Approximate Code: A Cost-Effective Erasure Coding Framework for Tiered Video Storage in Cloud Systems

Huayi Jin[1], Chentao Wu[1*], Xin Xie[1], Jie Li[1], Minyi Guo[1], Hao Lin[2], and Jianfeng Zhang[2]
[1]MoE Key Lab of Artificial Intelligence, AI Institute & Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
[2]The Alibaba Group, China
*Corresponding author: wuct@cs.sjtu.edu.cn

ABSTRACT

Nowadays massive video data are stored in cloud storage systems, which are generated by various applications such as autonomous driving, news media, security monitoring, etc. Meanwhile, erasure coding is a popular technique in cloud storage to provide both high reliability and low monetary cost, where triple disk failure tolerant arrays (3DFTs) is a typical choice. Therefore, how to minimize the storage cost of video data in 3DFTs is a challenge for cloud storage systems. Although there are several solutions like approximate storage technique, they cannot guarantee low storage cost and high data reliability concurrently.

To address this challenge, in this paper, we propose Approximate Code, which is an erasure coding framework for tiered video storage in cloud systems. The key idea of Approximate Code is distinguishing the important and unimportant data with different capabilities of fault tolerance. On one hand, for important data, Approximate Code provides triple parities to ensure high reliability. On the other hand, single/double parities are applied for unimportant data, which can save the storage cost and accelerate the recovery process. To demonstrate the effectiveness of Approximate Code, we conduct several experiments in Hadoop systems. The results show that, compared to traditional 3DFTs using various erasure codes such as RS, LRC, STAR and TIP-Code, Approximate Code reduces the number of parities by up to 55%, saves the storage cost by up to 20.8% and increase the recovery speed by up to 4.7X when double nodes fail.

CCS CONCEPTS

• **Information systems → Distributed storage**; • **Computer systems organization → Dependable and fault-tolerant systems and networks**.

KEYWORDS

Erasure Codes, Approximate Storage, Multimedia, Cloud Storage

ICPP 2019, August 5–8, 2019, Kyoto, Japan

ACM ISBN 978-1-4503-6295-5/19/08…$15.00
https://doi.org/10.1145/3337821.3337869

ACM Reference Format:
Huayi Jin, Chentao Wu, Xin Xie, Jie Li, Minyi, Hao Lin, and Jianfeng Zhang. 2019. Approximate Code: A Cost-Effective Erasure Coding Framework for Tiered Video Storage in Cloud Systems. In *48th International Conference on Parallel Processing (ICPP 2019), August 5–8, 2019, Kyoto, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3337821.3337869

1 INTRODUCTION

For typical cloud storage systems such as Windows Azure [6] and Amazon AWS [2], erasure coding is a popular technique to provide both high reliability and low monetary cost [3–5, 8, 9, 11, 40, 46], where triple disk failure fault tolerant arrays (3DFTs) are widely used. Typical erasure codes can be divided into two categories, RS-based Codes [30] [18] and XOR-based codes [3, 19, 43, 49]. RS-based codes are encoded based on the Galois Field(GF) computations [30], which allow flexible configurations. XOR-based codes simplify the computations compared to RS-based code, but they suffer other issues like scalability [21, 42].

With the increasing demand on higher resolution and frame rate for video applications such as autonomous driving, news media, security monitoring, etc., massive storage devices are highly desired in cloud storage systems, which makes data centers much bigger. Therefore, in this paper, we set out to answer the following question:

In a cloud storage system, how to efficiently store tremendous video data in 3DFTs?

To reduce the storage cost in cloud storage systems, a feasible solution is approximate storage. Approximate storage exposes unimportant data to errors, saving the overhead of redundant backups [26, 31] , thus the data reliability cannot be guaranteed. The second solution is using disk arrays like RAID-5[24] or RAID-6 [27], but the capability of fault tolerance is sacrificed. Data compression[1] is the third strategy to reduce the storage overhead [10, 51, 52], which needs to be collaborated with erasure codes for high reliability. However, the compression/decompression processes result in a large amount of computational overhead, which limits their usage in cloud systems. Therefore, existing solutions are difficult to provide low storage cost and high reliability simultaneously.

To address the challenge, in this paper, we propose Approximate Code, which is an erasure coding framework to provide a comprehensive solution for tiered video data storage in cloud systems. The key idea of Approximate Code is treating the important/unimportant data in different ways. For important data, we add additional parities to provide high capability of fault tolerance. On the other hand, the unimportant data are encoded with a minimum number of parities, which only supply the basic requirement

[1]Typically, video coding (e.g., H.264 [41]) is a special type of data compression.

Table 1: The symbols used in this paper.

Symbols	Description
k	the number of data nodes (in a local stripe)
r	the number of parity nodes (in a local stripe)
n	the number of all data and parity nodes in a local stripe. $n = k+r$
h	the number of local stripes to construct a global stripe
g	the number of global parity nodes
f	the number of failed nodes
p	a prime number
N	total number of all nodes ($N = h*(k+r)+g$)
$C_{i,j}$	the element at the i-th row and j-th column
$A_{i,j}$	the coefficient of C_{ij} in the Galois Field
D_i	a data node
P_I	the mathematical expectation of important data being recoverable under faults that exceed fault tolerance
P_U	the mathematical expectation of unimportant data being recoverable under faults that exceed fault tolerance
$\oplus$	an XOR operation
$\binom{n}{k}$	the number of k-combinations from n elements
LP	the local parity node
GP	the global parity node
ID	the important data in a video file
UD	the unimportant data in a video file

of the recovery. When triple disks fail, the lost unimportant data can be reconstructed via a fuzzy manner.

We have the following contributions of this work,

(1) We propose Approximate Code, which is a cost-effective framework to store video data in cloud storage systems.
(2) Approximate Code can be implemented by combining various erasure codes in 3DFTs, such as RS, LRC, STAR Code, TIP-Code, etc.
(3) We conduct several quantitative analysis and experiments according to different layouts of various erasure codes, and the results show that Approximate Code achieves a small number of parities, low storage cost, high single write performance, and fast data recovery under various failure scenarios.

The rest of the paper is organized as follows. In Section 2, we introduce related work and our motivation. In Section 3, the design of Approximate Code Framework will be illustrated in detail. The evaluation is presented in Section 4 and the conclusion of our work is in Section 5.

2 RELATED WORK AND OUR MOTIVATION

In this section, we introduce the background of video storage, existing solutions to reduce storage cost and the motivation of this paper. To facilitate the discussion, we summarize the symbols used in this paper in Table 1.

2.1 Basis of Video Storage

For normal high-definition (HD, resolution 1280×720, 8-bit, 30 fps) video, the amount of raw video data in 1 minute is 4.63 GB, so video data is usually encoded by lossy compression algorithms before storage.

2.1.1 Video coding. H.264 [41] is one of the advanced algorithms. This coding technique is widely used on platforms such as YouTube because it has higher compression ratio and lower complexity than its predecessors. For a normal HD video file as mentioned above, H.264 can reduce its size by about 10 times, only 443.27MB.

H.264 classifies all frames into three different categories,

(1) I frame: A frame that is self-contained, which means it can be decoded independently.
(2) P frame: A frame holds the changes compared to the previous frame, which saves the storage space by leaving out redundant information.
(3) B frame: A frame decreases the storage space as well by utilizing the data of both the preceding and following frames.

