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

Huayi Jin

ABSTRACT

Multimedia data generated by autonomous driving, media industry and security monitoring is often stored in cloud storage systems and occupies a large amount of space. Meanwhile, to ensure the data reliability, distributed file systems often use erasure codes to backup data. However, the commonly used triple disk failure tolerant arrays (3DFTS) erasure code scheme is expensive, not only because simultaneous damage of multiple disks is relatively rare, but also due to its ignorance of redundant information inside the data, resulting in multiple complete parity disks being excessive. On the other hand, the recently proposed approximate storage scheme can effectively reduce storage costs, but at the cost of sacrificing the reliability of some data.

In this article, we propose Approximate Code for multimedia applications, which is an erasure code using an approximation strategy. Approximate Code aims to ensure different reliability of important and minor data by means of erasure coding, thereby reducing storage overhead. It provides complete recovery when fewer disks fail, and ensures approximate recovery (recover most data) in the event of multiple disk failures. To demonstrate the effectiveness of Approximate Code, we conduct several experiments in Hadoop and Alibaba Cloud systems. The results show that compared with the typical high-reliability erasure code schemes, Approximate Code significantly reduces reduce the storage overhead and provide higher reliability of important data.

KEYWORDS

Erasure Codes, Approximate Storage, Multimedia, Cloud Storage

ACM Reference Format:

Huayi Jin. 2019. Approximate Code: A Cost-Effective Erasure Code 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, 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 and is excessive because 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 while storing the minor data 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 minor 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 3, 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 5 and the conclusion of our work is in Section 6.

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

Here we give some introduction to existing erasure codes Specifically, we need to illustrate the existing erasure codes for video applications in detail. Summarize the existing erasure codes in a table (assume Table 1), and illustrate them whether they satisfy the previous requirements in Section II.A

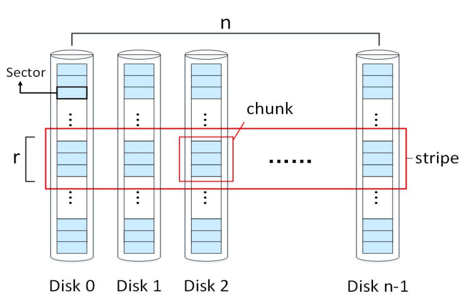


Figure 1: A sample of chunks and sectors

text text

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 2.

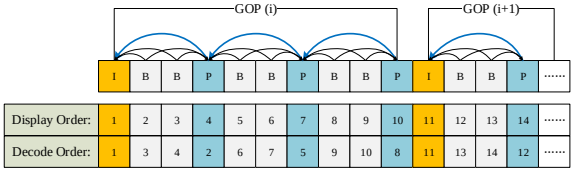


Figure 2: A sample of GOPs in H.264

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[3] based on recent advances in spatial transformers and composite perceptual losses as well as a context-aware Synthesis approach[2] 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./refsampson2014approximate 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 [1] 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, itâĂŹs importance can not exceed that of the I frames.

Table 1: Comparison of fault tolerance and storage overhead between approximate storage, RS and Approximate Code

Scheme	Storage Overhead	FT (Imp)	FT (Minor)
Approximate Storage	very low	some bits	-
RS(n-m,m)	$\frac{n}{n-m}$	m	m
Approximate Code(n, m, r, s, t)	$\frac{n \times r}{(n-m) \times r - s \times t}$	$m + s$	m

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 5.

3.1 Design of Approximate Code

We use Figure 3 to illustrate the construction of Approximate Code. In the n chunks of each stripe, m ones are for coding. In the remaining $n-m$ chunks, $s \times t$ additional sectors are for coding important data, where s is the number of the columns and t is the number of the rows. For convenience, we label $h = n - m - s$ as the number of columns of important data that is storaged in $h \times t$ blocks in our assumption.

Based on the above definition, our design has 5 configurable parameters (n, m, r, s, t) that uniquely determine the construction of Approximate Code. Figure **??** shows an example of Approximate Codes with $n = 7$, $m = 2$, $r = 5$, $s = 2$ and $t = 2$, where we label the data disks with $d_{i,j}$, the important data parity sectors with $q_{i,j}$ and the minor parity sectors with $p_{i,j}$.

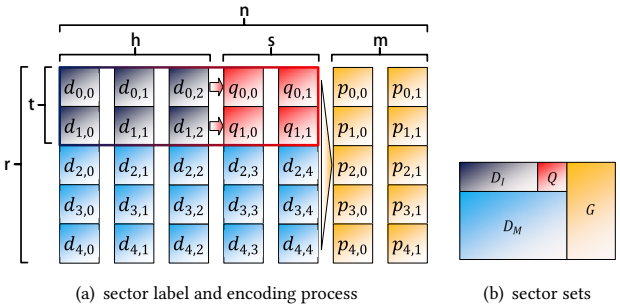
We then define the area of sectors as follows:

- $D_I = \{d_{i,j} | 0 \leqslant i < t, 0 \leqslant j < h\}$ important data sector zone.
- $D_M = \{d_{i,j} | t \leqslant i < r, 0 \leqslant j < n-m\}$ minor data sector zone.
- $Q = \{q_{i,j} | 0 \leqslant i < t, 0 \leqslant j < s\}$ important parity sector zone.
- $G = \{p_{i,j} | 0 \leqslant i < r, 0 \leqslant j < m\}$ global parity sector zone.

