\section{Introduction}

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

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

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

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

Therefore, we propose Approximate Codes for multimedia data that reduce storage overhead by reducing the parity of data that is not sensitive to errors. In the scenario shown in Figure \ref{fig-ap-41425}, 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.

Our work contributions include:

\begin{enumerate}

\item We propose Approximation Code that reduces storage overhead and improves the reliability and availability of important data with the approximate strategy.

\item We prove the mathematical correctness of the Approximation Code.

\item 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.

\end{enumerate}

The rest of the paper is organized as follows. In Section \ref{RelatedWork}, we introduce related work and our motivation.

In Section \ref{ApCode}, the design of Approximate Code and its encoding and decoding process will be illustrated in detail.

Section \ref{Implementation} introduce the implementation of our design.

The evaluation is presented in Section \ref{evaluation} and the conclusion of our work is in Section \ref{Conclusion}.