A group of pictures (GOP) consists of one I frame followed by a number of consecutive P and B frames, as shown in Figure 1. Within each GOP, the I frame is the most significant because all other frames rely on it for recovery. P frames are the second important, and B frames have the least value. Based on this feature, a special program can be designed to get the important video frames.

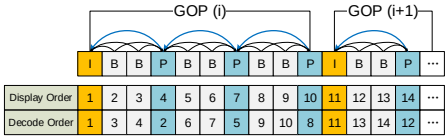


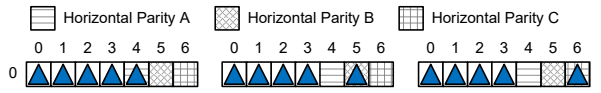
Figure 1: A sample of GOPs in H.264.

2.1.2 Video Frame Recovery. In the circumstance of video approximate storage, it's common to lose some frames and make the video incompleted. However, the lost frames can still be recoverable with the benefit of powerful deep learning techniques. One of them is named video frame interpolation [25, 26, 37].

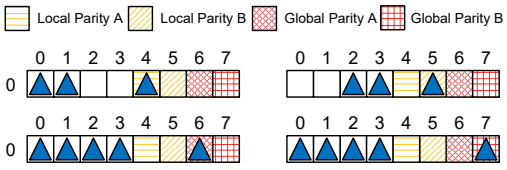
2.2 Existing Erasure Codes

Reliability is a critical issue since disk failures are typical in storage systems. To improve the reliability of storage systems, several RAID forms (e.g., RAID-5, RAID-6, 3DFTs) and erasure codes are proposed by researchers. Traditional erasure codes can be categorized into two classes, Maximum Distance Separable (MDS) codes and non-MDS codes. MDS codes aim to offer data protection with optimal storage efficiency. On the other hand, non-MDS codes improve the performance or reliability by consuming extra storage space.

In the past two decades, several famous erasure codes are proposed for double Disk Failure Tolerant arrays (2DFTs or RAID-6), such as EVENODD code [3], RDP code [8], Blaum-Roth code[4], X-code [46], Liberation code [29], Liber8tion code [28], Cyclic [7] code, B-Code [45], Code-M [38], H-code [43], P-code [22] and HVcode [33], etc. In Triple Disks Failure Tolerant Arrays (3DFTs), typical MDS codes include Reed-Solomon codes [30], Cauchy-RS codes [5], STAR code [19], Triple-Star code [40], Triple-Parity code [9], HDD1 code [35], RSL-code [11], and so on. Typical non-MDS codes contain WEAVER codes [15], HoVercodes [16], T-code [34], HDD2 code [35], Pyramid codes [17], Local Reconstruction Codes [18], Locally Repairable Codes [32], AZ-code [44], etc. Besides these codes, several methods are proposed to accelerate the encoding/decoding [48][12]. In the following we mainly introduce the classic erasure codes used in this paper.

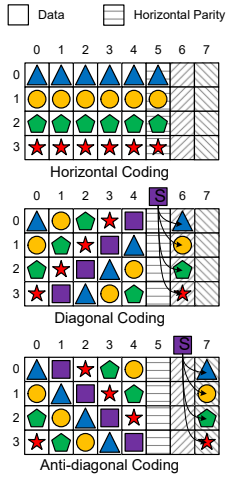


(a) **RS Code** for arbitrary storage nodes ($k = 4$, $r = 3$). The figure shows the encoding of different horizontal parities. (e.g., $C_{0,5} = \sum_{i=0}^{3} A_{0,i} C_{0,i}$)

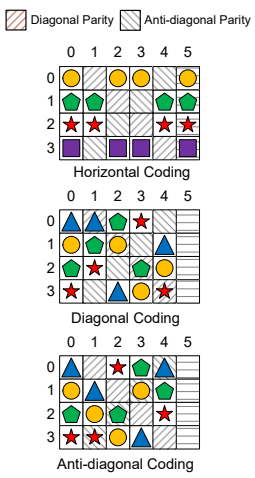


(b) **LRC Code** for arbitrary storage nodes ($k = 4$, $r = 2$, $g = 2$), where 4 nodes are divided into 2 local groups (node 0-1 and node 2-3) and each of them has a local parity. The figure shows the encoding of local and global parities. (e.g., the local parity $C_{0,5} = \sum_{i=2}^{3} A_{0,i} C_{0,i}$ and the global parity $C_{0,7} = \sum_{i=0}^{3} A_{0,i} C_{0,i}$)

Figure 2: Encoding of RS codes and LRC.



(a) **STAR Code:** For example, in the diagonal parity $S = C_{3,1} \oplus C_{2,2} \oplus C_{1,3} \oplus C_{0,4}$ and $C_{0,6} = C_{0,0} \oplus C_{3,2} \oplus C_{2,3} \oplus C_{1,4} \oplus S$.



(b) **TIP-Code:** For example, in the horizontal parity $C_{0,5} = C_{0,0} \oplus C_{0,2} \oplus C_{0,3}$

Figure 3: Encoding of STAR ($p = 5$) (Figure 3(a)) and TIP ($p = 3$) (Figure 3(b)) codes. Each column represents a node, each block represents an element, and parity chains are represented by elements with same shape. STAR requires the number of data nodes to be p, and TIP requires that to be $p - 2$.

Reed Solomon (RS) Code [30] is a classic MDS code, as shown in Figure 2(a). The encoding and decoding operations of RS code are based on Galois Field (GF), leading to high scalability as well as high computational complexity [13].

Local Construction Code (LRC) [18] is shown in Figure 2(b). It divides the data nodes into several groups, which are called local stripes. Typically, each local stripe contains one local parity node, which is calculated via the data nodes in the same stripe. Global stripe is generated via crossing multiple local stripes.

STAR code [19] is an extension of EVENODD [3] code, as shown in Figure 3(a). In the encoding process of STAR code, the generation of diagonal parity and anti-diagonal parity requires to calculate $S1$ and $S2$ first, which leads to a long parity chain.

TIP-Code[49] is also an XOR-based 3DFTs erasure code, as shown in Figure 3(b). Compared with STAR code, the parities are generated independently, which reduces the number of I/O operations for partial stripe writes.

2.3 Approximate Storage

Approximate storage is an innovative approach for storage devices [14, 20, 31, 50], which enables applications to store data approximately. It can improve the performance, lifetime, or reduce the power consumption of different storage devices.

Specifically, for video applications, there are several approximate storage methods [14, 20] for bit-level reliability in progressively encoded multimedia data. [14] identifies the relative importance of the encoded bits on the image quality and applys different levels of error correction. [20] shows how reliability of video storage can be traded for density and achieves the optimal quality/density points by tracking visual and metadata dependencies within the encoded bit stream.

2.4 Our Motivation

We summarize various methods to reduce storage cost for video files in Table 2, the device level scheme of existing 3DFTs has high storage overhead, while RAID-5 or RAID-6 maintains medium storage overhead but sacrifices fault tolerance. Although approximate storage achieves low cost, its reliability cannot satisfy cloud environment. Application layer methods such as compression algorithms have high computational overhead and need to be used together with erasure codes to ensure reliability. Therefore, the existing solutions cannot guarantee high reliability and low overhead at the same time of video storage.

