITY, Railtel, MHRD, DST, IBM, Samsung, CSI, and Microsoft. He is the Founding General Co-Chair of SLICE, in 2018, and on board of the IEEE SLICE, the IEEE DSEA, the IEEE MASS, the IEEE ATC, the IEEE SmartComp, the IEEE iNIS, and the IoTSMS. He is serving as a Reviewer for the IEEE TRANSACTIONS ON CLOUD COMPUTING, the IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE ACCESS, *MTAP*, *COSE*, *COMNET*, *JNCA*, and *CLUS*.