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 usually use erasure code redundant 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 reduces the storage overhead by 7.64% at the expense of reasonable probability of data quality loss.

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, 8 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 that is distinguishing data importance.

This work is usually done by experienced programmers, but fortunately, multimedia data is commonly compressed and stored in encoded formats, which results in a certain portion of such data being inherently 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 is done automatically by a system tailored to specific encodings in our design.

Our work contributions include:

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

(1) We propose an approximation code that reduces storage overhead and improves the reliability and availability of important data with an approximate strategy.
(2) We prove the mathematical correctness of the approximation code.
(3) A series of experiments show that the 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. The evaluation is presented in Section ?? 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

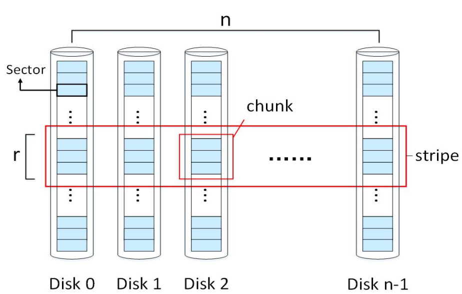


Figure 1: A sample of chunks and sectors

text text

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, so 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.

Something about H.264…

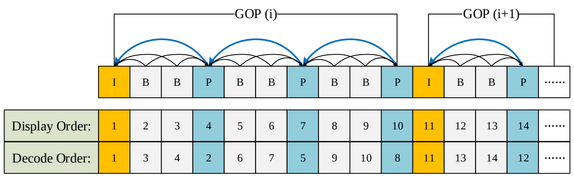


Figure 2: A sample of GOPs in H.264

Videos are typically stored in lossy compression, so they are subject to a certain quality loss when stored compared to the original version, and the extent of this quality loss can be specified by the application. For high-quality coded video, it is difficult for the human eye to distinguish the difference between them and the original video, because they preserve most of the brightness information, which human eye is very sensitive to, and some color information is discarded, which human eye is not sensitive to. Therefore, the encoded video data has an uneven degree of importance, that is, the loss of some less important data is tolerable because it's hard for human eyes to detect. On the other hand, loss of important data will have more serious consequences.

Add something related to the former subsection

In the circumstance of video approximate storage, it's common to lose some minor data making the video incomplete...

In the circumstance of data loss, it's common to lose some frames and then the video is not complete. One easy way to solve this problem is to store consecutive frames in different disks, so the video is still good for displaying only with fewer frame rates. Another alternative is that we can actually recover the lost frames by applying the a technique 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. Optical flow is commonly addressed in the video frame interpolation problem. Optical flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene. Video frame interpolation algorithms typically estimate optical flow or its variations and use them to warp and blend original frames to produce interpolation results.

Recently, phase-based methods have shown promising results in applications such as motion and view extrapolation [1][8][5]. These methods rely on the assumption that small motions can be encoded in the phase shift of an individual pixel¡s color. Meanwhile, deep learning approaches, and in particular Convolutional Neural Networks (CNNs), have set up new state-of-the-art results across many computer vision problems which also makes big improvements to frame interpolation. In neural networks, optical flow features are trained in a supervised setup mapping two frames to their ground truth optical flow field[2][4]. Among them, a multi-scale network[7] based on recent advances in spatial transformers and composite perceptual losses as well as a context-aware Synthesis approach[6] have so far produced the best results in terms of PSNR and middlebury benchmark respectively.

All the frame interpolation method are based on that we have the input of full images of two consecutive frames (maybe one or two missing in the middle). However, in the popular coding format like H.264, only I frame have full image data of itself, and all the other image data is dependent on that in I frames. Without an I frame, the following ones that depend on it will go invalid and be impossible for recovery which means I frames require more storage overhead to ensure its integrity and reliability.

2.3 Approximate Storage

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.

It loosens the requirement of storage reliability by allowing some quality loss of 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 redundantly 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 [3] 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 be undecodable. 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 frame depends on it.

We define I frames are our important data

2.4 Our Motivation

Based on Table [], the existing erasure codes 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. To solve these two 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

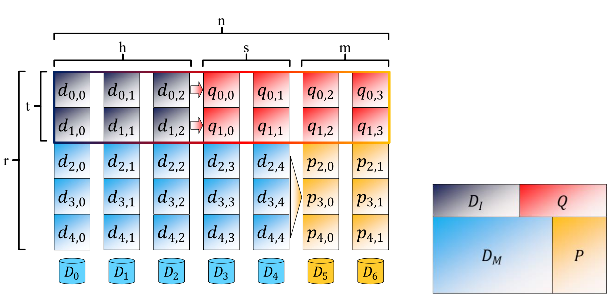
In a system of n disks, each disk can be divided into multiple sectors. We focus on the r sectors of the same logical position of each device, and we treat these r sectors as a *chunk*. The $r \times n$ sectors of n chunks constitute a stripe, as shown in Figure 1 . We also use the term symbol in coding theory to refer to sectors. Since each stripe is independent of the entire system, we only consider a single stripe.