From the summary of Table 2, existing solutions have their drawbacks on providing high reliability with low storage cost. A feasible solution is combining various erasure codes with tiered storage [23] [39] [47] [36], which motivates us to propose Approximate Code.

Table 2: Comparison of storage overhead, fault tolerance and computing overhead among various storage methods for video files.

Schemes	Layer	Storage Overhead	Fault Tolerance	Computing Overhead
Data Compression	Application Level	low	/	high
RS-based EC for 3DFTs	Device Level	high	high	high
XOR-based EC for 3DFTs		high	high	low
RAID-6		medium	medium	low
RAID-5		low	low	low
Approximate Storage		very low	very low	low
Approximate Code		low	high	low

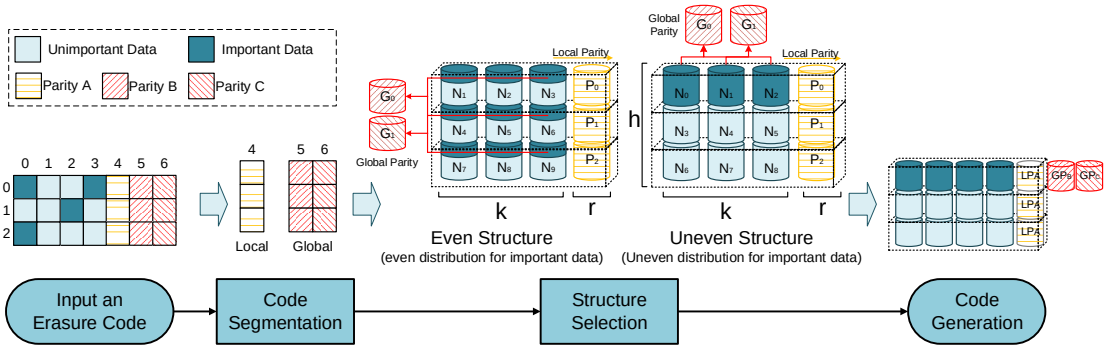


Figure 4: The Framework of Approximate Code.

3 APPROXIMATE CODE

In this section, we introduce the Approximate Code Framework and its properties through a few simple examples.

3.1 Overview of Approximate Code Framework

The Approximate Code Framework contains four main steps, code input, code segmentation, structure selection and code generation, as shown in Figure 4.

3.1.1 Code Input. Acquire an erasure coding input and get the corresponding parameters. Typically, for cloud storage systems, erasure codes used in 3DFTs are all accepted as the input. Several parameters related to erasure codes need to be collected, such as the number of data nodes k and the number of parity nodes r.

3.1.2 Code Segmentation. For an inputted erasure code, we assume that its fault tolerance is x. The Approximate Code first splits its parities into two parts, local and global parities. The local parities are used for all data, while the global parities focus on the important data. Code segmentation are designed to ensure that local parities can tolerate any r node failures, so the fault tolerance of unimportant data is r. For important data, the code segmentation ensures that the important data have the capabilities of fault tolerance by up to x nodes.

For example, several erasure codes in 3DFTs like STAR have three types of parities, horizontal, diagonal and anti-diagonal parities. In the code segmentation step, the horizontal parities are regarded as local parities, which are separated from the diagonal/anti-diagonal parities (shown in Fig. 6 and illustrated in Section 3.3 in detail).

3.1.3 Structure Selection. The framework chooses the structure after code segmentation. There are two main structures for distributing the important and unimportant data, as shown in Figure 4. In Even Structure, important data are distributed uniformly among all data nodes, while in Uneven Structure, they are stored among several nodes dedicatedly.

Since Even Structure distributes the important data across each node, it guarantees a balanced workload. Meanwhile, Uneven Structure aggregates important data into several dedicated nodes, which provides better reliability (illustrated in Section 3.4).

3.1.4 Code Generation. After the code segmentation and structure selection, the Approximate Code Framework generates an approximate form of the inputted erasure code. The construction of the Approximate Code is determined by four parameters (k, r, g, h) and the structure type.

We define the output Approximate Code as **APPR.CodeName (k, r, g, h, Structure Type)**, where "CodeName" is the name of the input erasure code. Structure type is Even or Uneven and the default configuration selects both two structures.

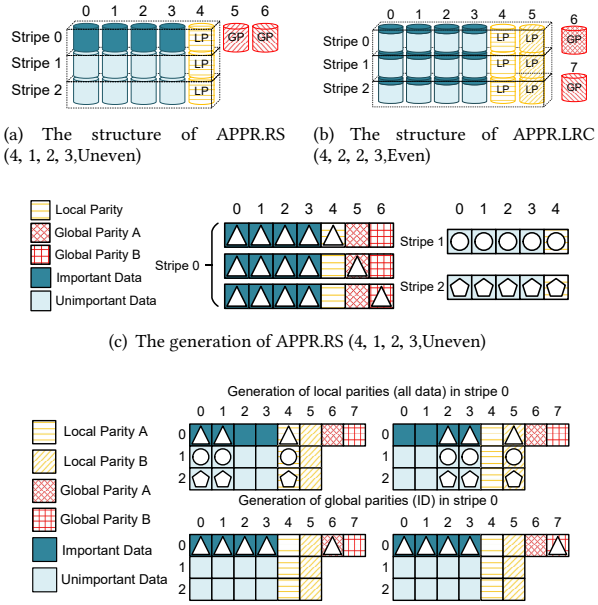
According to these parameters, Approximate Code Framework divides the data/parity nodes into several local stripes. Each local stripe has n nodes, In an n-node stripe, k of them are data nodes and $r = n - k$ nodes are local parity nodes. We define that the ratio of important data is $1/h$, and the framework utilizes h local stripes to construct a global stripe, which includes $k * h$ data nodes and $r * h$ local parity nodes in total. In a global stripe, g additional nodes are designed for global parities, which are calculated by the important data, so the total number of nodes in APPR.CodeName (k, r, g, h) is $N = h * (k + r) + g$.

Approximate Code guarantees that the unimportant data can tolerate any r node failures and the important data can tolerate any $r + g$ node failures if the inputted erasure code is an MDS code. Since the 3DFTs is a typical configuration, we mainly discuss the situation of $r + g = 3$ in the rest of our paper, so r is usually 1 or 2.

3.2 Approximate RS-based Codes

RS-based codes are based on the calculation of Galois Field (GF), which are designed for arbitrary number of data nodes. Therefore, in the code segmentation step of Approximate Code Framework, we divide the RS-based codes into two parts, r parties and g parities. In the structure selection step, the important data can be distributed in both two structures. Here we take RS and LRC codes to show how Approximate Code Framework works.

3.2.1 Approximate RS Code (APPR.RS). We consider RS(4, 3) as the inputted erasure code, as shown in Figure 5(c). Three horizontal parities are divided into one local and two global parities with the Uneven Structure, and finally we generate APPR.RS (4, 1, 2, 3,Uneven). As a result, the unimportant data and their corresponding parities constitute a RS (4, 1) array, and the important data and their corresponding parities form a RS (4, 3).



(a) The structure of APPR.RS (4, 1, 2, 3,Uneven)

(b) The structure of APPR.LRC (4, 2, 2, 3,Even)

(c) The generation of APPR.RS (4, 1, 2, 3,Uneven)

(d) The generation of APPR.LRC (4, 2, 2, 3,Even)

Figure 5: Construction of APPR.RS and APPR.LRC codes via Approximate Code Framework.

