

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

Huayi Jin

## 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 with 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 challenge for cloud storage systems. Although there are several solutions like approximate storage technique for storage devices, it cannot guarantee low storage cost and high data reliability in storage systems concurrently.

To address this problem, in this paper, we propose Approximate Code, which is an erasure coding framework for video applications in cloud storage systems. The key idea of Approximate Code is distinguishing the important data and unimportant data in videos with different capabilities of fault tolerance. On one hand, for important data, Approximate Code provides triple parities to provide 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, STAR and TIP-Code, Approximate Code reduces the number of parities by up to 60%, saves the storage cost by up to 10%, increase the recovery speed by up to 1.5X when single disk fails, and can reconstruct the whole video data via fuzzification when triple disks fail.

## KEYWORDS

Erasure Codes, Approximate Storage, Multimedia, Cloud Storage

### ACM Reference Format:

Huayi Jin. 2019. Approximate Code: A Cost-Effective Erasure Coding Framework for Video Applications in Cloud Storage Systems. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

## 1 INTRODUCTION

For typical cloud storage systems such as Windows Azure [CITE] and Amazon AWS [CITE], erasure coding is a popular technique to provide both high reliability and low monetary cost [CITE EC], where triple disk failure fault tolerant arrays (3DFTs) are widely

used. Typical erasure codes can be divided into two categories, RS-based Codes [CITE] and XOR-based codes [CITE]. RS-based codes [CITE] are encoded according the Galois Field Computation in Reed Solomon Code [CITE], which allow flexible configuration and have a little higher computation cost. XOR-based codes [CITE] simplify the computation, but the scalability is a significant issue in previous literatures.

With the increasing demand on higher resolution and frame rate for video data, tremendous storage devices are highly desired in cloud storage systems, which makes data centers much bigger. On YouTube, nearly 140,000 hours of video are played every minute and 400 hours of video are uploaded [DESCRIBE AND CITE YOUTUBE]. Therefore, in this paper, we set out to answer the following question, In a cloud storage system, how to efficiently store the tremendous video data in 3DFTs?

To reduce the storage cost in cloud storage systems, a feasible solution is approximate storage. [DESCRIBE AND CITE APPROXIMATE STORAGE] but the data reliability cannot be guaranteed. Another solution is using disk arrays like RAID-5 or RAID-6 [CITE RAID], but the capability of fault tolerance should be sacrificed. Therefore, existing solutions cannot provide low storage cost and high reliability simultaneously.

To address the above problem, in this paper, we propose Approximate Code, which is an erasure coding framework to provide comprehensive solution for 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 of the recovery. When triple disks fail, the lost data can be reconstructed via a fuzzy manner.

We have the following contributions of this work,

- (1) We propose Approximate Code, which a cost-effective framework to store video data in cloud storage systems.
- (2) Approximate Code can be implemented by combining most erasure codes in 3DFTs, such as RS, STAR Code, TIP-Code, etc.
- (3) We conduct several quantitative analysis, simulations and experiments according to different layouts of various erasure codes, and the results show that Approximate Code achieves lower storage cost and faster data recovery when single disk fails.

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 and its encoding and decoding process will be illustrated in detail. Section 4 introduce the implementation of our design, and section 5 analysis the performance of approximate video storage. The evaluation is presented in Section 6 and the conclusion of our work is in Section 7.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

Conference'17, July 2017, Washington, DC, USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

**Table 1: The symbols used in this paper.**

Symbols	Description
$k$	the number of data nodes in an array
$r$	the number of local parity nodes in an array
$n$	the number of nodes in an array ( $n = k + r$ )
$h$	the number of arrays
$s$	the number of blocks in a node
LP	local parity nodes
GP	global parity nodes
ID	important data
UD	unimportant data

## 2 RELATED WORK AND OUR MOTIVATION

In this section, we introduce the background of Video frame recovery, 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 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 [1], RDP code [8], Blaum-Roth code[3], X-code [44], Liberation code [28], Liber8tion code [27], Cyclic [7] code, B-Code [45], Code-M [40], H-code [43], P-code [20] and HVcode [36], etc.