In the n chunks of each stripe, m ones are for coding and in the remaining $n-m$ chunks. We use $s \times t$ additional sectors for coding important data, where s is the number of the columns and t is the number of the rows, as shown in Figure 3. 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 3 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 area.
- $D_M = \{d_{i,j} | t \leqslant i < r, 0 \leqslant j < n-m\}$ minor data sector area.
- $Q = \{q_{i,j} | 0 \leqslant i < t, 0 \leqslant j < s\}$ important parity sector area.
- $G = \{p_{i,j} | 0 \leqslant i < r, 0 \leqslant j < m\}$ global parity sector area.



(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 parity chunks (orange area G) and each chunk has 5 sectors. In this example, there are 6 sectors (dark blue area D_I) for important data, 15 sectors (light blue area D_M) for minor data and 4 other sectors (red area Q) are designed for encoding important data.

3.2 Encoding and Decoding Process

In this section, we use Figure 3 to illustrate the encoding and decoding process of Approximate Code$(7, 2, 5, 2, 2)$.

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.

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. When 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 any way, as long as their parity method are linearly independent of the way the important parity blocks are. Therefore, EC methods such as RS-based, XOR-based, MSR, MDS or PMDS codes can be used to construct Approximate Code because it is essentially a way of data classification and data distribution, and current EC schemes do not assume anything about data feature.

We then introduce RS-based and XOR-based Approximate Code

3.2.1 RS-based Approximate Code. In *I-Phase*, we use RS(3,2) to encode 3 groups of important data sectors and generate 2 groups of parity sectors labeled with $q_{i,j}$. The calculation of $q_{i,j}$ are defined by equation (2), where α_k is the coefficient in Galois Field (GF). The coding coefficient matrix is transformed from the Vandermonde matrix by elementary transformation.

Coding matrix like below?

$$\begin{pmatrix} 1 & 1 & 1 \\ \alpha_0 & \alpha_1 & \alpha_2 \\ \alpha_0^2 + \alpha_3^2 + \alpha_4^2 * \alpha_0 & \alpha_1^2 + \alpha_3^2 + \alpha_4^2 * \alpha_1 & \alpha_2^2 + \alpha_3^2 + \alpha_4^2 * \alpha_2 \\ \alpha_0^3 + \alpha_3^3 + \alpha_4^3 * \alpha_0 & \alpha_1^3 + \alpha_3^3 + \alpha_4^3 * \alpha_1 & \alpha_2^3 + \alpha_3^3 + \alpha_4^3 * \alpha_2 \end{pmatrix} \quad (1)$$

For example, $q_{1,1} = \alpha_0 d_{1,0} + \alpha_1 d_{1,1} + \alpha_2 d_{1,2}$.

$$q_{i,j} = \sum_{k=0}^{h-1} \alpha_k^j d_{i,k} \quad (2)$$

In *I-Phase*, we use RS(5,2) to generate 2 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 (3) and (4). For example, $p_{1,1} = \alpha_0^3 d_{1,0} + \alpha_1^3 d_{1,1} + \alpha_2^3 d_{1,2} + \alpha_3^3 q_{1,0} + \alpha_4^3 q_{1,1}$.

$$\underset{i<t}{p_{i,j}} = \sum_{k=0}^{h-1} \alpha_k^{s+j} d_{i,k} + \sum_{k=0}^{s-1} \alpha_{k+h}^{s+j} q_{i,k} \quad (3)$$

$$\underset{i \geqslant t}{p_{i,j}} = \sum_{k=0}^{n-m-1} \alpha_k^{s+j} d_{i,k} \quad (4)$$

The full recovery decoding process of RS-based Approximate Code is obvious since it is the same as the decoding process of

RS(5,2). For the approximate decoding process, there are 4 independent equations that can tolerate any 4 disk failures because the coding coefficient matrix is full rank.

3.2.2 XOR-based Approximate Code. It is difficult to design XOR-based codes that m=2,s=1. Codes like EVENODD or STAR cannot be used to recover the important data when 3 disks fails. However, the desgin of m=1 is easy, because we only use horizontal parity blocks.