3.2.2 Approximate LRC (APPR.LRC). We take LRC (4, 2, 2) [18] as an example (shown in Figure 5(d)), which means that two local parities and two global parities are generated initially. Similar to the construction of RS code, the global parities are generated only by the important data. At last we get the APPR.LRC (4, 2, 2, 3, Even)[2].

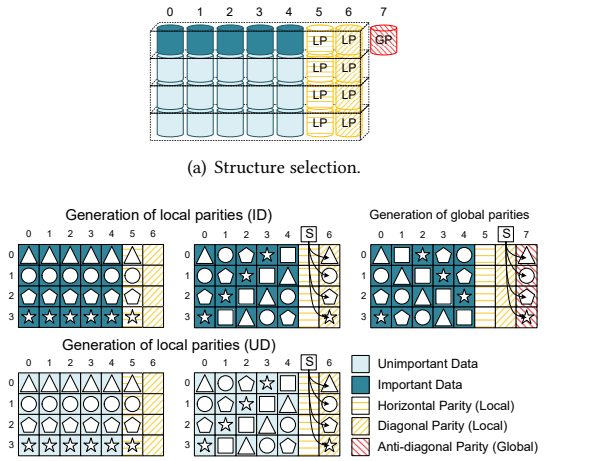
3.3 Approximate XOR-based Codes

Since we mainly provide 3DFTs for important data, we prefer to construct Approximate Code with several XOR-based codes to provide faster encoding/decoding speed than RS-based codes. Here we introduce two typical constructions of Approximate XOR-based codes, Approximate STAR Code and Approximate TIP-Code.

3.3.1 Approximate STAR Code (APPR.STAR). As introduced in Section 2.2, STAR Code [19] is a typical XOR-based erasure code for 3DFTs, which is a direct extension of EVENODD[3] (RAID-6 code) by adding the anti-diagonal parities. Figure 6 shows the construction of APPR.STAR (5, 2, 1, 4,Uneven). In this case, the horizontal and diagonal parities are regarded as local parities, the anti-diagonal parities are defined as global parities in Approximate Code Framework.

Briefly, according to the construction of APPR.STAR (5, 2, 1, 4, Uneven), the important data are encoded by STAR code while the unimportant data are generated by EVENODD code. It is clear that the important/unimportant data can tolerate triple/double nodes fail, respectively.

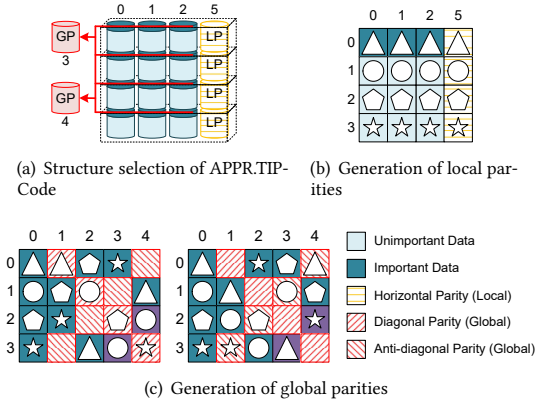
[2]Since LRC code is not an MDS erasure code, the fault tolerance of APPR.LRC (4, 2, 2, 3) is the same as APPR.RS (4, 1, 2, 3)



(a) Structure selection.

(b) Generation of APPR.STAR.

Figure 6: Construction of APPR.STAR (5, 2, 1, 4,Uneven) via Approximate Code Framework.



(a) Structure selection of APPR.TIP-Code

(b) Generation of local parities

(c) Generation of global parities

Figure 7: The construction and the encoding process of APPR.TIP (3, 1, 2, 4,Even). The node 3 and 4 in TIP-Code are defined as the global parity nodes in this case.

3.3.2 Approximate TIP-Code (APPR.TIP). We use the generation of APPR.TIP (3, 1, 2, 4,Even) as an example (shown in Figure 7). In the code segmentation phase, we split TIP-Code into two parts, the horizontal parity part, and the diagonal/anti-diagonal parities part. The important data are arranged in an Even Structure. Since the parities of the TIP-Code are distributed among several nodes, the code generation process is different from APPR.STAR.

We first generate local parity nodes by the horizontal parities for all data. Then, we reorganize the important data to construct new global stripes, where the diagonal and anti-diagonal parities are used to encode the important data. The number of global stripes are based on the ratio of important data ($1/h$). Here $h = 6$, which means that the framework utilizes the important data from six local stripes to generate one global stripe.

3.4 Reconstruction Process and Fault Tolerance

Existing work [18] has demonstrated that when we have r local parities and g global parities, the capability of fault tolerance is at least $r+g$ in typical. Thus the reconstruction (less than $r+g$ nodes fail) of the framework is classified into two cases,

(1) When $f \leqslant r$, all data (both important and unimportant data) are reconstructed via local parities.

(2) When $f \leqslant r+g$, the important data are reconstructed via both local and global parities.

However, under some special cases, important/unimportant data are also recoverable when $(f > r+g)/(f > r+1)$. Here we only consider two situations: $f = r+1$ and $f = r+g+1$, because the former just exceeds the fault tolerance of unimportant data so we calculate P_U, while the latter just exceeds the fault tolerance of important data so we calculate P_I. Since in this paper we focus on the 3DFTs, $r+g$ is set to 3.

3.4.1 When $f = r+1$. Under Even Structure, the unimportant data nodes are recoverable as long as $f \leqslant r$. There are $\binom{N}{r+1}$ cases of loss of $r+1$ nodes and there are h stripes in all. In each stripe, $\binom{k+r}{r+1}$ cases result in unrecoverable data.

Under Uneven Structure, only $h-1$ stripes contain unimportant data, so P_U is

$$P_{U-Even} = 1 - \frac{h \times \binom{k+r}{r+1}}{\binom{N}{r+1}} \quad (1)$$

$$P_{U-Uneven} = 1 - \frac{(h-1) \times \binom{k+r}{r+1}}{\binom{N}{r+1}} \quad (2)$$

3.4.2 When $f = r+g+1$ ($f = 4$). The number of cases that $f = 4$ is $\binom{N}{4}$. We calculate the total number of cases on losing the important data, which are based on the different losses of the global parity nodes. The results are as follows,

$$P_{I-Even} = 1 - h * \frac{\sum_{i=0}^{g} \binom{k+r}{4-i} * \binom{g}{i}}{\binom{N}{4}} (g = 1, 2) \quad (3)$$

$$P_{I-Uneven} = 1 - \frac{\binom{k+3}{4}}{\binom{N}{4}} \quad (4)$$

According to the above equations, for APPR.RS (3, 1, 2, 3, Even), 80.21% double failures cases are recoverable for unimportant data, and 95.50% quad failures is recoverable for important data nodes. For APRR.RS (3, 1, 2, 3, Uneven), $P_U = 86.81\%$, $P_I = 98.50\%$.

3.5 Properties of Approximate Code

We analyze the nature of the Approximate Code from the following aspects, and summarize the results shown in Table 3.

- Low Storage Overhead. Approximate Code reduces storage overhead by approximating and tiered storage strategies for important/unimportant data.
- Low Update Complexity. When one node is updated, Approximate Code needs to write r local parity nodes and only g/h global parity nodes, which are less than the inputted erasure codes.
- High reliability for important data. Important data can be recovered in all triple nodes failure and most quad nodes failure environment.
- High Flexibility. Approximate Code Framework can be applied with any erasure codes in 3DFTs, and various parameters can be set.

