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/nnnnnnn.nnnnnnn

1 INTRODUCTION

Multimedia data consumes massive storage space in cloud storage systems, and this trend is exacerbated as applications demand higher resolution and frame rates. On YouTube, nearly 140,000 hours of video are played every minute and 400 hours of video are uploaded. Rapidly growing data imposes very high requirements of reliability and availability on large-scale storage systems as well as low cost.

Although multiple replicas can be used to ensure data availability and reliability, this method is too expensive and is only used to save hot data in practice. In contrast, the amount of cold data is far more than hot data, and erasure code (EC) schemes are ideal for storing such data. It provides lower storage overhead and write bandwidth than replication with the same fault tolerance. Currently, many cloud storage systems use erasure code to tolerate disk failures and ensure data availability, such as Windows [], Amazon AWS [] or Alibaba Cloud []. Typical erasure codes configuration use three-disk fault tolerant array (3DFTs). However, its overhead is still too high because the simultaneous damage of triple disks is relatively rare.

The recently proposed approximate storage strategy can significantly reduce the consumption of storage resources and energy. Common methods are to ensure the reliability of important data (marked as ID) while storing the unimportant data (marked as UD) on relatively unreliable media or reducing their error correction coding. Multimedia data is a typical application scenario for approximate storage because they can tolerate data corruption compared to other data. For example, video data records at least 20 frames per second, which makes it difficult for a typical user to perceive the loss of several frames. Also, some pixel errors in the image data do not affect the information of the entire picture. However, the direct application of approximate storage in a cloud storage system will result in the unimportant data being unacceptable volatile.

//////////////////////

Therefore, we propose Approximate Codes for multimedia data that reduce storage overhead by reducing the parity of data that is not sensitive to errors. In the scenario shown in Figure 5, the Approximate Codes are designed for systems composed of n disks where m disks are dedicated to coding. Other $s \times t$ sectors are encoded for important data thus raise its reliability. Approximate Codes ensure that the important data can tolerate $m + s$ device failures while all data can tolerate m device failures. When more than s disks fails, Approximate Code recovers the important data and then transfer the surviving data to the upper layer for recovery.

//////////////////////

With proper data distribution and algorithm design, the quality loss of video or image can be controlled within an acceptable range of applications, which leads to another important task in approximating storage, distinguishing data importance. This work is traditionally done by experienced programmers. Fortunately, multimedia data is commonly compressed and stored in encoded formats, which results in a certain portion of such data being more important than others. For example, in the progressive transform codec (PTC) compressed image, control and run-length bits are much more important than refinement bits. Therefore, this work can be done automatically by a system tailored to specific encodings.

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

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Our work contributions include:

(1) We propose Approximation Code that reduces storage overhead and improves the reliability and availability of important data with the approximate strategy.
(2) We prove the mathematical correctness of the Approximation Code.
(3) We perform a series of experiments and show that Approximate Code performs better than the traditional method in the full recovery mode, and the data loss in the approximate mode is acceptable.

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. The evaluation is presented in Section 6 and the conclusion of our work is in Section 7.

2 RELATED WORK AND OUR MOTIVATION

This section presents background on erasure codes, related video storage methods, approximate storage, and our motivation.

2.1 Existing Erasure Codes

Reed Solomon Code (RS code) is a kind of Maximum Distance Separable (MDS) Codes, which have the optimal storage efficiency. The encoding and decoding operations of RS code are based on Galois Field, which leads to a higher computational complexity comparing to XOR-based codes.

However, due to its high scalability, RS code has been widely applied in traditional cloud storage systems. In a RS code which is delegated by RS(k, r), $n = k + r$ denotes the total number of nodes participating in the erasure coding schema, k stands for the number of data nodes, and r is the number of parity nodes. Generally data is organized and encoded/decoded with the minimum coding unit block. RS(k, r) can tolerate at most r failures at the same time, and single node failure can be recovered from any k survivors. The encoding case of RS code and the failure tolerance processus are shown in Figure 1.