In Triple Disks Failure Tolerant Arrays (3DFTs), typical MDS codes include Reed-Solomon codes [33], Cauchy-RS codes [4], STAR code [19], Triple-Star code [41], Triple-Parity code [9], HDD1 code [39], RSL-code [10], RL-code [11], and so on. Typical non-MDS codes contain WEAVER codes [14], HoVer codes [13], T-code [38], HDD2 code [39], Pyramid codes [18], Local Reconstruction Codes [16], Locally Repairable Codes [34], etc.

Several classic erasure codes are illustrated in detail as below,

### 2.2 Video Frame Recovery

In the circumstance of video approximate storage, it's common to lose some frames and leave the video incomplete. However, the lost frames may still be recoverable with the benefit of nowadays powerful deep learning techniques. One of them is named video frame interpolation.

Video frame interpolation is one of the basic video processing techniques, an attempt to synthetically produce one or more intermediate video frames from existing ones, the simple case being the interpolation of one frame given two consecutive video frames. This is a technique that can model natural motion within a video,

and generate frames according to this modelling. Artificially increasing the frame-rate of videos enables the possibility of frame recovery.

In deep learning methods, optical changes between the frames are trained in a supervised setup mapping two frames to their ground truth optical flow. Among all these, a multi-scale network[6] based on recent advances in spatial transformers and composite perceptual losses as well as a context-aware Synthesis approach[4] have so far produced the new state-of-the-art results in terms of PSNR and middlebury benchmark respectively.

### 2.3 Approximate Storage

Storage techniques nowadays generally regard all information of the same importance, which causes significant costs in energy, disk drives and computing resources. But not all data need high-reliability storage for its backup. That is when the concept of approximate storage is introduced. Approximate Storage is another way outside of traditional methods of trading off the limited resource budget with the costly reliability requirements, which recently receives more attentions since data centers are faced with storage pressure from the ever-increasing data.

Use cases for approximate storage range from transient memory to embedded settings and mass storage cloud servers. Mapping approximate data onto blocks that have exhausted their hardware error correction resources, for example, to extend memory endurance. On embedded settings, it enables the reduction of the cost of accesses and preserve battery life to loosen the capacity constraints. [5] Here, in data-center-scale video database, approximate storage can provide multiple levels of fault tolerance for data of different importance, avoiding redundant backup for the less-important data, thus saving a significant amount of space.

Approximate storage loosens the requirement of storage reliability by allowing quality loss of some specific data. Therefore, programmers can specify the importance of the data segments and assign them to different storage blocks. The critical data is still safe because they are stored and sufficiently backed up by expensive and highly reliable storage devices. Meanwhile, non-critical data is exposed to error, thus increasing storage density and saving cost.

However, it is too naive to store data in approximate storage units indiscriminately. Related research [2] shows that this can lead to unacceptable data pollution. To ensure data quality in this case, higher error correction costs are required resulting in an increase in overall storage costs. Therefore, it is very important to distinguish the importance of video data. This can usually be done by an experienced expert or automatically by a program.

**Table 2: Comparison of fault tolerance and storage overhead between approximate storage, EC and Approximate Code**

Schemes		Storage Overhead	Realibilities	Performance
EC	RS	high	high	medium
	RAID 6	medium	medium	high
Ap-Storage		low	low	high
Ap-Code	Imp	low	high	high
	Unimp		medium	high

## 2.4 Our Motivation

Based on Table 2, either the existing erasure codes or the approximate storage methods cannot meet the requirements of video applications in the cloud storage system due to the following reasons.

First, existing erasure codes generally reach or exceed 3DFTs, and use more than 3 parity disks. However, the simultaneous damage of 3 disks is very rare, and the storage overhead paid for this is too large. Second, the existing erasure codes provide the same fault tolerance for all data without distinction, which results in the same reliability of important data that is sensitive to errors and data that is robust. Last but not least, the current approximate storage methods are unreliable since they are not designed to tolerate disk level failure.

To solve these problems, we propose a new erasure code called Approximation Code. It provides different fault tolerance for important and non-critical data to reduce storage overhead and protect critical data better.