Table 3: Comparison of storage overhead, fault tolerance and average single write performance among typical erasure codes and their Approximate Codes.

Erasure Coding	Storage Overhead	Fault Tolerance Capability	Avg. Single Write Overhead
RS(k, r)	$(k+r)/k$	r	$r+1$
LRC(k, l, r)	$1+\frac{l+r}{k}$	$r+1$	$r+2$
STAR(p)	$(p+3)/p$	3	$6-\frac{4}{p}$
TIP-code(p)	$\frac{p+1}{p-2}$	3	4
APPR.LRC (k, r, g, h)	$\frac{(k+r)h+g}{kh}$	$1+g$	$2+\frac{g}{h}$
APPR.RS (k, r, g, h)		$r+g$	$1+r+\frac{g}{h}$
APPR.STAR (k, 2, 1, h)		3	$2\frac{k-h-1}{kh}+4$
APPR.TIP (k, 1, 2, h)		3	$2+\frac{2}{h}$

3.6 Implementation in Storage Systems

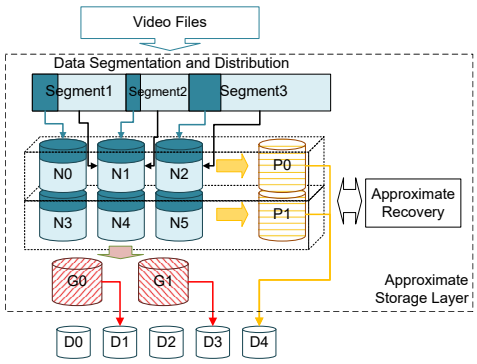


Figure 8: Overview of Approximate Storage Layer.

Due to classification of important/unimportant data, the implementation of Approximate Code in storage systems includes three modules (shown in Figure 8), data identification and distribution, Approximate Code and video recovery.

3.6.1 Data Identification and Distribution Module. This module splits the video stream files into multiple data segments, and automatically identifies the important data. A natural approach is to follow the GOP segmentation. According to Section 2.1, the GOP of the encoded video starts with an I frame, and the remaining data in the GOP depends on it for decoding. Therefore, we define the I frames as the important data, the reminder are unimportant data.

3.6.2 Approximate Code Module. The Approximate Code module is responsible for encoding/decoding the video data and completely recovering the data. For data that cannot be recovered, the Approximate Code module transmits it to the video recovery module.

3.6.3 Video Recovery Module. For unrecovered data, the video recovery module using advanced video recovery strategies illustrated in Section 2.1. Generally, the lost frames are recovered using the interpolation algorithms, where the corrupted frames are processed into fuzzy images and approximately recovered by superpixel techniques.

4 EVALUATION

In this section, we conduct a series of experiments to demonstrate the efficiency of Approximate Code Framework.

4.1 Evaluation Methodology

The experiments consist of two parts. In the first part we evaluate the performance/reliability of the Approximate Code, which includes encoding/decoding performance, single write performance, etc. In the second part we verify the capability of interpolation algorithm[25, 26, 37] to recover the lost video frames. The dataset we used is a collection of 60fps videos from **YouTube-8m** [1]. We define a 1% data loss on the unimportant video frames. The average quality of the recovered pictures is commonly above 35dB in terms of Peak Signal to Noise Ratio (PSNR). The recovered frame has a lower resolution and differs from the original image, but the unconstrained frame contains most of the context information. Therefore, the video is still available and useful for several recovery scenarios. It validates the fact that the video can tolerate a certain amount of data loss (e.g., more than $r + g + 1$ nodes).

4.1.1 Erasure Codes. We select several erasure codes and their corresponding Approximate Codes in our evaluation as below ($k = 5, 7, 9, 11, 13, 15, 17$), which can tolerant concurrent triple node failures and be used in typical cloud storage systems.

- **Approximate Code:** APPR. RS/LRC/TIP/STAR $(k, 1, 2, 4)$, APPR. RS/LRC/TIP/STAR $(k, 1, 2, 6)$. Different structures have little effect on the metrics metioned in 4.1.2. If an Approximate Code has two structures (Even/Uneven), we use the average value as the result in our comparison.
- **RS Code [30]:** RS$(k, 3)$.
- **LRC Codes [18]:** LRC $(k, 4, 2)$, LRC $(k, 6, 2)$.
- **TIP-Code [49]:** TIP$(k, 3)$.
- **STAR Code [19]:** STAR$(k, 3)$.

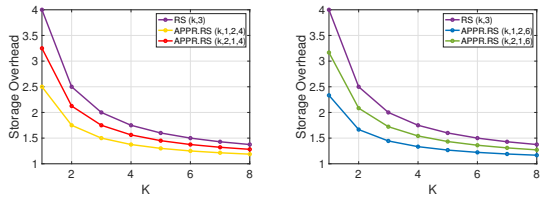
4.1.2 Metrics. We select the **Storage Overhead**, **Single Write Cost**, **Encoding Time**, **Decoding Time** and **Recovery Time** as the metrics in our evaluation.

Storage overhead is the ratio between the total amount of data (all data and parities) and the original data. Single write cost is the average number of I/O operations to handle a single write. Encoding Time is the time for generating all parity nodes and decoding time is the computation time for recovering the lost nodes. Recovery time is the time to recover a lost data, which consists of computation time, I/O overhead, transmission time, etc.

4.1.3 Experimental Environment. The experimental environment is shown in Table 4. We set a Hadoop system with one NameNode and h DataNodes. Each DataNode is the same as data nodes in Approximate Code, and the global parities are distributed on each DataNodes. We set $h = 4, 6$, and the size of each node is $1GB$.

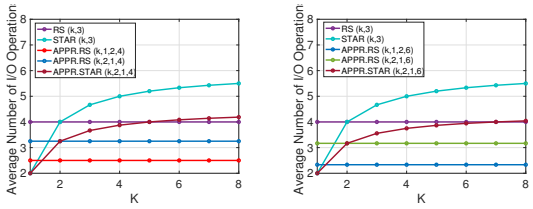
Table 4: Details of Our Evaluation Platform.

Description	DELL R730 Server
CPU	Intel Xeon 3.0GHz
NIC	10Gbps
Memory	32GB
Disk	8TB HDD
OS	Linux 3.19
Platform	Hadoop HDFS 3.0.3



(a) h=4 (b) h=6

Figure 9: Storage overhead comparison between RS and APPR.RS.



(a) h=4 (b) h=6

Figure 10: Single write performance comparison among RS, STAR, APPR.RS and APPR.STAR.

4.2 Mathematical Analysis

In this section, we show the numerical results of mathematical analysis by comparing the Approximate Code with its corresponding original code.

- **Storage Overhead:** We compare the storage overhead between RS $(k, 3)$, APPR.RS $(k, 1, 2, h)$ and APPR.RS $(k, 2, 1, h)$, and the results are shown in Figure 9. It is clear that APPR.RS has much lower storage overhead than RS Code. The degradation is up to 21.4% when $h = 4, k = 4$, and 23.8% when $h = 6, k = 4$. When $k = 6$, APPR.RS $(6, 1, 2, 4)$ reduces the average number of parity nodes from 3 to 1.33. Table 5 shows the improvements on storage overhead, note that the results also applies to other 3DFTs MDS codes.
- **Single Write Performance:** We compare the single write cost among different erasure codes, and the results are shown in Figure 10. We notice that APPR.RS has the lowest single write time, which decreases the average number of I/Os by up to 41.3%.