One other way to generate XOR parity sectors is to divide each sector into multiple smaller blocks and apply EVENODD or RAID6 on them. This might be too complex but its correctness is easy to proof.

text text

3.3 Proof of Correctness

I am not quite clear what to write here, our proof of correctness is organized in encoding and decoding process

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 x. 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. Important data High reliability 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. Lower recovery overhead Due to the shorter checksum chain of important data blocks, the recovery code of the approximate code in the full recovery mode is less expensive.

4 IMPLEMENTATION

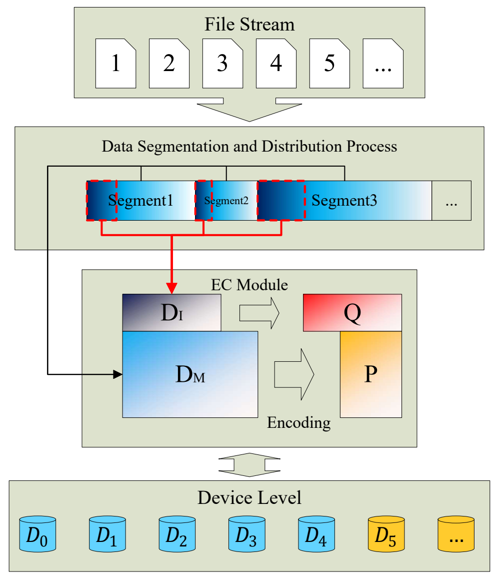


Figure 4: 111

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 module is necessary to preprocess the data. In our design, the middle layer performs automatic data identification and data distribution. It also implement *I-Phase* in the encoding process and approximate recovery mode in the decoding process.

4.1 Data Identification

For H.264 video data, we define I frames as important data, and P frames and B frames as minor data based on the analysis in the 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 $\omega_{act} = \frac{D_I}{D_I+D_M} = \frac{(n-m)\times t}{(n-m)\times r-s\times t}$ as the actual important rate the code can provide. Algorithm 1 and 2 shows the data distribution and reorganization methods.

Algorithm 1 Data Distribution Algorithm

Input: A stripe of video data.
Output: D_I and D_M.
Divide video data into several GOPs;
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 $1-\omega_{act}$;
    Store the former in D_I, the latter in D_M;
until All GOPs are classified;

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.

The data distribution scheme is shown in the figure 5. 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 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.

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

In our evaluation, we first compare the approximate code with the RS code and some XOR-based code to demonstrate the performance benefits of our solution. Since 3DFTS is a typical erase 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

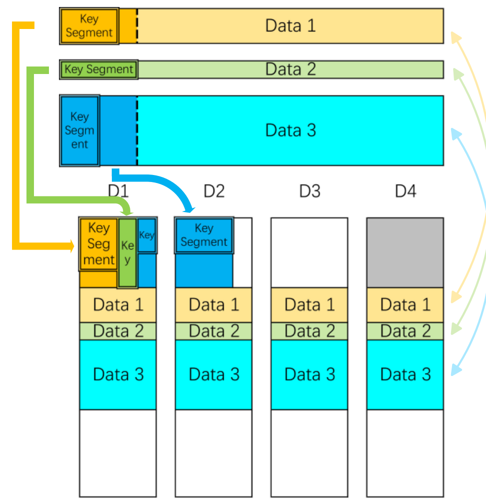


Figure 5: A sample of data distribution method, where the gray block in the upper right corner is the Q area. Here for each data segment, $\omega_1 = 15\%$, $\omega_2 = 20\%$ and $\omega_3 = 10\%$, so the $\omega_{max} = 20\%$.

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 approximation code.

5.1 Evaluation methodology

5.1.1 Erasure Codes in Our Comparisons. text

5.1.2 Platforms and Configurations. text

Table 1: 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 Efficiency	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

5.1.3 Metrics. text

5.1.4 Parameters and Assumption in Our Evaluation. text

5.2 Results

text text

5.3 Analysis

Illustrate why Approximate Code achieve high reliability with low cost text

text text

6 CONCLUSION

text text

ACKNOWLEDGMENTS

REFERENCES

[1] Piotr Didyk, Pitchaya Sitthi-Amorn, William Freeman, Frédo Durand, and Wojciech Matusik. 2013. Joint view expansion and filtering for automultiscopic 3D displays. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 221.

[2] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. 2015. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*. 2758–2766.

[3] 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.

[4] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. 2017. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2462–2470.

[5] Simone Meyer, Oliver Wang, Henning Zimmer, Max Grosse, and Alexander Sorkine-Hornung. 2015. Phase-based frame interpolation for video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1410–1418.

[6] 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.

[7] 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).

[8] Neal Wadhwa, Michael Rubinstein, Frédo Durand, and William T Freeman. 2013. Phase-based video motion processing. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 80.