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, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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.

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

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.

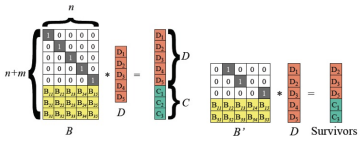


Figure 1: Encoding process of RS(5, 3)

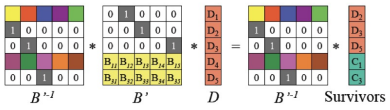


Figure 2: Recovery processing of RS(5, 3) in the case of 3 random failure

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.

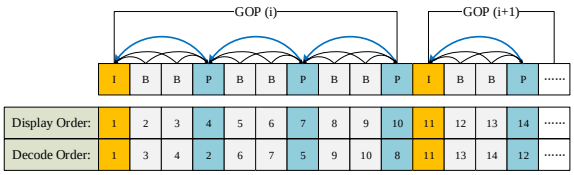


Figure 3: A sample of GOPs in H.264

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.

The construction of the Approximate Code is determined by 4 parameters k, r, h and s. First, we use Figure 4 to illustrate the term *chunk*, *block* and *stripe* we use in this paper. In a n-node system, 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 is in the same n-node system.

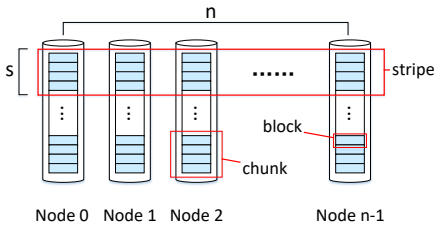


Figure 4: A sample of chunks, stripes and blocks

3.1 Design of Approximate Code

Approximate Code distribute important and unimportant data separately and mainly in two structure. As shown in Figure 5, in *Structure 1*, important data occupies 1 of s blocks in each chunk, and in *Structure 2*, it occupies 1 of all h data stripes. So, in *Structure 1*, important data accounts for $1/s$ of all data, and $1/h$ in *Structure 2*. In each stripe, r chunks are local parity chunks, which is encoded by the rest k data chunks in the stripe. 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 different nodes and should be different from the data nodes.

The number of important data block is $h \times k$ in *Structure 1* and $s \times k$ in *Structure 2*, so they generate $h \times (3-r))$ or $s \times (3-r))$ global parity blocks. To sum up, the number of blocks involved in the approximate code is

- Structure 1: $(k+r)sh + (3-r)h$
- Structure 2: $(k+r)sh + (3-r)s$

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,*Structure 1*) and Approximate (3,1,3,3,*Structure 2*) can be illustrated by Figure 5,

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 stripes with 22 nodes.

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

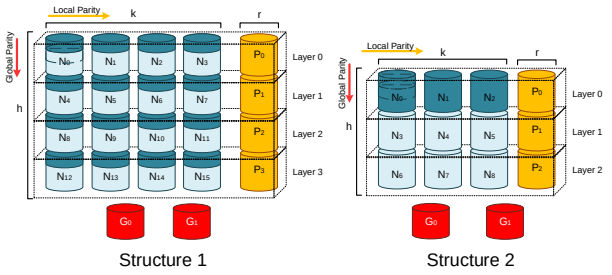


Figure 5: A sample of Approximate Codes $(4,1,4,3)$ in *Structure 1*, and Approximate Code $(3,1,3,3)$ in *Structure 2*. Data chunks are marked as N_i, important data are marked as water green, local parity chunks are marked as orange and global parity chunks are marked as red.

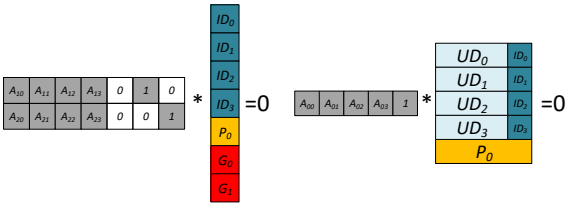


Figure 6: The generate matrix of RS-based Approximate Codes $(4,1,4,3)$, where ID_i and UD_i represent the important and unimportant data in N_i. G_0 and G_1 are the parity blocks corresponding to important data blocks.