Table 5: Improvement of APPR.RS codes over RS(k, 3) on storage overhead.

Coding Method	Number of Data Nodes					
	4	5	6	7	8	9
APPR.RS(k, 1, 2, 4)	21.4%	18.8%	16.7%	15.0%	13.6%	12.5%
APPR.RS(k, 2, 1, 4)	10.7%	9.4%	8.3%	7.5%	6.8%	6.2%
APPR.RS(k, 1, 2, 6)	23.8%	20.8%	18.5%	16.7%	15.2%	13.9%
APPR.RS(k, 2, 1, 6)	11.9%	10.4%	9.3%	8.3%	7.6%	6.9%

4.3 Experimental Results

In this subsection, we show the experimental results of various codes in terms of encoding/decoding time and recovery time.

4.3.1 Encoding Time. The results are shown in Figure 11.

- Compared to STAR/TIP Codes: Approximate Code encodes faster than STAR/TIP code by up to 51.7%, 54.3%, respectively.
- Compared to RS/LRC Code: Approximate Code improve the encoding significantly. The optimization ratios are up to 54.0% for RS code, 61.7% for LRC codes, respectively.

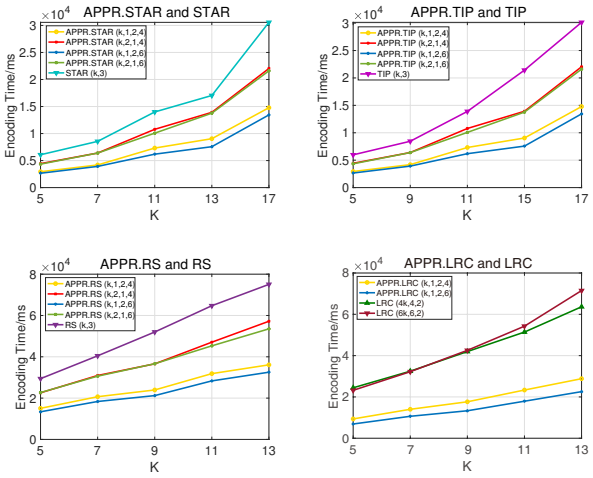


Figure 11: Encoding Time Comparison among various erasure coding methods.

4.3.2 Decoding Time under Single Node Failure Condition. The results are shown in Table 6.

- Compared to STAR/TIP Codes: The decoding time of Approximate Code is almost the same as original code when single node fails, with an increment by up to 5.5% for STAR code, up to 10.8% for TIP-Code, respectively.
- Compared to RS/LRC Codes: Approximate Code decodes nearly the same as RS code (improvements by up to 0.9%) and LRC codes (enhancement by up to 1.3%).
- In some scenarios, Approximate Codes are worse than original codes, which is affected by small differences on parameter configurations.

4.3.3 Decoding Time under Double Nodes Failure Condition. The results are shown in Figure 12.

- Compared to STAR/TIP Codes: The recovery acceleration is up to 76.5% for STAR, up to 77.6%, respectively.
- Compared to RS/LRC Codes: Approximate Code reconstruct the lost data faster than original erasure codes, which improve the reconstruction by up to 75.4% for RS code, and 78.7% for LRC codes, respectively.

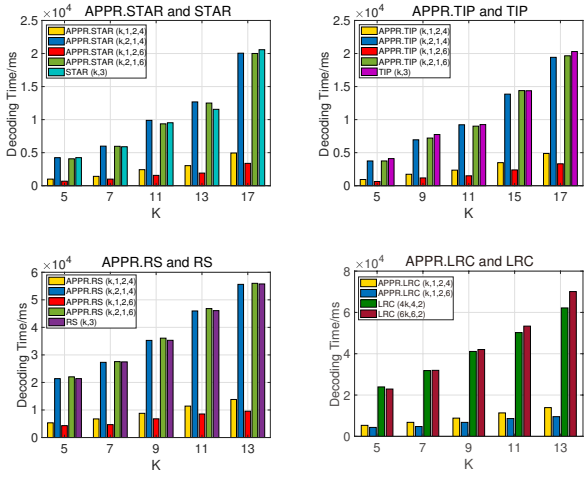


Figure 12: Double Nodes Decoding Time Comparison among various erasure coding methods.

4.3.4 Decoding Time under Triple Nodes Failure Condition. The results are shown in Figure 13.

- Compared to STAR Code: The maximum optimization ratio is 75.1% (between APPR.STAR (k, 1, 2, 6) and STAR (k, 3) when $k = 5$).
- Compared to TIP-Code: This condition is similar to STAR code with an optimization ratio is up to 75.6% (between APPR.TIP(k, 1, 2, 6) and TIP(k, 3) when $k = 9$).
- Compared to RS Code: Approximate Code decodes largely faster than RS code under three failure nodes condition by up to 75.0% (between APPR.RS (k, 1, 2, 6) and RS(k, 3) when $k = 5$).
- Compared to LRC Codes: Approximate Code largely reduces the decoding time by up to 87.9% (between APPR.LRC (k, 1, 2, 6) and LRC (k, 6, 2) when $k = 13$).

We combined the encoding/decoding time of all erasure codes in Figure 14 when $k = 5$. The results show that Approximate Code has the best encoding/decoding performance among all erasure codes.

4.3.5 Recovery Time. Figure 15 shows the recovery time of various erasure codes (ECs) under two and three failure nodes condition. We can see that Approximate Code owns the best recovery performance in all ECs. The optimization ratio is up to 95.9%. The main reason is that Approximate Code only recover important data under multiple failure nodes condition which can significantly reduce computation time, I/O overhead and transmission time.

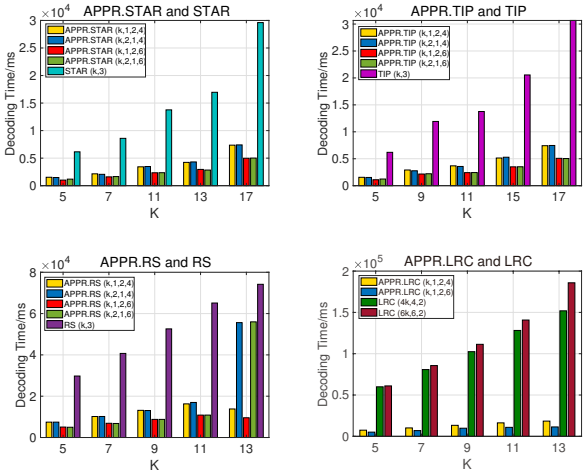


Figure 13: Triple Nodes Decoding Time Comparison among various erasure coding methods.

Table 6: Improvement of Approximate Codes (k, 1, 2, 4) over their corresponding erasure codes.

