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

Huayi Jin[1]

***Abstract*—Erasure codes are commonly used to ensure data availability in cloud storage systems, where video data produced by autopilot, multimedia industry and security monitoring occupies large amounts of space. Typical erasure codes, such as Reed-Solomon (RS) [1] codes or XOR-based codes use several parity disks to fully recover failure disks. However, this is expensive, not only because the scenes in which multiple disks are simultaneously damaged are relatively rare, but also because they do not consider redundant information inside the data, which results in multiple complete parity disks being excessive.**

Therefore, we propose Approximate Codes for video data, which significantly reduce storage overhead and increase availability for more important data segments. Approximate Codes provide complete recovery when fewer disks fail, and approximate recovery (recover most data) in the event of multiple disk failures. To demonstrate the effectiveness of Approximate Codes, we conduct several experiments in Hadoop and Alibaba Cloud systems. The results show that compared with the typical high-reliability erasure code schemes, Approximate Codes reduce the storage overhead by 7.64% at the expense of reasonable probability of video quality loss.

***Index Terms*—Erasure Codes, Approximate Storage, Multimedia, Cloud Storage**

I. Introduction

Currently, many cloud storage systems use erasure codes to tolerate disk failures and ensure data availability, such as Windows [], Amazon AWS [] or Alibaba Cloud. It is known that erasure codes provide much lower storage overhead and write bandwidth than replication with the same fault tolerance.

Typical erasure codes schemes can be divided into two categories such as RS-based code (RS, LRC), or XOR-based code (...). Other erasure codes (SD, STAIR) use the parity blocks to tolerate sector failures in addition to disk-level fault tolerance.

Video data consumes massive space in cloud storage systems, and this trend is exacerbated as applications demand increased resolution and frame rates. Using multiple copies to ensure video data security will generate storage cost that are several times larger than the original data, which is obviously too expensive, while erasure codes can significantly reduce this cost.

Existing erasure codes are designed to completely recover corrupted data. Typical configurations use triple disk failure tolerant arrays (3DFTS), such as Windows Azure [], which requires at least 3 parity disks. These methods are often excessive because scenes with 3 disks being corrupted at the same time are very rare as well as they do not consider that plenty of video applications can tolerate a certain amount of data loss. For example, video data typically records at least 20 frames per second, which makes losing a few frames difficult for a typical user to perceive. In addition, even if the video data suffers a certain loss, the existing AI-based interpolation algorithm and super pixel algorithm can recover most of the damaged data [].

We also find that video data is usually stored after being encoded to save space, while the encoded video data stream is non-uniformly sensitive to data loss, which makes it inappropriate to provide uniform fault tolerance using conventional erasure codes. With the motion compensation mechanism, common video coding algorithms such as H.264 only needs to store the complete content of key frames and a little part of other frames, which makes other frames rely on the key frames for computation while decoding.

Therefore, we propose Approximate Codes for video data that significantly reduce storage overhead by reducing the parity of data that is not sensitive to errors. In the scenario shown in (Figure 1), the Approximate Codes are designed for systems composed of n disks where m disks are dedicated to coding and another s sectors encoded for the first strip. This allows the data of the first stripe to tolerate any $m+s$ disks corruption, so we specifically store important segments of video data there. With an appropriate data distribution scheme, non-critical data segments will still retain $(n-2m-s)/(n-m)$ data when any $m+s$ disks are corrupted, which makes recovery schemes such as interpolated or superpixel still effective. The Approximate Codes provide two recovery modes, full recovery and approximate recovery. The former applies to no more than m disk corruptions and recovers all data, the latter applies to no more than m+s disks corruptions and retains important data.

II. Related Work and Our Motivation

Video data can be classified into hot data and cold data, the former requiring high availability and reliability, which often relies on expensive replication approach. The latter's storage usually uses erasure codes schemes, because cold data far exceeds hot data, and erasure codes can guarantee its reliability with low monetary overhead. However, existing erasure code schemes do not consider the characteristics of video data and are not specifically designed for them.

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, it is tolerable that some of those less important data losses because the users are hard to detect. On the other hand, loss of important data will have more serious consequences.

This section then introduces the relevant background and our motivation.

A. Existing Erasure Codes

RS/LRC/CLAY/SD/STAIR???
XOR-based
RS-based
RAID6
MSR

B. 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.

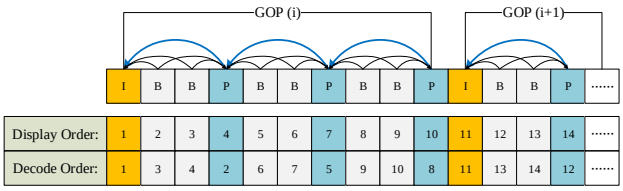

Fig. 1. A sample of GOPs in H.264

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.

C. 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 [] 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 II-B, 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.

D. 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 between important and non-critical data to reduce storage overhead and protect critical data better.

III. Approximate Code

In this section, we introduce the design of Approximate Codes and its properties through a simple example. In order to be general and easy to describe, we use fewer data blocks and more parity blocks (resulting in greater storage overhead). A more optimized parameter selection scheme for practical applications will be introduced in XXX.