## 3 APPROXIMATE CODE

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

### 3.1 Overview of Approximate Code

The construction of the Approximate Code is determined by 3 parameters  $k$ ,  $r$  and  $h$ . In a  $n$ -node array,  $k$  of them are data nodes and  $r = n - k$  nodes are for local parity, where each node can be divided into multiple blocks, and we label the number of blocks in a node as  $s$ . Approximate Code are designed for  $h$  arrays, therefore it includes  $k * n$  data nodes and  $r * h$  local parity nodes. Besides them,  $3 - r$  nodes are designed for global parity, and the total number of nodes in Approximate Code ( $k, r, h$ ) is

$$h * (k + r) + 3 - r$$

### 3.2 Framework Design

The Approximate Code provides an erasure code framework for video storage in the cloud system. It uses an approximate strategy to split the erasure code scheme into two parts with different fault tolerances for application to important and minor data. It mainly includes two operations of coding segmentation and structure selection.

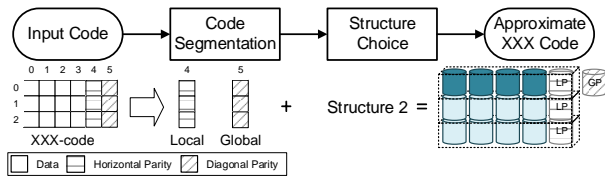


Figure 1: The framework of Approximate Code

**3.2.1 Code segmentation.** For any type of input erasure code, we assume that its fault tolerance is  $x$ . The Approximate Code first splits its parity block into two parts: local and global parity. The former verifies all important and non-critical data, while the latter

only verifies important data. Code segmentation are designed to ensure that local parities can tolerate any  $r$  node failures, so the fault tolerance of non-critical data is  $r$ . For important data, the code segmentation ensures that the parity block of important data can be completely recombined into the original xDFTs erasure code scheme.

**3.2.2 Structure selection.** The Approximate Code framework then choose the Structure for the input erasure code. Approximate Code distribute important and unimportant data separately and mainly in two structure. As shown in Figure 2, in *Structure 1*, important data occupies 1 of  $s$  blocks in each node, and in *Structure 2*, it occupies 1 of all  $h$  data arrays. We define that  $h = s$ , so in *Structure 1* and *Structure 2*, important data accounts for  $1/h$  of all data.

Figure 1 shows the process of turning a code (assume that it is XXX) to Approximate XXX code. XXX-code has two parities chunks, the horizontal and the diagonal. Approximate Code framework define the former as local parity chunk and the latter as the global parity chunk. Then *Structure 2* is selected and the Approximate XXX Code is a 16-node system.

In each array,  $r$  nodes are local parity nodes, which is encoded by the rest  $k$  data nodes in the array. In addition, there are another  $3 - r$  global nodes, which are calculated only by the important data blocks. It should be point out that the global parity nodes should be placed in different nodes and should be different from the data nodes.

The number of important data blocks is  $h \times k$ , and they generate  $h \times (3 - r)$  global parity blocks. To sum up, the number of blocks involved in the Approximate Code is

$$(k + r)h^2 + (3 - r)h$$

Approximate Code guarantees that for each array, the unimportant data can tolerate any  $r$  node failures. Usually,  $r$  is equal to 1 or 2, which covers most node failure scenarios, while important data are provided 3DFTs.

The construction of Approximate Code (4,1,4, *Structure 1*) and Approximate (3,1,3, *Structure 2*) can be illustrated by Figure 2,

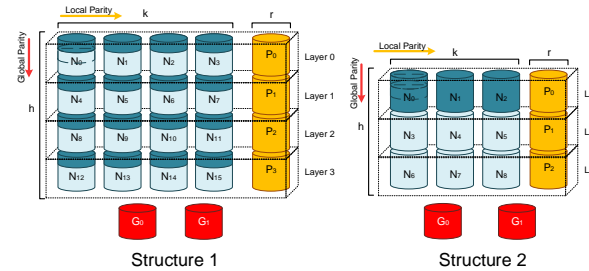


Figure 2: A sample of Approximate Codes (4, 1, 4) in *Structure 1*, and Approximate Code (3, 1, 3) in *Structure 2*. Data nodes are marked as  $N_i$ , important data are marked as water green, local parity nodes are marked as orange and global parity nodes are marked as red.