Experiment Scenario	Coding Method	Number of Data Nodes				
		5	7	9	11	13
Encoding	RS	48.80%	48.91%	53.96%	50.79%	51.90%
	STAR	51.65%	51.04%	/	48.45%	46.92%
	TIP	51.68%	/	54.29%	48.43%	/
	LRC	61.69%	56.82%	57.95%	54.60%	54.71%
Decoding under Single Node Failure	RS	0.09%	0.89%	0.18%	0.62%	0.57%
	STAR	5.50%	3.67%	/	-2.05%	-5.26%
	TIP	8.78%	/	10.79%	-1.68%	/
	LRC	1.32%	-0.75%	-0.91%	-3.42%	-9.54%
Decoding under Double Node Failure	RS	75.03%	75.35%	75.10%	75.23%	75.22%
	STAR	76.46%	75.98%	/	74.54%	73.73%
	TIP	77.11%	/	77.63%	74.53%	/
	LRC	77.75%	78.70%	78.53%	77.41%	77.62%
Decoding under Triple Node Failure	RS	75.00%	75.01%	74.99%	74.99%	75.02%
	STAR	75.11%	74.94%	/	75.11%	75.03%
	TIP	74.85%	/	75.57%	73.23%	/
	LRC	87.58%	87.40%	87.00%	87.29%	87.87%

4.4 Analysis

The improvements of Approximate Code over typical erasure codes are listed in Table 5 and Table 6. From the table, we notice that Approximate Code is a cost-effective solution for 3DFTs. There are several reasons to achieve these gains. First, Approximate Code only provides triple parities for important data, which leads to the degradation of storage cost and improvements on encoding/decoding performance. Second, Approximate Code simplifies the generation of local parities, which reduces the computational complexity. Third, the parity chains in Approximate Code are short, which decreases the transmission time and I/O overhead in cloud storage systems.

5 CONCLUSION

In this paper, we propose a novel erasure coding framework called Approximate Code, which is used for tiered video storage in cloud systems. The framework provides high reliability for important data and low reliability for unimportant. By this way, the storage cost can be saved. The experimental results show that Approximate Code have the following advantages, (1) reduces the number of parities by up to 55%, (2) saves the storage cost by up to 20.8%, (3) increases the recovery speed by up to a factor of 4.7X under double/triple nodes failure conditions.

ACKNOWLEDGEMENT

We thank anonymous reviewers for their insightful comments. This work is partially sponsored by the National Key R&D Program of China (No.2018YFB0105203), the National 973 Program of China (No.2015CB352403), the Natural Science Foundation of Shanghai (No.18ZR1418500), the Alibaba Group through Alibaba Innovative Research (AIR) program, and the Open Research fund by Data Storage System Lab, Ministry of Education of China.

REFERENCES

[1] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. 2016. Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675* (2016).

[2] Ignacio Bermudez, Stefano Traverso, Marco Mellia, and Maurizio Munafo. 2013. Exploring the cloud from passive measurements: The Amazon AWS case. In *2013 Proceedings IEEE INFOCOM*. IEEE, 230–234.

[3] Mario Blaum, Jim Brady, Jehoshua Bruck, and Jai Menon. 1995. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Transactions on computers* 44, 2 (1995), 192–202.

[4] Mario Blaum and Ron Roth. 1999. On Lowest Density MDS Codes. *IEEE Transactions on Information Theory* 45, 1 (1999), 46–59.

[5] Johannes Bloemer, Malik Kalfane, Richard Karp, Marek Karpinski, Michael Luby, and David Zuckerman. 1995. *An XOR-based Erasure-Resilient coding scheme.* Technical Report TR-95-048. International Computer Science Institute.

[6] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, et al. 2011. Windows Azure Storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 143–157.

[7] Yuval Cassuto and Jehoshua Bruck. 2009. Cyclic Lowest Density MDS Array Codes. *IEEE Transactions on Information Theory* 55, 4 (2009), 1721–1729.

[8] Peter Corbett, Bob English, Atul Goel, Tomislav Grcanac, Steven Kleiman, James Leong, and Sunitha Sankar. 2004. Row-Diagonal Parity for Double Disk Failure Correction. In *Proc. of the USENIX FAST'04*. San Francisco, CA.

[9] Peter Corbett and Atul Goel. 2011. Triple parity technique for enabling efficient recovery from triple failures in a storage array. US Patent 8,015,472.

[10] Peter Deutsch. 1996. *DEFLATE compressed data format specification version 1.3.* Technical Report.

[11] Gui Liang Feng, Robert Deng, Feng Bao, and J C Shen. 2005. New efficient MDS array codes for RAID Part I: Reed-Solomon-like codes for tolerating three disk failures. *IEEE Trans. Comput.* 54, 9 (2005), 1071–1080.

[12] Junqing Gu, Chentao Wu, Xin Xie, Han Qiu, Jie Li, Minyi Guo, Xubin He, Yuanyuan Dong, and Yafei Zhao. 2019. Optimizing the Parity-Check Matrix for Efficient Decoding of RS-based Cloud Storage Systems. In *The 32nd IEEE International Parallel & Distributed Processing Symposium (IPDPS 2019)*. IEEE.

[13] Junqing Gu, Xin Xie, Han Qiu, Jie Li, Minyi Guo, and Xubin He. 2019. Optimizing the Parity-Check Matrix for Efficient Decoding of RS-based Cloud Storage Systems. In *The 35th International Conference on Massive Storage Systems and Technology (MSST 2019)*. IEEE.

[14] Qing Guo, Karin Strauss, Luis Ceze, and Henrique S Malvar. 2016. High-density image storage using approximate memory cells. In *ACM SIGPLAN Notices*, Vol. 51. ACM, 413–426.

[15] James Hafner. 2005. WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies*, Vol. 5. 16–16.

[16] James Hafner. 2006. HoVer Erasure Codes For Disk Arrays. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*. 217–226.

[17] Cheng Huang, Minghua Chen, and Jin Li. 2007. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. In *Network Computing and Applications, 2007 NCA'07. Sixth IEEE International Symposium on*. IEEE, 79–86.

[18] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. 2012. Erasure coding in windows azure storage. In *the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*. 15–26.

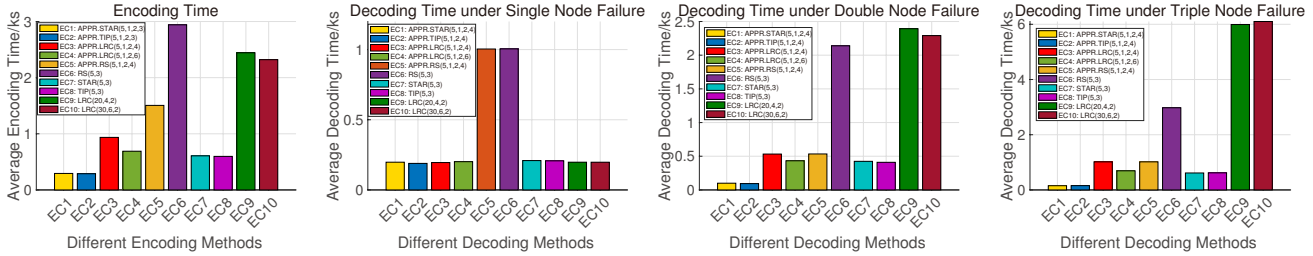
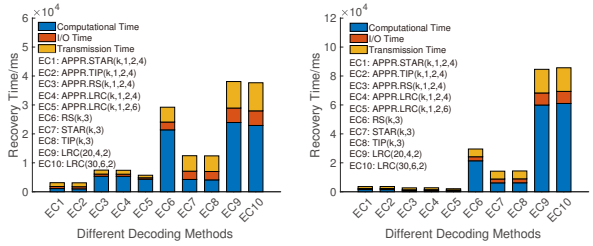


Figure 14: Comparison of several metrics with five data nodes ($k = 5$).