3.2 Encoding and Decoding Process

We also use Figure 3 to illustrate the basic idea of encoding and decoding process of Approximate Code.

The encoding process consists of two phases: the important coding phase (*I-Phase*) and the global coding phase (*G-Phase*). The *I-Phase* is expressed as two red arrows in the blod box, where Q is generated to verify D_I, and the *G-Phase* is expressed as a yellow triangle arrow, where G is generated to verify D_I, D_M and Q.



(a) sector label and encoding process (b) sector sets

Figure 3: A sample of Approximate Codes $(7, 2, 5, 2, 2)$ with 7 chunks where 2 of them are global parity chunks (orange zone G) and each chunk has 5 sectors. In this example, there are 6 sectors (dark blue zone D_I) for important data, 15 sectors (light blue zone D_M) for minor data and 4 sectors (red zone Q) encode important data.

The decoding process provides two modes: the approximate recovery mode and the full recovery mode. When no more than m divices fail, Approximate Code guarantees full recovery of lost data, and if the number of failed devices is larger than m but no more than $m + s$, Approximate Code guarantees full recovery of lost important data, which require D_I, Q and part of G for joint recovery.

It should be noted that in the calculation of G in *G-Phase* and in the decoding process in full recovery mode, we consider D_I, D_M and Q as the same, and the minimum coding unit is chunk. That is, when we do not distinguish the importance of the data($s, t = 0$), the Approximate Code is a typical m disk redundancy code.

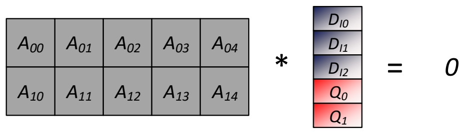
In general, Approximate Code performs extra parities on important data. Since our design guarantees the important data, the important parity and the minor data blocks completely fill $n - m$ chunks, we can construct the remaining m parity chunks in several ways. Therefore, EC methods such as RS-based, XOR-based, MSR, MDS or PMDS codes can be used to construct Approximate Code.

We then introduce two typical Approximate Code implementation schemes: RS-based and XOR-based Approximate Code. Different code selection of *I-Phase* and *G-Phase* will cause different effect.

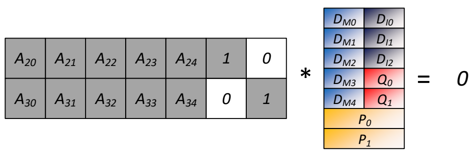
3.3 RS-based Approximate Code

Without loss of generality, we use Figure 4 to introduce the construction method of RS-based Approximate Code with parameter $(7, 2, 5, 2, 2)$.

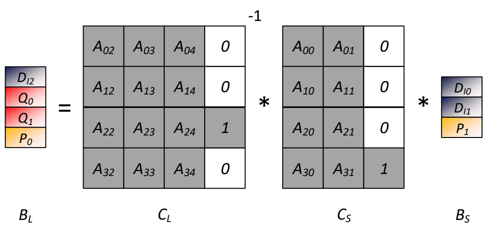
3.3.1 Encoding and Decoding Process. In *I-Phase*, we encode 3 groups of important data sectors D_{I0} to D_{I2} and generate 2 groups of parity sectors Q_0 and Q_1. The calculation of $q_{i,j}$ are defined by equation (1) and shown in Figure 4(a), where A_k is the coefficient in Galois Field (GF). In fact, the encoding process of Q_0 and Q_1 should be considered as a special decoding case that Q_0 and Q_1 have lost.



(a) I-Phase



(b) G-Phase



(c) The decoding process of important data. B_L represents the lost data vector and B_S represents the surviving data vector. C_L and C_S are the coefficient matrices of B_L and B_S respectively.

Figure 4: Generation Matrix of RS-based Approximate Code. D_{I0} to D_{I2} (dark blue) are important data sectors. D_{M0} to D_{M4} (light blue) are minor data sectors. Q_0 and Q_1 are important parity sectors (red). P_0 and P_1 are global parity sectors (orange).

$$\{\sum_{j=0}^{h-1} A_{k,j} d_{i,j} + \sum_{j=0}^{s-1} A_{k,j+h} q_{i,j} = 0 | 0 \leqslant i < t, 0 \leqslant k < s\} \quad (1)$$

In *G-Phase*, we use RS(5,2) to generate 2 global parity chunks labeled with $p_{i,j}$ from 5 chunks consist of data sectors and important parity sectors. The calculation of $p_{i,j}$ are defined by equation (2) and (3).

$$\{\sum_{j=0}^{h-1} A_{k,j} d_{i,j} + \sum_{j=0}^{s-1} A_{k,j+h} q_{i,j} = p_{i,k-s} | 0 \leqslant i < t, s \leqslant k < s+m\} \quad (2)$$

$$\{\sum_{j=0}^{h+s-1} A_{k,j} d_{i,j} = p_{i,k-s} | t \leqslant i < r, s \leqslant k < s+m\} \quad (3)$$

In approximate recovery mode, assume that any k chunks fail ($m < k \leqslant m + s$). We reorganized important data blocks, important parity blocks, and global parity blocks into surviving blocks set B_S and lost blocks set B_L. We label their corresponding coefficient matrices as C_S and C_L, as shown in Figure 4(c). We use the following equation to fully recover lost blocks.

$$B_L = (C_L)^{-1} \times C_S \times B_S$$

Furthermore, the encoding process of Q_0, Q_1, P_0 and P_1 of important data sectors are also calcuated by this equation.

