# 纠删码存储系统中的优化方法综述

张嘉琪[1)]

[1)](华中科技大学 计算机科学与技术学院，湖北省武汉市 435400)

**摘 要** 如今的数据中心为了保持数据的可靠性，常常采用纠删码来产生冗余数据。纠删码方案相较于数据复制方案，虽然节省了很多的存储空间，但是数据存储和修复时的开销相对较大。本文调研了近几年在纠删码存储系统上做出的一些改进工作。Shan和Chen[1]等人提出了针对对象存储系统中，RC编码修复过程的优化，将存储对象根据几何划分规则，分割为大小不同的数据块，再进行RC编码，并通过流水技术将修复与传输过程重叠起来，大大减低降级读取时间。Wang[2]等人提出了一种自适应规则，即对高频访问的数据使用LRC进行编码，而低频使用的数据采用HH编码，从而在大大降低降级读取时间的同时保持很好的存储效率。以及其他的一些工作。

关键词 对象存储；纠删码；几何划分；自适应规则；转换算法

## Summary of Improvements in Erasure Coded Storage System

Zhang Jiaqi[1)]

[1)](School of Computer Science & Technology, Huazhong University of Science and Technology, Wuhan Hubei, China)

**Abstract** Nowadays, data centers usually use erasure code to maintain its reliability. Although the erasure code scheme saves a lot of storage space, compared to the data replication scheme, the overhead in data storage and repair is relatively large. This paper investigates some improvements that has been done recent years on erasure coded storage system. Shan et al. propose a novel scheme for regenerating code called Geometric Partitioning, which organizes an object into several data chunks with different size and uses pipeline to parallel repair and transfer. It aims at the repair procedure of regenerating code and greatly reduces degraded read time. Wang et al. introduce an adaptive scheme for erasure code and a code-switching algorithm. They use local reconstruction code to encode hot datum which is frequently accessed and Hitchhiker code to encode cold datum which infrequently accessed. It could make the storage system maintain a high storage efficiency while tremendously reduce degraded read time.

**Key words** Object Storage; Erasure Code; Geometric Partitioning; Adaptive Scheme; Code-Switching

## 1 引言

如今的存储系统为了应对海量的数据存储需求，通常由大量的存储节点构成。而为了防止节点出现故障，经常会通过一定的技术手段来保证数据的可靠性。常见的手段就是给数据中加入冗余，以便于在部分数据发生故障时，能够通过冗余信息来将其还原。很多存储系统采用复制和编码技术来产生冗余。与简单地存储相同副本的复制不同，纠删码能够以更少的存储消耗达到相同的容错程度，因此纠删码在分布式存储系统中被大量采用。

纠删码的操作总体上来说可以分为两个过程，即编码和解码。编码就是根据现有的数据块进行计算产生一定的冗余信息，从而产生新的存储块。而解码则是根据现存的数据块剔除冗余数据来还原数据块的信息。虽然，纠删码能够在保持较高的存储效率的同时产生足够的冗余数据，但是，当单个节点受损时，便需要通过网络来读取其他现存数据块，以恢复该节点数据。从而，纠删编码往往会占用很大的网络带宽以及引起多次I/O操作。