(a) Double nodes recovery (b) Triple nodes recovery

Figure 15: Recovery time comparison under double/triple nodes failure among various erasure coding methods.

[19] Cheng Huang and Lihao Xu. 2008. STAR: An efficient coding scheme for correcting triple storage node failures. *IEEE Trans. Comput.* 57, 7 (2008), 889–901.

[20] Djordje Jevdjic, Karin Strauss, Luis Ceze, and Henrique S Malvar. 2017. Approximate storage of compressed and encrypted videos. In *ACM SIGARCH Computer Architecture News*, Vol. 45. ACM, 361–373.

[21] Yanbing Jiang, Chentao Wu, Jie Li, and Minyi Guo. 2016. BDR: A Balanced Data Redistribution scheme to accelerate the scaling process of XOR-based Triple Disk Failure Tolerant arrays. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 72–79.

[22] Chao Jin, Hong Jiang, Dan Feng, and Lei Tian. 2009. P-Code: A new RAID-6 code with optimal properties. In *Proc. of the ICS'09*. Yorktown Heights, NY.

[23] KR Krish, Ali Anwar, and Ali R Butt. 2014. hats: A heterogeneity-aware tiered storage for hadoop. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 502–511.

[24] Jai Menon. 1995. A performance comparison of RAID-5 and log-structured arrays. In *Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing*. IEEE, 167–178.

[25] Simone Meyer, Oliver Wang, Henning Zimmer, Max Grosse, and Alexander Sorkine-Hornung. 2015. Phase-based frame interpolation for video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1410–1418.

[26] Simon Niklaus and Feng Liu. 2018. Context-aware synthesis for video frame interpolation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1701–1710.

[27] D. Patterson et al. 2008. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proc. of the SIGMOD'88*.

[28] J. Plank. 2008. A new minimum density RAID-6 code with a word size of eight. In *Proc. of the IEEE NCA'08*. Cambridge, MA.

[29] J. Plank. 2008. The RAID-6 Liberation Codes. In *Proc. of the USENIX FAST'08*. San Jose, CA.

[30] Irving S Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics* 8, 2 (1960), 300–304.

[31] Adrian Sampson, Jacob Nelson, Karin Strauss, and Luis Ceze. 2014. Approximate storage in solid-state memories. *ACM Transactions on Computer Systems (TOCS)* 32, 3 (2014), 9.

[32] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. 2013. XORing elephants: Novel erasure codes for big data. In *Proceedings of the VLDB Endowment*, Vol. 6. VLDB Endowment, 325–336.

[33] Zhirong Shen and Jiwu Shu. 2014. HV Code: An All-Around MDS Code to Improve Efficiency and Reliability of RAID-6 Systems. In *Proc. of the IEEE/IFIP DSN'14*. Atlanta, GA.

[34] Dan Tang, Xiaojing Wang, Sheng Cao, and Zheng Chen. 2008. A New Class of Highly Fault Tolerant Erasure Code for the Disk Array. In *Workshop on Power Electronics and Intelligent Transportation System, PEITS'08*. IEEE, 578–581.

[35] C. Tau and T. Wang. 2003. Efficient parity placement schemes for tolerating triple disk failures in RAID architectures. In *Proc. of the AINA'03*. Xi'an, China.

[36] Aniruddha N Udipi, Naveen Muralimanohar, Rajeev Balsubramonian, Al Davis, and Norman P Jouppi. 2012. LOT-ECC: localized and tiered reliability mechanisms for commodity memory systems. In *ACM SIGARCH Computer Architecture News*, Vol. 40. IEEE Computer Society, 285–296.

[37] Joost van Amersfoort, Wenzhe Shi, Alejandro Acosta, Francisco Massa, Johannes Totz, Zehan Wang, and Jose Caballero. 2017. Frame interpolation with multi-scale deep loss functions and generative adversarial networks. *arXiv preprint arXiv:1711.06045* (2017).

[38] S. Wan et al. [n. d.]. Code-M: A Non-MDS Erasure Code Scheme to Support Fast Recovery from up to Two-Disk Failures in Storage Systems. In *Proc. of the DSN'10*.

[39] Hui Wang and Peter Varman. 2014. Balancing Fairness and Efficiency in Tiered Storage Systems with Bottleneck-Aware Allocation. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)*. 229–242.

[40] Y. Wang, G. Li, and X. Zhong. 2012. Triple-Star: A Coding Scheme with Optimal Encoding Complexity for Tolerating Triple Disk Failures in RAID. *International Journal of Innovative Computing, Information and Control* 8, 3 (2012), 1731–1472.

[41] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. 2003. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology* 13, 7 (2003), 560–576.

[42] Chentao Wu, Xubin He, Jizhong Han, Huailiang Tan, and Changsheng Xie. 2012. SDM: A stripe-based data migration scheme to improve the scalability of RAID-6. In *2012 IEEE International Conference on Cluster Computing*. IEEE, 284–292.

[43] Chentao Wu, Shenggang Wan, Xubin He, Qiang Cao, and Changsheng Xie. 2011. H-Code: A hybrid MDS array code to optimize partial stripe writes in RAID-6. 782–793. https://doi.org/10.1109/IPDPS.2011.78

[44] Xin Xie, Chentao Wu, Junqing Gu, Han Qiu, Jie Li, Minyi Guo, Xuebin He, Yuanyuan Dong, and Yafei Zhao. 2019. AZ-Code: An Efficient Availability Zone Level Erasure Code to Provide High Fault Tolerance in Cloud Storage Systems. In *The 35th International Conference on Massive Storage Systems and Technology (MSST 2019)*. IEEE.

[45] Lihao Xu, Vasken Bohossian, Jehoshua Bruck, and David Wagner. 1999. Low-Density MDS Codes and Factors of Complete Graphs. *IEEE Transactions on Information Theory* 45, 6 (1999), 1817–1826.

[46] Lihao Xu and Jehoshua Bruck. 1999. X-Code: MDS Array Codes with Optimal Encoding. *IEEE Transactions on Information Theory* 45, 1 (1999), 272–276.

[47] Gong Zhang, Lawrence Chiu, Clem Dickey, Ling Liu, Paul Muench, and Sangeetha Seshadri. 2010. Automated lookahead data migration in SSD-enabled multi-tiered storage systems. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–6.

[48] Yongzhe Zhang, Chentao Wu, Jie Li, and Minyi Guo. 2015. PCM: A Parity-check Matrix Based Approach to Improve Decoding Performance of XOR-based Erasure Codes. In *The 34th International Symposium on Reliable Distributed Systems (SRDS 2015)*. IEEE.

[49] Yongzhe Zhang, Chentao Wu, Jie Li, and Minyi Guo. 2015. Tip-code: A three independent parity code to tolerate triple disk failures with optimal update complextiy. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 136–147.

[50] Hengyu Zhao, Linuo Xue, Ping Chi, and Jishen Zhao. 2017. Approximate image storage with multi-level cell STT-MRAM main memory. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 268–275.

[51] Jacob Ziv and Abraham Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on information theory* 23, 3 (1977), 337–343.

[52] Jacob Ziv and Abraham Lempel. 1978. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory* 24, 5 (1978), 530–536.