$$\begin{bmatrix} A_{10} & A_{11} & A_{12} & A_{13} & 0 & 1 & 0 \\ A_{20} & A_{21} & A_{22} & A_{23} & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} ID_0 \\ ID_1 \\ ID_2 \\ ID_3 \\ P_0 \\ G_0 \\ G_1 \end{bmatrix} = 0$$

$$\begin{bmatrix} A_{10} & A_{11} & A_{12} & A_{13} & 1 \end{bmatrix} * \begin{bmatrix} UD_0 & ID_0 \\ UD_1 & ID_1 \\ UD_2 & ID_2 \\ UD_3 & ID_3 \\ P_0 \end{bmatrix} = 0$$

**Figure 3: The generate matrix of RS-based Approximate Codes (4, 1, 4), where  $ID_i$  and  $UD_i$  represent the important and unimportant data in  $N_i$ .  $G_0$  and  $G_1$  are the parity blocks corresponding to important data blocks.**

### 3.3 RS-based Approximate Code

The construction of the Approximate Code based on RS is simple. In *Structure 1*, the encoding unit is block. A horizontal RS parity is performed on the corresponding block in each node to generate a local parity block. At the same time, the important data blocks is horizontally verified by RS and generate multiple global check blocks.

In *Structure 2*, the encoding unit is node. We perform horizontal RS parity in each array to generate local parity nodes. The global parity node is generated only by the important data array.

Figure 3 is the RS generation matrix with  $k = 4$  and  $r = 1$ . It shows that to ensure that the coefficients of the local and the global parity are linearly independent is easy. If necessary, Approximation Code can easily achieve higher reliability guarantees based on RS, but in most cases this is excessive, so we will not discuss it here.

### 3.4 XOR-based Approximate Code

Since we mainly provide 3DFTs for important data, we prefer to construct Approximate Code with several RAID6-based codes to provide faster encoding and reconstruction speed than RS. This section introduce typical cases of the construction of XOR-based Approximate Code (5, 1, 6, *Structure 1*) and (4, 2, 4, *Structure 2*).

**3.4.1 RAID5-TIP Approximate Code.** Figure 4 is a typical construction of Approximate Code (5, 1, 6) in *Structure 1*, based on RAID5 and TIP-code [7]. RAID5 tolerate one node failure while TIP-code is an XOR-based 3DFTs code. In each array, local parity blocks provide horizontal XOR parity, and between arrays, global parity provide diagonal parity and anti-diagonal parity.

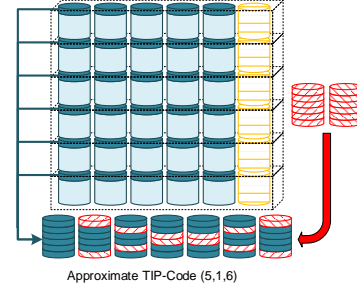
The local (horizontal) parity elements are calculated by the following encoding equations.

$$LP_{i,0} = \bigoplus_{j=0}^{k-1} ID_{i,j} (0 \leq i < h) \quad (1)$$

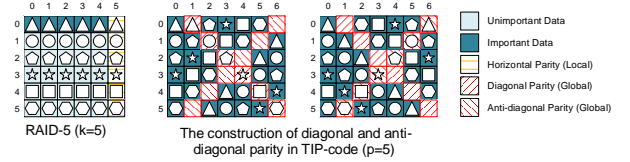
$$LP_{i,p} (0 < p < s) = \bigoplus_{j=k(p-1)}^{kp-1} UD_{i,j} (0 \leq i < h) \quad (2)$$

When generating global parity blocks, the important data block, the local parity block, and the global parity block should be arranged as the encoding form of TIP-code. Figure 5 shows the coding method of diagonal and anti-diagonal parity blocks. Since the encoding

process of important data blocks can be seen as RAID5 plus TIP-code, it is obvious that they achieve 3DFTs.