而如今已经有很多工作致力于对纠删编码的效率进行优化，本文将对从不同方面进行的优化进行总结。

# 2 背景介绍

## 2.1 纠删码入门知识

纠删码通常用(k, r)来表示，其中k表示含有k个数据块或k位信息位，r表示r个校验块或r位检验位。

## 2.2 修复

纠删码的修复，通常指如下过程：从其他节点收集必要的数据，然后对该数据进行解码操作，以获取当前不可利用的数据。下面介绍不同纠删编码的修复过程。

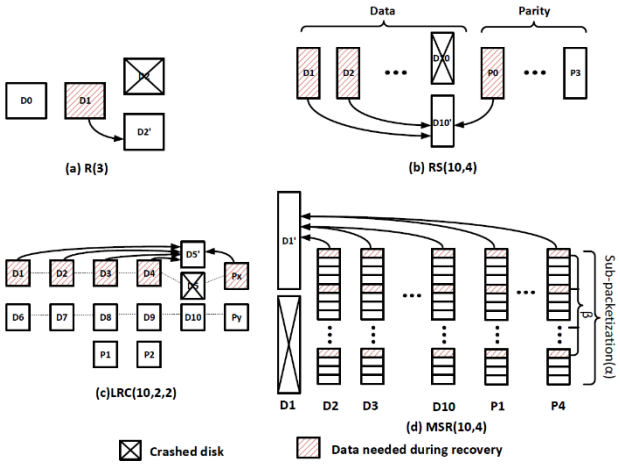


图1

RS编码的修复过程。RS编码是一种被普遍使用的纠删码。图1(b)展示了RS编码的修复过程。图中可以看出，对于单个节点失效的情况，需要从剩余所有幸存的节点传输数据，然后通过解码操作来恢复数据。这个操作十分耗时，同时也会进行大量的I/O操作和占用大量的网络带宽。RS编码之所以可靠性比较高，是因为它是一种MDS码。MDS码是指能够修复任意个r节点失效的编码。

LRC编码的修复过程。LRC编码通过减小连接的节点数来减小磁盘I/O和网络流量。图1(c)展示了LRC编码的修复过程。其中$P_x$和$P_y$是两个本地的校验节点。LRC编码将数据节点划分为多个组，给每个组生成各自的本地校验节点，最后再生成全局校验节点来增加可靠性。当单个节点失效时，只需要访问该分组的节点即可，从而减小磁盘I/O和网络流量。但是LRC编码在存储效率和可靠性之间的权衡并没有RS编码好，这是因为，它并不是MDS码。

RC编码的修复过程。不同于LRC编码的策略，RC编码需要读取d(d>k)个节点。RC编码是通过引入粒度更小的子块来减小磁盘I/O和网络流量的。RC编码将一个数据块分割成多个子块，当一个节点失效时，只需要从d个节点中分别读出一小部分子块便能完成修复。每个数据块所包含的子块数量记为α，修复过程中需要从每个节点中读取的子块数量记为β。

## 2.3 降级读取

对象存储系统是指，系统存储的是一些不可分割的较大的二进制对象，包括照片，视频和压缩文件等等。对于一个对象存储系统来说，有两个和修复过程紧密相连的操作，分别是：

1) 对当前不可利用的对象的降级读取，通常由于系统维护或网络故障引起。

2) 对已损坏的硬盘或者失效的节点的修复。

降级读取时间是十分影响用户的体验的，因此也是对象存储系统的一个重要的衡量指标。为了获取当前暂时不可利用的对象，会发生如下操作：用户向存储系统发送一条HTTP请求，然后HTTP服务器将该对象进行修复，并将该对象传输给用户。

因此，对于一个对象的修复过程和传输过程是可以通过流水线的方式来堆叠起来，从而节省时间。因此，降级读取时间取决于对象的修复时间、对象的传输时间，以及它们是如何通过流水线技术连接起来的。

# 3 优化

## 3.1 几何划分

RS编码通常和和子块划分技术一起使用，来减小磁盘I/O。数据块在进行编码之前就已经被划分为子块了，同时，RS编码只能在块级的粒度下进行修复。这也意味着，数据块的大小是固定的，你无法在编码时采用较大的块尺寸，在解码时使用较小的块尺寸。这也使得块尺寸的选择变得很重要。

采用较大的块尺寸可以增加修复效率。尽管RS编码能够大幅度地减少需要读取的数据量，但是，它也带来了碎片和不连续读取。当块尺寸较小时，尽管所需要的数据量是一定的，但是，在网络中传输的通信块地大小却很大，因为没有充分利用到网络的带宽，导致每次都需要传输额外的无用数据。当块尺寸小于网络单次传输I/O的尺寸时，性能将大大降低。过大的块尺寸将会影响降级读取时间。尽管，前文提到块尺寸应当大一些，但是并不能无限制地增大。当块尺寸比对象本身还要大的时候，就会引起读取放大并且增大降级读取时间。