A. 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 2 . We also use the term symbol in coding theory to refer to sectors. Since each strip is independent of the entire system, we only consider a single strip.

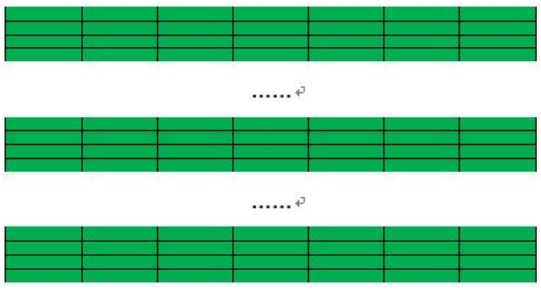

Fig. 2. TEST–A sample of chunk and sector

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. Our design assumes that the parameters (n,m,s,t) are configurable parameters shus the constructor of the Approximate Code will be determined by these 4 parameters. Although the choice of parameter r is arbitrary, we recommend setting it to $r = s+h$. The relevant instructions will be shown in III-C. Meanwhile, we label $h = n-m-s$ as the number of columns of important data for convenience. Figure 3 shows an example of Approximate Codes with $n = 7$, $m = 2$, $s = 2$ and $t = 2$, where we label the data disks with $d_{i,j}$, the important data parity sectors with $q_{i,j}$ and the normal parity sectors with $p_{i,j}$.

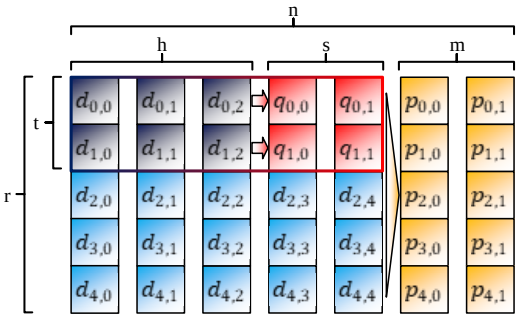

Fig. 3. A sample of Approximate Codes $(7,2,2,2)$ with 7 chunks where 2 of them are parity chunks (presented by orange) and 4 other sectors (presented by red) are designed for encoding important data. In this example, there are 6 sectors (presented by dark blue) for important data and 15 sectors (presented by light blue) for normal data.

A variety of erasure codes can be used for the encoding and decoding process of Approximate Codes. Here we only introduce our RS-based design as an example. Approximate Codes based on other erasure code schemes include XOR-based codes or MSR codes will be introduced in Section IV.

1) Encoding Process: The encoding process can be divided into two phases: important coding phase and global coding phase.

In *phase 1*, we use RS$(h+s,h)$ to encode h groups of important data sectors and generate s groups of parity sectors labeled with $q_{i,j}$. The calculation of $q_{i,j}$ are defined by equation 1, where α_k is the coefficient in Galois Field (GF). For example, $q_{1,1} = \alpha_0^2 d_{1,0} + \alpha_1^2 d_{1,1} + \alpha_2^2 d_{1,2}$. As shown in the blod box in Figure **??**, we use RS(5,3) in this phase.

$$q_{i,j} = \sum_{k=0}^{h-1} \alpha_k^{j+1} d_{i,k} \tag{1}$$

In *phase 2*, we use RS$(n,n-m)$ to generate m parity chunks labeled with $p_{i,j}$ from $n-m$ chunks consist of data sectors and important parity sectors. The calculation of $p_{i,j}$ are defined by equation 2 and 3. Despite the difference in expression, the encoding process for $p_{i,j(i<t)}$ and $p_{i,j(i\geqslant t)}$ are same on chunk level because we treat $q_{i,j}$ and $b_{i,j}$ as the same on chunk level. For example, $p_{1,1} = \alpha_0^4 d_{1,0} + \alpha_1^4 d_{1,1} + \alpha_2^4 d_{1,2} + \alpha_3^4 q_{1,0} + \alpha_4^4 q_{1,1}$. As shown in orange chunks in Figure **??**, we use RS(7,5) in this phase.

$$\underset{i<t}{p_{i,j}} = \sum_{k=0}^{h-1} \alpha_k^{s+j+1} d_{i,k} + \sum_{l=0}^{s-1} \alpha_{l+h}^{s+j+1} q_{i,l} \tag{2}$$

$$\underset{i\geqslant t}{p_{i,j}} = \sum_{k=0}^{n-m-1} \alpha_k^{s+j+1} d_{i,k} \tag{3}$$

B. Proof of Correctness

C. Data Distribution

D. Properties of Approximate Code

IV. Evaluation

A. An Evaluation methodology

1) Erasure Codes in Our Comparisons:

2) Platforms and Configurations:

3) Metrics:

4) Parameters and Assumption in Our Evaluation:

B. Results

C. Analysis

Illustrate why Approximate Code achieve high reliability with low cost

V. Conclusion

Acknowledgment

References

[1] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Elsevier, 1977, vol. 16.