**Figure 4: The XOR-based Approximate Code (5, 1, 6, *Structure 1*), constructed based on RAID5 and TIP-code.**



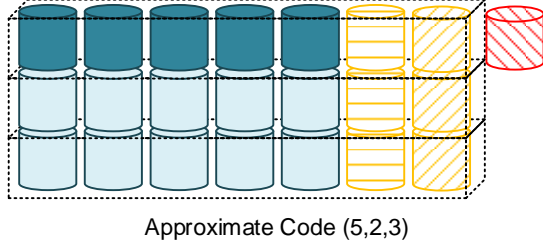
**Figure 5: Encoding pf TIP-code**

**3.4.2 EVENODD-STAR Approximate Code.** Figure 6 shows the construction of Approximate Code (5, 2, 4) in *Structure 2* based on EVENODD [1] and STAR [3]. EVENODD is a typical RAID6 erasure code scheme, while STAR is a 3DFTs extension scheme of EVENODD. In this case, important data blocks are distributed in array 0, while unimportant data blocks occupy the rest arrays.  $LP_{i,0}$  to  $LP_{i,3}$  consist the horizontal parity nodes and  $LP_{i,4}$  to  $LP_{i,7}$  consist diagonal parity nodes for array  $i$ . So the form of every array fit the construction of EVENODD. Because the STAR code is constructed just by adding a reverse parity chain to the EVENODD code. The  $GP$  is the anti-diagonal parity node and guarantee important data nodes form the construction method of STAR, which ensures the 3DFTs.

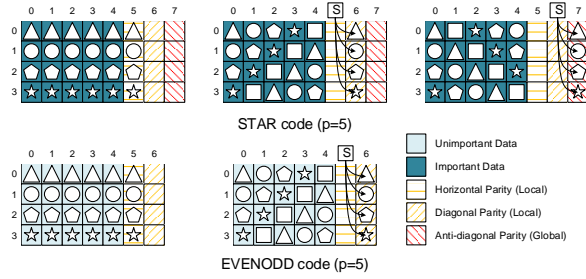
### 3.5 Case Study

This section describe the decoding methods in different cases of node failure based on *Structure 1* and *Structure 2*.

**3.5.1 Reconstruction cases in Structure 1.** Under *Structure 1*, important data and unimportant data are distributed in the same nodes, so they suffer the same risk of device failures. When  $r$  or less nodes fail, all the data can be reconstructed by local parity. When the number of failed nodes is over  $r$  but no more than 3, the Approximate Code conduct approximate recovery, in which the important data are recovered by global parity nodes and the damaged video data will be processed by the video recovery and fuzzy algorithm.



**Figure 6: The XOR-based Approximate Code (5, 2, 4), constructed based on EVENODD and STAR code. LP represent the local parity and GP represent the global parity.**



**Figure 7: EVENODD only consist horizontal and diagonal parity, while STAR add the anti-diagonal parity node based on EVENODD and can tolerate 3 device failures.**

**3.5.2 Reconstruction cases in Structure 2.** Under *Structure 2*, we consider important and unimportant data nodes separately.

For node level failure, the fault tolerance of *Structure 2* is exactly as *Structure 1*. When it comes to node level failures, *Structure 2* can tolerate more unimportant data node failures than *Structure 1*. The parity chain of important data does not pass through the unimportant data, so the damage of these nodes does not affect the recovery of important data.

Under the random probability model of nodes, the probability of failure of the Approximate Code under 4 node damage is

$$P_{failure} = \frac{C_{k+3}^4}{C_{(k+r)h+3-r}^4} \quad (3)$$

which is the 4 node failure situation in  $k$  important data nodes and 3 important parity nodes. For example, the failure situation of Approximate Code (4, 1, 4, *Structure 2*) probability is 0.478%.

### 3.6 Properties of Approximate Code

We analyze the nature of the Approximate Code from the following aspects, and the calculation method of the relevant indicators is given in Table 3.