3.3.2 Proof of Correctness. The guarantee of correctness is based on the choice of coefficients. Here we define the coefficient matrix as Vandermonde matrix.

The correctness of full recovery mode of RS-based Approximate Code is obvious since it is the same as the decoding process of RS(5,2), as shown in Figure 4(b).

In approximate recovery mode, the key to ensuring that important data can tolerate any $m + s$ block failures is to prove that any $m + s$ columns in coefficient matrix is reversible. According to the number of global parity block losses (from 0 to m), the column combination of the coefficient matrix can be divided into $m + 1$ cases. When no global parity sectors fail, the coefficient matrix must be full rank since it is a Vandermonde matrix. When some of global parity sectors fail, as C_L in Figure 4(c), we calculate the coefficient determinant corresponding to the lost blocks. By expanding the coefficient determinant by the last column, we can get a Vandermonde determinant lacking a line, which is still full rank.

3.4 XOR-based Approximate Code

We take RAID-6 code

text text

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

3.5 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 1.

- 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 the fault tolerance of the important data $m + s$.
- Flexibility. The implementation of the Approximate Code can be based on RS, XOR or a mixture of the two; at the same time, the construction of the Approximate Code can also be used for encoding such as LRC or MSR.

4 IMPLEMENTATION

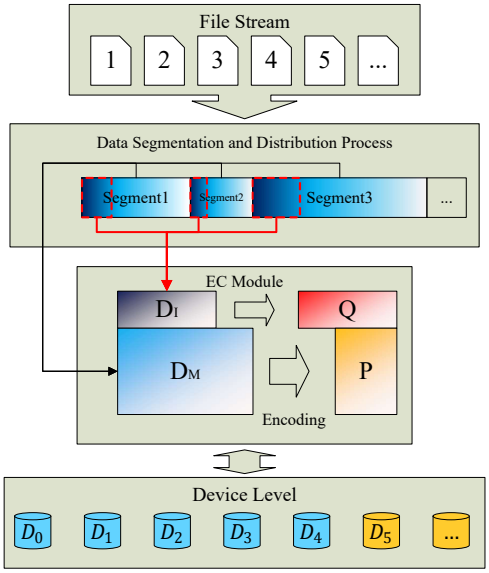


Figure 5: Overview of Approximate Code implementation

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. The approximate storage layer is responsible for the identification and allocation of data. It controls the EC module and implements the encoding and decoding work of the Approximate Code.

4.1 Data Identification

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 minor 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 5.

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 4.

$$\omega_{act} = \frac{D_I}{D_I + D_M} = \frac{(n-m) \times t}{(n-m) \times r - s \times t} \tag{4}$$

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
```

The data distribution scheme is shown in the figure 6. We present ω_i as the ω of Data(i), and $\omega_{max} = \omega_2$. For example, Data 3 is represnted by blue, and its key segment (10%) and part of minor 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

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 $l \times \frac{1-\omega_{act}}{\omega_{act}}$ in D_M and combine two parts.
until All blocks are read.

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.

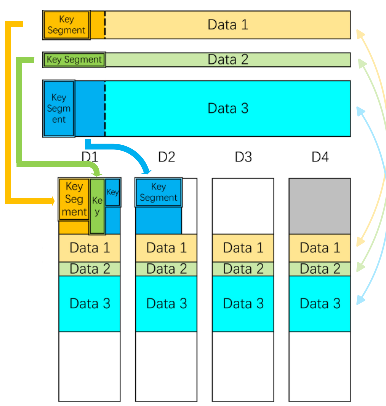


Figure 6: A sample of data distribution method, where the gray block in the upper right corner is the Q area. Here for each data segment, $\omega_1 = 15\%$, $\omega_2 = 20\%$ and $\omega_3 = 10\%$, so the $\omega_{max} = 20\%$.

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 EVALUATION

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

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 text

5.1.2 *Platforms and Configurations.*

text text

5.1.3 *Metrics.*

text text

5.1.4 *Parameters and Assumption in Our Evaluation.*

text text

5.2 Results

text text

Table 2: 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) (Minor Data)	any m disks	m disks and s sectors	high	high	high

Code Config	Storage Overhead	FT (Imp)	FT (Minor)	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

5.3 Analysis

Illustrate why Approximate Code achieve high reliability with low cost text

text text

6 CONCLUSION

text text

text text

ACKNOWLEDGMENTS

REFERENCES

[1] 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.

[2] 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.

[3] 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).