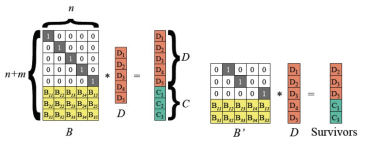


Figure 1: Encoding process of RS(5, 3)

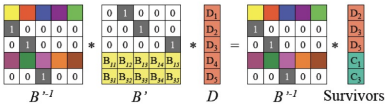


Figure 2: Recovery processing of RS(5, 3) in the case of 3 random failure

In our case, RS code could be applied. As the important data are uniformly-distributed in every node, the data in every node could be considered independent to each other. The principal of video storage divides the data into a small part of important frame and other subordinate frames. If a storage failure occurs in a node that stores largely the important frame, the data recovery may not be complete because the loss of the important frame could lead to a video quality degradation. In this way, this kind of uniformly-distribution would not allow the failure causing the data loss bigger than expected.

The data distribution assures the independency of nodes, the recovery of data could follow nearly the theoretical case. The recovery processus need the reversible matrix of original matrix. The calculation is shown in Figure 2.

Another type of erasure code worth mentioning is XOR based codes, which uses purely XOR operations. There are two different approaches to implement XOR codes. One is to use generator matrix[10] for encoding/decoding operations, and the other is done by using specific algorithm for each code[7]. In nowadays cloud storage systems, erasure codes for correcting two or three disk failures are widely used to ensure the reliability, and XOR-based codes show great advantages on encoding/decoding speed when the storage system is recovering from failures. XOR-based codes can be further categorized as XOR-based codes used in RAID-6 and ones to backup triple disk failure.

When it comes to RAID-6, Maximum Distance Separable (MDS) [8] [4] [2] [5] [1] [3] codes are frequently discussed. There are two types of MDS codes with different properties. One of them is called horizontal MDS codes, but it suffers from high write complexity and unbalanced I/O distribution. Accordingly, the other is called vertical MDS codes which has a common disadvantage of having resource-consuming partial strip write to continuous data.

A novel and efficient XOR-based RAID-6 code, named hybrid code(h-code) [13], is introduced a few years ago. This type of codes exploits the benefits of both horizontal and vertical MDS codes, therefore achieving optimization of partial stripe writes for RAID-6 in addition. H code can be applied to any array of (p+1) disks, where p is a prime number. The parities in H-Code are typical row parity and anti-diagonal parity. H-code does not have a specialized anti-diagonal parity strip, while it distributes the anti-diagonal parity elements among all the disks. Its horizontal parity ensures a partial stripe write to continuous data elements in a row share the same row parity chain, which achieves optimal partial stripe write performance. H code can be applied to any array of (p+1) disks, where p is a prime number. The analysis shows that H-Code achieves excellent performance in storage efficiency, encoding/decoding computational complexity and single write complexity.

2.2 Video Storage

For normal 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 and compressed before storage. Lossy compression is a common method that provides a much lower compression ratio than lossless compression while ensuring tolerable loss of video quality, and that is why we focus on such algorithms.

H.264 is one of the advanced algorithms for this type of work. This coding technique is widely used on platforms such as YouTube because it has higher compression ratio and lower complexity than its predecessor. For the HD video mentioned earlier, 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 does not depend on other frame data, which means it can be decoded independently of other frames.
(2) P frame: A frame holds the changes compared to the previous frame, thus saving much space by leaving out redundant information.
(3) B frame: A frame saves more space by utilizing the data of both the preceding and following frame.

In order to prevent the circumstance where a P or B frame references another distant frame, the concept of GOP is introduced. A GOP consists of multiple consecutive I,P and B frames which are independent of the frames in other GOPs. In other words, a P or B frame can only reference the ones inside the GOP which it belongs to, as shown in Figure 3.

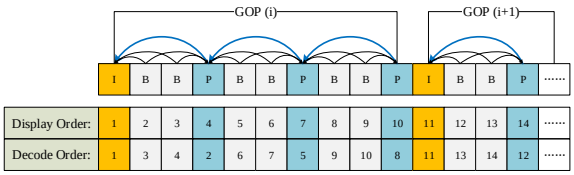


Figure 3: A sample of GOPs in H.264

2.2.1 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[12] based on recent advances in spatial transformers and composite perceptual losses as well as a context-aware Synthesis approach[9] have so far produced the new state-of-the-art results in terms of PSNR and middlebury benchmark respectively.

The methods are relied on the completeness of some video frame, which enhances the importance of the only intact type of frame data in commonly used H.264 standard: the I frame.

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. [11] 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 [6] 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.

In the storage of video data, as described in 2.2, the I frame is the key to decoding the entire GOP. An error in the I frame will cause a decoding error in the P frames and the B frames, and the data loss of the I frame will cause the entire GOP to fail. In contrast, the error or loss of a P frame has less impact, while the B frame is most tolerant of errors because no other frames depend on it.