- **Low Storage Overhead:** Approximate Code reduces storage overhead by approximating storage strategies. This property

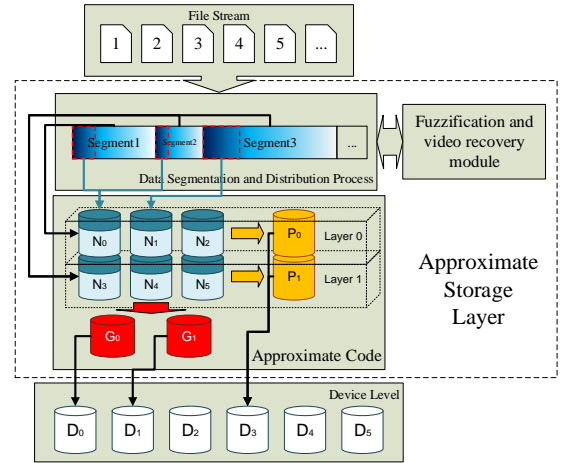
is more pronounced for data with a smaller proportion of important data.

- **Low Update Overhead:** When one node is updated, Approximate Code only needs to write  $r \times h$  local parity nodes and only  $3 - r$  global parity nodes, while typical 3DFTs EC methods should write  $3 \times h$  parity nodes.
- **High reliability for important data.** The Approximate Code guarantees 3DFTs for important data.
- **Flexibility.** The implementation of the Approximate Code can be based on RS, XOR or a mixture of the two.

**Table 3: Comparison of storage overhead and fault tolerance between Ap-Code (short for Approximate Code), RS and several XOR-based codes.**

EC	Storage Overhead	Fault Tolerance	Single Node Update Cost
RS( $k, r$ )	$(k + r)/k$	$r$	$r$
EVENODD( $k, 2$ )	$(k + 2)/k$	2	2
TIP-code/STAR( $k, 3$ )	$(k + r)/k$	3	3
Ap-Code ( $k, r, h$ )	$\frac{(k+r)h+3-r}{k \times h}$	$r$ to 3	$r + \frac{3-r}{h}$

## 4 IMPLEMENTATION



**Figure 8: Overview of Approximate Code implementation**

Compared with the traditional scheme that does not consider the meaning of the upper array data, the Approximate Code pays attention to the difference of the importance of the data, so an intermediate array between the upper array application and the underlying distributed storage system is necessary to preprocess the data. We call it the approximate storage array, which include 3 parts: data identification and allocation, encoding and decoding process of Approximate Code and the video fuzzification and recovery.

#### 4.1 Data Segmentation and Distribution module

The data segmentation module splits the video file stream into multiple data segments and automatically discriminates the important data. A natural approach is to follow the GOP segmentation. According to the ??, the GOP of the encoded video starts with an I frame, and all the data in that GOP depends on it for decoding, so we define it as important data and divide the GOP into I frames (important part) and other data. (unimportant part).

The data distribution module distributes important data and unimportant data in different blocks and records the location. This location information is also defined as important data. This module is also responsible for analyzing the proportion of important data and unimportant data in the data stream to select the most appropriate coding parameter configuration to ensure fault tolerance and high storage efficiency of important data.

#### 4.2 Approximate Code module

The Approximate Code module is responsible for encoding and decoding important and unimportant data and completely recovering the data within fault tolerance. For data that cannot be recovered, the Approximate Code module transmits it to the video recovery module for approximate recover videos.

Approximate Code module is also responsible for allocating blocks of data to ensure that blocks of each array are distributed to different nodes, while ensuring that global check blocks and data blocks are not on one node.

#### 4.3 Data Fuzzification and Video Recovery

In the approximate recovery mode, the remaining data and important data are transferred to this module. Using a series of methods described in Section ??, frames that suffer from frame loss will be recovered using the interpolation algorithm; the corrupted frames will be processed into fuzzy images and approximately recovered by superpixel techniques.

### 5 APPROXIMATE RECOVERY

In the approximate recovery mode, since unimportant data suffers unrecoverable loss, the recovery of damaged video files must be considered. Therefore, we evaluate the recovery of the video file in this section.

#### 5.1 Experimental Setup

**5.1.1 Coding Algorithm.** We evaluate the condition of the video of H.264 encoding [] under approximate recovery. Mainly because H.264 is an advanced video coding algorithm widely used at present, at the same time, the common advanced coding algorithms have similar coding processes.

**5.1.2 Data Set.** We choose 100 videos from xxx data set[]. All videos are stored in ".mp4" format at 1280\*720 HD resolution.