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 stripe 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).

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 stripe, local parity blocks provide horizontal XOR parity, and between stripes, global parity provide diagonal parity and anti-diagonal parity.

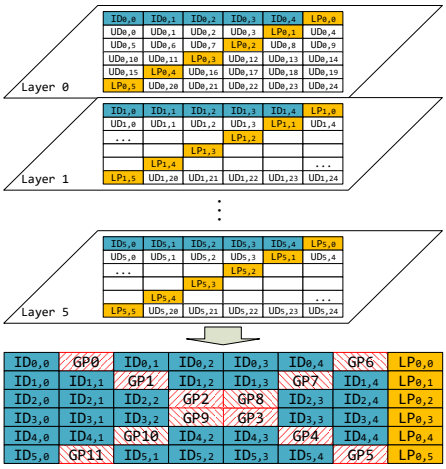


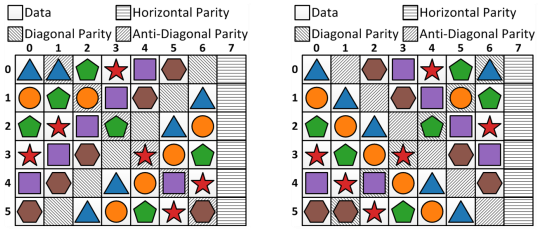
Figure 7: The XOR-based Approximate Code $(5, 1, 6, 6)$, constructed based on RAID5 and TIP-code.

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.



(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

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, while STAR is a 3DFTs extension scheme of EVENODD. In this case, important data blocks are distributed in stripe 0, while unimportant data blocks occupy the rest stripes. 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.

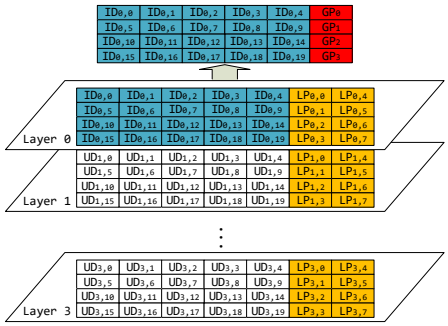


Figure 9: The XOR-based Approximate Code $(4, 2, 4, 4)$, constructed based on EVENODD and STAR code. *LP* represent the local parity and *GP* represent the global parity.

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

Compared with the traditional scheme that does not consider the meaning of the upper stripe data, the Approximate Code pays attention to the difference of the importance of the data, so an intermediate stripe between the upper stripe application and the