Considering the vital role the I frame plays in the video coding, we therefore define I frame data as the critical data of a video file. Although some part of P frames may play a relatively important role in the decoding process of a video, it's importance can not exceed that of the I frames.

Table 1: 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
	unimportant		medium	high

2.4 Our Motivation

Based on Table 1, 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 the design of Approximate Code and its properties through a few simple examples. For convenience of description and without loss of generalizability, we use fewer data blocks (resulting in greater storage overhead). A more optimized parameter selection scheme for practical applications will be introduced in Section 6.

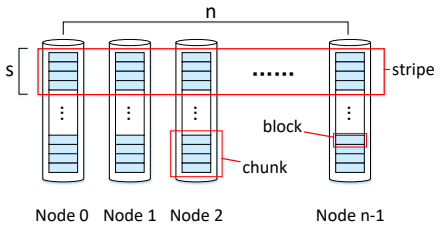


Figure 4: A sample of chunks, stripes and blocks

3.1 Design of Approximate Code

The construction of the Approximate Code is determined by 4 parameters k, r, h and s. We use Figure 4 to illustrate the term *chunk*, *block*, *stripe* and *layer* we use in this paper. In a system of n nodes, k of them are data nodes and $r = n - k$ nodes are for parity. Each node can be divided into multiple blocks. We focus on the s blocks of the same logical position of each node, and we treat these s nodes as a *chunk*. The n chunks constitute a *stripe*. Approximate Code are designed for h stripes, where each stripe can be in the same n-node system, or on different n-node systems. For the latter situation, we specifically use *layer* instead of stripe, since these two cases will result in different performance and reliability.

For s blocks in each chunk, we assume that important data occupies 1 block, so the ratio of important data to the whole chunk is $1/s$. In addition, there are another $3 - r$ global chunks, which are calculated only by the important data blocks. It should be point out that the global parity chunks should be placed in several different nodes and should be different from the data nodes. The number of important data block is $h \times k$, and they generate $h \times g$ global parity blocks. To sum up, the number of blocks involved in the approximate code is

$$((k + r)s + 3 - r) \times h$$

Approximate Code guarantees that for each stripe, 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 (4,1,4,3) can be illustrated by Figure 5, where N_i are data nodes, P_i are local parity nodes and G_i are global parity chunks. This example includes 22 nodes.

As mentioned before, if it is considered that the chunks from the same column belong to the same node, this example includes 4 stripes with 7 nodes; otherwise, it includes 4 layers with 22 nodes.

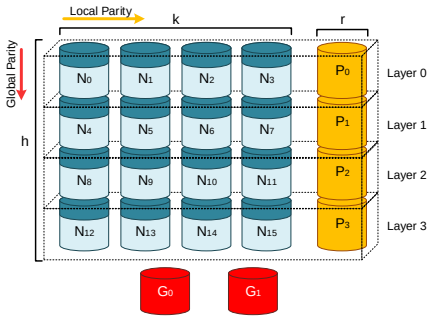


Figure 5: A sample of Approximate Codes $(4, 1, 4, 3)$ with $h = 4$ layers of nodes, where each layer contains $k = 4$ data nodes (marked as water green) and $r = 1$ parity nodes(marked as orange). The important data (marked as blue) occupies $1/5$ of the total data node, and generate 2 global parity chunks (marked as red).

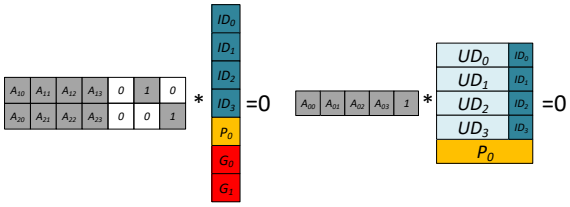


Figure 6: The generate matrix of RS-based Approximate Codes $(4, 1, 4, 3)$, where ID_i and UD_i represent the important and unimportant data in N_i. G_0 and G_1 are the parity blocks corresponding to important data blocks.

3.2 RS-based Approximate Code

We still use Figure 5 to illustrate RS-based Approximate Code $(4, 1, 4, 3)$ and its properties. The local parity chunks are encoded by RS(4,1), and the important data generate another 2 global parity

chunks. It is easy to find appropriate coefficient to ensure important data forms the form of RS(4,3) and tolerate any 3 chunk failure. Figure 6 shows the generate matrix of Layer 0 in Figure 5.