**5.1.3 Metrics.** We use the number of frames saved as an indicator to evaluate the ability of the approximate storage and normal scheme to preserve data in the event of an unrecoverable node failure.

We define that when the I frame of the video is corrupted, the entire GOP is marked as unavailable (lost).

Since the approximation code stores the I frame as important data, the integrity of the I frame is always guaranteed in our scheme.

**5.1.4 Simulation.** We store the video files on a single disk according to the approximate storage scheme and the common scheme, while traversing the damage of all the disks, from 1 to n-1, and counting the number of damaged p-frames and b-frames. We use this to get the number of P and B frames that can be completely saved in the case of m out of n disks.

#### 5.2 Results

The experimental results are shown in Figure 9, where  $n$  is 4,5,6 and 8. In a 8-node array, when 2 nodes are unrecoverable, the approximate recovery method protects nearly 1/4 P or B frames, while the common method lost more than half of these frames. In addition, the approximate storage scheme can guarantee the number of frames of the video close to the theoretical limit.

#### 5.3 Analysis

Our experiments prove that if data is stored indiscriminately, the loss of important data will seriously affect the security of all data, and our scheme will control the data loss near the theoretical limit, ensuring video recovery and fuzzification. feasibility.

#### 5.4 Frame Recovery ????

### 6 EVALUATION

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

#### 6.1 Evaluation Methodology

We choose the RS code to compare with several XOR-based code, including TIP-code, EVENODD, STAR, RAID5 and RAID6. Since 3DFTs are typical configurations, we set the number of check nodes for the RS to 3. Experiments as well as mathematical analysis are used to demonstrate the performance of Approximate Code.

**6.1.1 Metrics and Methods for Mathematical Analysis:** We use the **Storage Efficiency, Fault Tolerance and Write(update) Cost** defined in Section 3.6 as measure. We first analyze the impact of the settings of the three parameters on these metrics, then we compare these metrics with several EC methods.

**6.1.2 Metrics and Methods for Experiments:** We use **Encoding Time and Recovery Time** as the metrics in our experiments.

Our experiment includes the calculation of these two metrics and can mainly be divided into several cases. We measure the encoding time by calculating the time to generate all parity blocks, and we measure the recovery time by calculating the time to fully or approximately recover the failure nodes. Both of them should be divided into 3 situation: single, double and triple node, so we discuss them separately. Also, the encoding and recovery time consist of computation time, I/O time and transmission time in distributed file system, which should be considered separately.

The environment of our experiments are shown in Table 4. We set a Hadoop system with one NameNode and 4 DataNodes. text