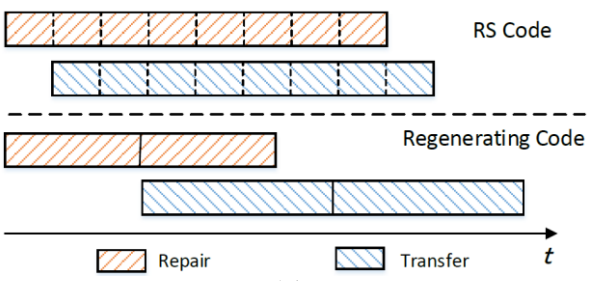


图2

采用较小的块尺寸可以降低降级读取时间。降级读取的过程可以分为两部分：修复存储服务器上

的必要数据和将修复好的对象传输给用户。对于较大的对象，将这两个过程划分为更多的小过程可以极大的通过流水线来减小降级读取时间。这对于RS编码来说是很自然的，但是对于RC编码来说，如果第一块尺寸过大，流水线可能会堵塞。

总结来说，当块过大时，流水线可能会发生堵塞，且修复完成之后需要将额外的数据剔除，即修复过程中修复了很多无用的信息。当数据块过小时，不能充分利用磁盘的性能，且网络需要传输很多无用的信息。下图展示了平均降级读取时间和平局带宽与数据块大小的关系。

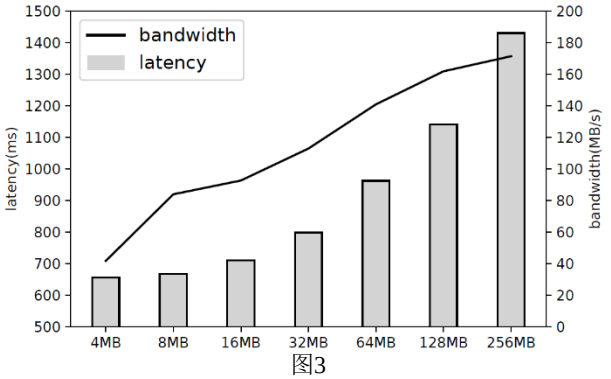


图3

经过上述分析，我们知道，如果采用固定的块尺寸，我们便不能充分利用RC编码的性能。于是，我们提出了几何划分，即对于同一个存储对象，我们可以将其划分为大小不同的块。下图展示了具体做法。

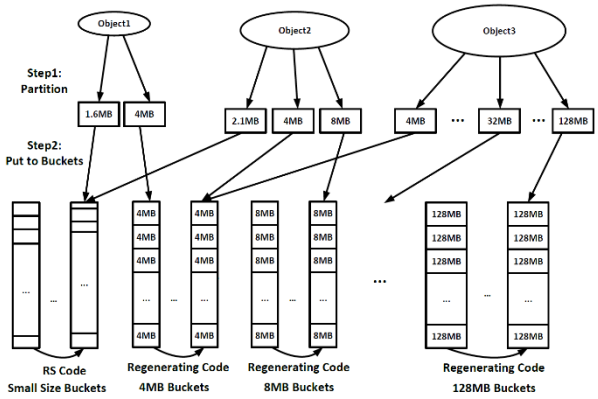


图4

对于每个大小为$S$的对象，我们可以将其划分为$S = R + \sum_{i=1}^{n} a_i \cdot s_0 q^{i-1}$。其中，$s_0$是指几何序列的初始大小，$q$是指序列的比率，$R = S \ mod \ s_0$，$a_i$表示该尺寸的buckets中含有多少个该对象的块。其中bucket是一个磁盘上的大文件，它包含着来自不同对象的大小相等的数据块。而一个对象的不同数据块都存储在同一个磁盘上。bucket所包含的数据块的尺寸是按照比率$q$指数上升的。不同大小的bucket的数据块尺寸组成的序列，就称为几何序列。为了方便划分，需要将每个对象的前部切分下来，存入small size buckets中，这类bucket不同于先前的，他们内部的数据块是没有固定尺寸的，这类bucket通过RS编码来产生冗余，使其能够有较高的修复效率，又因为他们尺寸较小，不会因为存储效率不高而对整个系统的存储效率产生较大影响。后面的bucket通过RC编码来产生冗余，以获得较高的存储效率。