3.3 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 tipical cases of the construction of XOR-based Approximate Code (5, 1, 6, 6) and (4, 2, 4, 4).

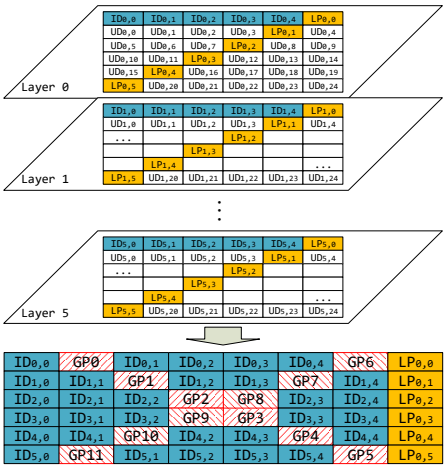


Figure 7: The XOR-based Approximate Code (5, 1, 6, 6), constructed based on RAID5 and TIP-code.

3.3.1 RAID5-TIP Approximate Code. Figure 7 is a tipical construction of Approximate Code (5, 1, 6, 6) based on RAID5 and TIP-code [14]. RAID5 tolerate one node failure while TIP-code is an XOR-based 3DFTs code. In each layer, local parity blocks provide horizontal XOR parity, and between layers, global parity provide diagonal parity and anti-diagonal parity.

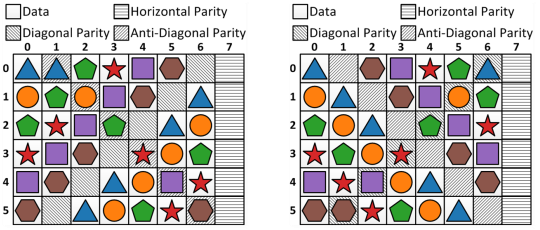
The local (horizontal) parity elements are calculated by the following encoding equations.

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

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

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 8 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.

3.3.2 EVENODD-STAR Approximate Code. Figure 9 shows the construction of Approximate Code (4, 2, 4, 4) based on EVENODD [1] and STAR [7]. EVENODD is a tipical RAID6 erasure code scheme,



(a) Diagonal parity GP_0 to GP_5, (e.g., $GP_0 = ID_{0,0} \oplus ID_{1,4} \oplus ID_{2,3} \oplus ID_{4,2} \oplus ID_{5,1}$

(b) Anti-diagonal parity GP_6 to GP_{11}, (e.g., $GP_7 = ID_{0,0} \oplus ID_{1,1} \oplus ID_{2,2} \oplus ID_{4,3} \oplus ID_{5,4}$)

Figure 8: Encoding pf TIP-code

while STAR is a 3DFTs extension scheme of EVENODD. In this case, important data blocks are distributed in Layer 0, while unimportant data blocks occupy the rest layers. Because the STAR code is constructed just by adding a reverse check chain to the EVENODD code. In this case, we design a global check block to store this check chain, so 3DFTs can be guaranteed.

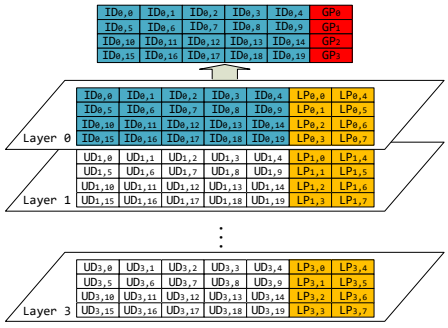


Figure 9: The XOR-based Approximate Code (4, 2, 4, 4), constructed based on EVENODD and STAR code. *LP* represent the local parity and *GP* represent the global parity.

Table 2: Comparison of storage overhead and fault tolerance between Approximate Code, RS and several XOR-based codes.

EC	Storage Overhead	Fault Tolerance
RS(k, r)	$(k + r)/k$	r
RAID-5$(k, 1)$	$(k + 1)/k$	1
EVENODD$(k, 2)$	$(k + 2)/k$	2
TIP-code/STAR$(k, 3)$	$(k + r)/k$	3
Approximate Code (k, r, h, s)	$\frac{(k+r)s+3-r}{k*s}$	r to 3

3.4 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 2.