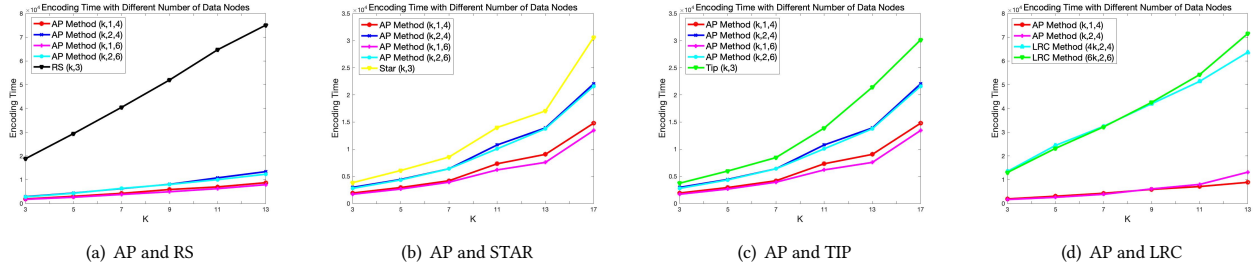


Figure 10: Encoding Time Comparison between AP method and other method

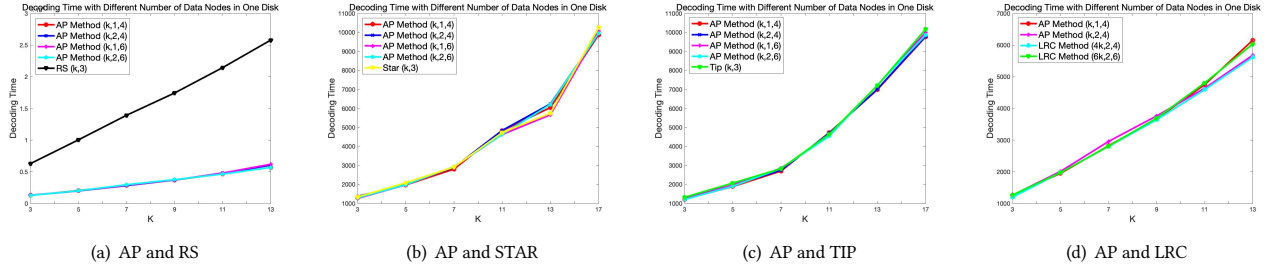


Figure 11: Decoding Time Comparison in One Disk between AP method and other method

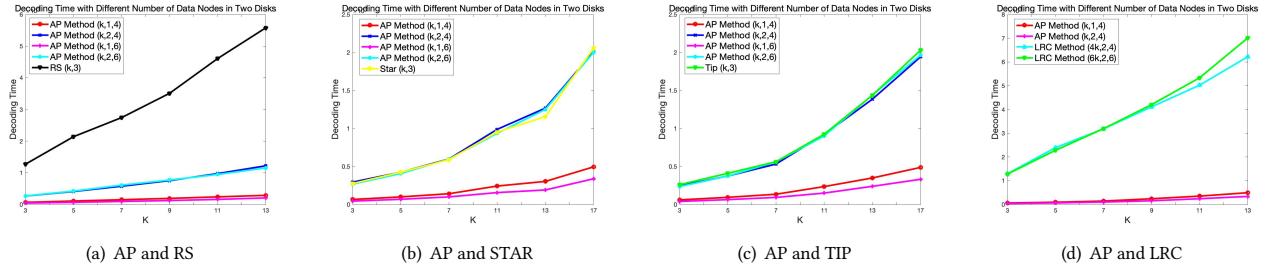


Figure 12: Decoding Time Comparison in Two Disk between AP method and other method

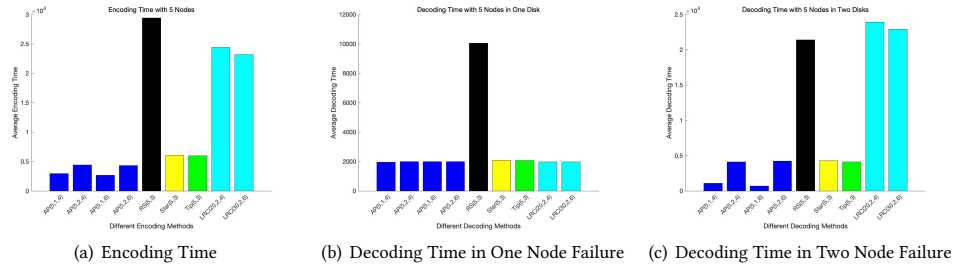


Figure 13: Comparison of several metrics between different methods when number of data nodes are 5

decoding time in two disks in comparison to LRC method. The rate of optimisation between AP code ( $k, 1, 4$ ) and LRC ( $4k, 2, 4$ ) is from 1971.5% (3 nodes' case) to 1265.5% (13 nodes' case); and the rate of optimisation between AP code ( $k, 1, 6$ ) and LRC ( $6k, 2, 6$ ) is from

2941.3% to 1970.7%. Considering the optimisation rate, AP code ( $k, 1, 6$ ) is the best among the four.



According to Figure xx, we see that RS (5, 3) has the longest decoding time among these methods, the LRC method is not suitable for decoding in two disks. AP code, here, largely reduces the decoding time in two disks, while the AP (5, 1, 6) keeps the shortest decoding time.

- Storage efficiency. . .
- Fault tolerance. . .
- Write overhead. . .

## 6.4 Encoding Performance

We use Approximate Code (4,2,4) and (5,1,6)...

## 6.5 Recovery Time

We use Approximate Code (4,2,4) and (5,1,6)...

## 6.6 Analysis

## 7 CONCLUSION

## ACKNOWLEDGMENTS

## REFERENCES

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] 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).
- [7] 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 complexity. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 136–147.