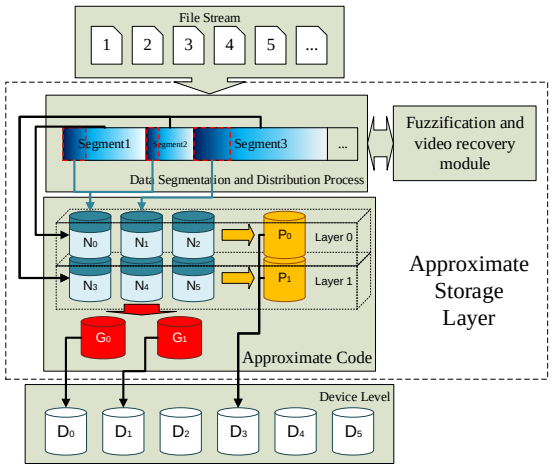


Figure 10: Overview of Approximate Code implementation

underlying distributed storage system is necessary to preprocess the data. We call it the approximate storage stripe, 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 2.2, 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 parts) 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 implements local and global encoding of $k*h$ data chunks and is responsible for reconstructing the wrong data chunk. When no more than r chunks are corrupted, Approximation Code uses the local check block to recover all data, both important and minor. When there is no more than 3 loss of important data, 2 forms Form 1: Important data and unimportant data are distributed at the same node. When there are no more than r blocks lost in a single stripe, the local check block completely recovers all data. When more than r but no more than 3 blocks are lost in a single stripe, the global check block recovers all important data, and the surviving unimportant data block and important data blocks are sent to the video recovery module. Form 2: The important data is distributed in a stripe, and the minor number is distributed on H-1 stripes. . . Important data stripe loses no more than 3 blocks and is fully recovered. The unimportant data stripe loses no more than r blocks and is fully recovered. The unimportant data stripe loses more than r, and the video is restored. The approximate code module is also responsible for allocating data blocks to ensure that the chunks of each stripe are distributed to different nodes, while ensuring that the global check chunk and the data chunk are not on one node.

4.3 Data Fuzzification and Video Recovery

In the approximate recovery mode, the recoverable and remaining data is passed to this module, and the blurred video is recovered through various video recovery techniques, and the upper application is notified. Through the content of the second chapter, we can approximate the data recovery. . .

5 APPROXIMATE STORAGE ANALYSIS

In this chapter, we mainly discuss the number of video frames saved in Mode 1 when the unimportant data corruption is greater than the fault tolerance of the local calibration and the important data can be fully recovered. This proves that if the video data is not distinguished, in a typical This compares the ability of the approximation code to secure video data.

Indicator: number of frames restored Test situation: Test set:

Our experiments analyzed a series of videos. 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.

The experimental results are shown in Figure x. The approximate storage scheme can guarantee the number of frames of the video close to the theoretical limit.

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.

6 EVALUATION

Our experiment is mainly divided into two parts: erasure code and approximate storage analysis. The former mainly analyzes the performance of the approximate code and compares our scheme with the typical erasure code configuration; the latter mainly analyzes the data protection capability of the approximate code and other schemes in the case of data loss.

In this section, we use mathematical analysis and a series of experiments to demonstrate the performance of the approximate code. Evaluation method: We choose RS code, TIP-code, EVENODD, STAR, RAID5 for comparison.

Mathematical analysis: We use the storage efficiency, fault tolerance and reconstruction cost defined above as a measure.

In short, for each stripe, the unimportant data can tolerate any r node failures, while important data can tolerate any three node failures. However, the actual scene will be more complicated: For mode one. . .

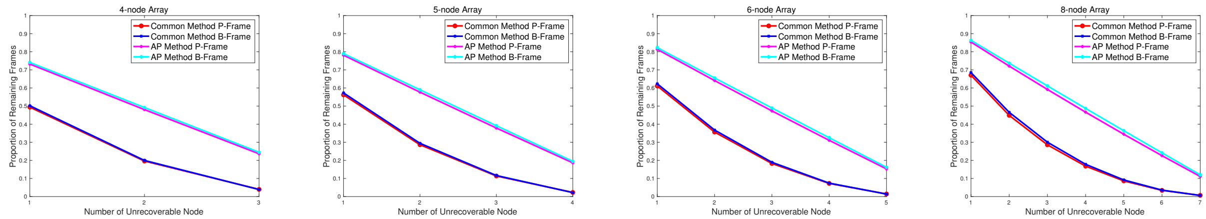


Figure 11: Comparison of the data retention between approximate storage and common methods when node is unrecoverable.

For fault tolerance, Mode 1 and Mode 2 will be different. Not only that, Mode 1 provides minimal fault tolerance, while Mode 2 provides near-code maximum fault tolerance.

Obviously, for mode one, the approximate code can tolerate any r node failures, and the fault tolerance of important data is 3. For mode two, the approximate code provides an arbitrary level of r-node failure, and only if

For mode two, if all chunks are distributed across different nodes, the approximate code provides the greatest degree of reliability. That is,

We calculated the recovery efficiency and reconstruction cost of the approximate code in the case of single-node, two-node and three-node failures. Since the approximate code can be constructed in many forms, we mainly focus on it. . . . . .

7 CONCLUSION

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.