- Low cost. Approximate code reduces storage overhead by approximating storage strategies. This property is more pronounced for data with a smaller proportion of important data.
- 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.

4 IMPLEMENTATION

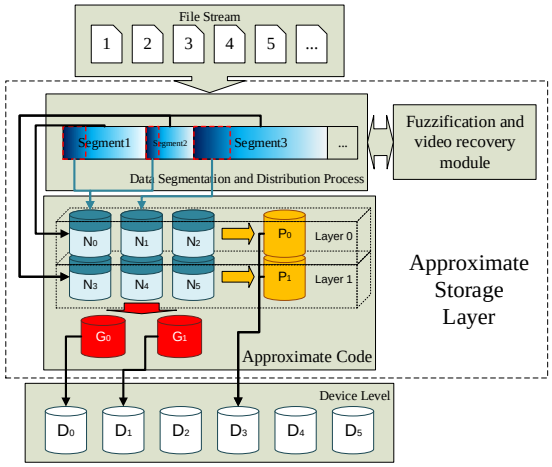


Figure 10: Overview of Approximate Code implementation

Compared with the traditional scheme that does not consider the meaning of the upper layer data, the Approximate Code pays attention to the difference of the importance of the data, so an intermediate layer between the upper layer application and the underlying distributed storage system is necessary to preprocess the data. We call it the approximate storage layer, 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 Process

The identification of data importance can be specified by the upper application as well as automatically determined by the approximate storage layer. We focus on the latter and take video file as an example to introduce our method of automatically distinguishing importance.

For video data encoded by H.264 or similar format, we define I frames as important data, while P frames and B frames as unimportant data based on the analysis in section 2.2. In practical, video data is rarely stored in the original form of H.264 streams (".264" files), but is usually storaged in an file such as ".mp4" files containing information such as audio. We transcode video files into raw video streams and other data, and we define these non-video data as important because they contain information that the video can't provide and only take up a small amount of space. The feasibility of this definition will be confirmed in 6.

4.2 Data Distribution and Reorganization

Fortunately, in an H.264 stream, each GOP begins with an I frame followed by a series of P and B frames. Therefore, we store data in units of GOPs. Our main purpose is to store I frames and other frames separately. For non-video data, we distribute it into multiple GOPs and treat it as a special part of the I frame. In the following description, we no longer consider such data especially and simply refer to them as I-frames. We define ω as the the important data ratio, which is the size of I-frames divided by the entire GOP size. We also define ω_{act} as the actual important rate the code can provide by equation 3.

$$\omega_{act} = \frac{D_I}{D_I + D_M} = \frac{(n-m)\times t}{(n-m)\times r - s \times t} \tag{3}$$

Algorithm 1 and 2 shows the data distribution and reorganization methods.

Algorithm 1 Data Distribution Algorithm

Input: n, m and s from the settings of approximate storage layer. Get the type of erasure code.
Output: D_I, D_M, Q and G.

```
while true do
    Divide data into several segments (several GOPs for video data);
    Calculate ω of each GOP, and mark the highest one as ω_max;
    Adjust t to find the closest ω_act to ω_max;
    repeat
        Divide each GOP into two parts: ω_act and 1 − ω_act;
        Store the former in D_I, the latter in D_M;
    until All GOPs are classified;
end while
```

Algorithm 2 Data Reorganization Algorithm

Input: D_I and D_M;
Output: A stripe of video data;

```
Find the parameter (n, m, r, s, t) and calculate ω_act.
repeat
    Read an I frame from D_I and record its length as l.
    Read l × (1−ω_act)/ω_act in D_M and combine two parts.
until All blocks are read.
```

The data distribution scheme is shown in the figure 11. We present ω_i as the ω of Data(i), and $\omega_{max} = \omega_2$. For example, Data 3 is represented by blue, and its key segment (10%) and part of unimportant segment (10%) are settled in D_I. The main idea of our data distribution method is to guarantee that each GOP has the same proportion of storage in D_I and D_M. This method improves

flexibility because it is not necessary to maintain a map that marks the location of each GOP, which makes it is easy to add or delete data at any time. Meanwhile, the method is ideal for streaming video data generated in real time by applications such as monitoring.

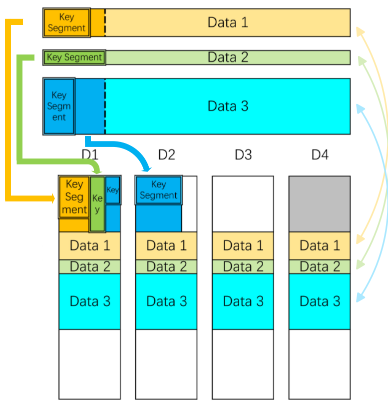


Figure 11: A sample of data distribution method, where the gray block in the upper right corner is the Q area. Here for each data segment, $\omega_1 = 15\%$, $\omega_2 = 20\%$ and $\omega_3 = 10\%$, so the $\omega_{max} = 20\%$.

In addition, our approach can be applied to a variety of other data types. In fact, encoded multimedia data mostly has the property of coexisting important data and non-essential data, such as PTC encoding used in image storage[][].

5 EXPERIMENTAL SETUP

5.1 Evaluation methodology

In our evaluation, we first compare the Approximate Code with the RS code and some XOR-based codes to demonstrate the performance benefits of our solution. Since 3DFTs is a typical erasure code configuration, we use it as a baseline and set the minimum fault tolerance of important data to 3 ($m + s \geqslant 3$). We will then demonstrate the quality loss of multimedia data under the traditional and approximate recovery models of severe disk failure to assess the benefits of our approximate storage scheme. We use mathematical analysis and experiments to prove the effectiveness of the Approximate Code.

5.1.1 Erasure Codes in Our Comparisons. text

5.1.2 Platforms and Configurations. text

5.1.3 Metrics. text

5.1.4 Parameters and Assumption in Our Evaluation. text

6 EVALUATION

This section introduces a series of experiments we have conducted to verify the efficiendy of the Approximate Code.

6.1 Results

text text

Table 3: Summary on Various Erasure Codes

Name	Fault Tolerance	Storage Overhead	Scalibility	Recovery Cost	Computational Complexity
RS(k, m) Code	any m disks	m disks	high	high	high
MSR(k, m) Code	any m disks	m disks	medium	low	very high
Raid 6	2	2	low	high	low
SD Code(m, s)	any m disks and s sectors	m disks and s sectors	low	low	medium
Approximate Code(n, m, s, t) (Important Data)	any $m + s$ disks	m disks and s sectors	high	high	medium
Approximate Code(n, m, s, t) (unimportant Data)	any m disks	m disks and s sectors	high	high	high

Code Config	Storage Overhead	FT (Imp)	FT (unimportant)	Important Rate
(6,2,4,1,2)	1.600	3	2	0.200
(8,2,6,1,1)	1.371	3	2	0.143
(10,2,8,1,1)	1.270	3	2	0.111
(11,2,9,1,1)	1.238	3	2	0.100
(11,3,7,1,1)	1.400	4	3	0.127
(13,3,10,1,2)	1.327	4	3	0.184
(8,2,6,1,2)	1.412	3	2	0.294
(8,2,4,1,2)	1.455	3	2	0.455
(10,2,6,2,2)	1.364	4	2	0.273
(11,3,8,2,2)	1.467	5	3	0.200

text text

6.2 Analysis

Illustrate why Approximate Code achieve high reliability with low cost text

7 CONCLUSION

text text

text text text text text text text text text text text text text text text text text text text text text text

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] Mario Blaum and Ron M Roth. 1999. On lowest density MDS codes. *IEEE Transactions on Information Theory* 45, 1 (1999), 46–59.

[3] Johannes Bloemer, Malik Kalfane, Richard Karp, Marek Karpinski, Michael Luby, and David Zuckerman. 1995. An XOR-based erasure-resilient coding scheme. (1995).

[4] Yuval Cassuto and Jehoshua Bruck. 2009. Cyclic lowest density MDS array codes. *IEEE Transactions on Information Theory* 55, 4 (2009), 1721–1729.

[5] 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 *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*. USENIX Association Berkeley, CA, USA, 1–14.

[6] 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.

[7] 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.

[8] Chao Jin, Hong Jiang, Dan Feng, and Lei Tian. 2009. P-Code: A new RAID-6 code with optimal properties. In *Proceedings of the 23rd international conference on Supercomputing*. ACM, 360–369.

[9] 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.

[10] James S Plank and Michael G Thomason. 2004. A practical analysis of low-density parity-check erasure codes for wide-area storage applications. In *International Conference on Dependable Systems and Networks, 2004*. IEEE, 115–124.

[11] 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.

[12] 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).

[13] 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

[14] 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.