通过几何划分，再结合流水线技术，当用户请求一个当前不可利用的对象时，便可以先修复对象的前部。因为前部较小，便能够在很短的时间内响应用户的请求，从而降低降级读取时间，而后续的修复和传输过程都通过流水线技术来堆叠，进一步增加整个对象的修复效率。下图展示了传输速度大于修复速度和传输速度小于修复速度的情况。

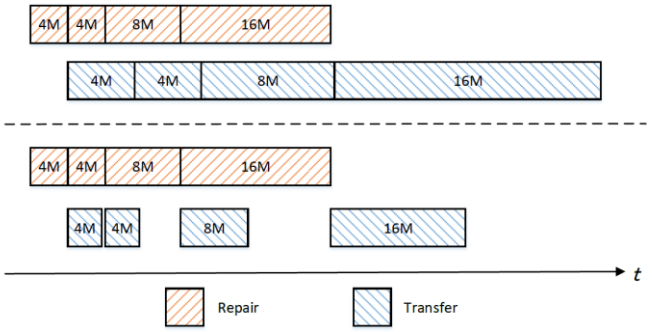


图5

### 3.2 自适应规则与编码转换算法

尽管传统的纠删码能够优化存储效率，但是它们也会在修复过程中极大地增大磁盘I/O和占用带宽，这是因为在纠删码存储系统中，修复一个数据节点，需要该节点好几倍的数据量。修复过程中数据需求的增加，导致了很高的降级读取延迟。

自适应规则采用多种编码方式来存储数据。它们的主要思想就是根据数据的不同的特点，来采用不同的编码方式。例如，对于高频读取的数据，应该更侧重优化降级读取延迟，而低频读取的数据应该更加注重存储效率。HACFS就是一种著名的自适应规则，它采用同一个编码族的两种编码来给数据编码。

本文提出一种新的自适应规则来保证数据存储的可靠性和效率。这个策略采用了两种来自不同家族的纠删码。第一种编码是LRC编码，它能够以非常低的修复花销来存储高频访问的数据。另一个是Hitchhiker编码，它能够在存储低频访问的数据时，保证很高的存储效率。根据分析，我们可以知道，对于一个分布式存储系统，整个系统的降级读取时间更多的是由高频访问的数据，但系统的存储效率则更多的由低频访问的数据来决定，因为这类数据在系统中的比例更高。在该规则中，数据是属于高频访问的还是低频访问的，是可以变化的，因此与之相匹配的还有一个转化算法。在该文件系统中，有着一个文件描述器。当系统的真实存储开销大于预期的开销时，我们将高频访问数据中的最近访问频率较低的部分设为低频访问。当一个高频访问数据，在一个阈值时间段内都没有被访问到，也会被设为低频数据。同理，当系统的实际存储开销

小于预期开销时，就将低频数据中最近访问频次较高的数据设为高频数据，而如果低频数据在短时间内被反复访问，也会被设为高频数据。

**Algorithm 1** code-switching algorithm from $(k, m-1, m)$-LRC to $(k, m)$-HH code

**Input:** two stripes $a$ and $b$ in LRC
**Output:** two sub-stripes $s_a$ and $s_b$ in HH code

$s_a.data := a.data$
$s_b.data := b.data$
$s_a.parity := a.global\_parity$
$s_b.parity[1] := b.global\_parity[1]$
**for** $i := 2$ to $m$ **do**
  $s_b.parity[i] := b.global\_parity[i] \oplus a.local\_parity[i-1]$
**end for**

图6

因为高频数据与低频数据是分别采用了不同的编码方式的，因此在将数据从一种类型变更为另一种时，需要用到一个比较高效的转化算法。下面两个算法分别展示了从LRC编码转换到HH编码，和从HH编码转换到LRC编码。

**Algorithm 2** code-switching algorithm from $(k, m)$-HH code to $(k, m-1, m)$-LRC

**Input:** two sub-stripes $s_a$ and $s_b$ in HH code
**Output:** two stripes $a$ and $b$ in LRC

$a.data := s_a.data$
$b.data := s_b.data$
$a.global\_parity := s_a.parity$
$b.global\_parity[1] := s_b.parity[1]$
**for** $i := 1$ to $m-1$ **do**
  $a.local\_parity[i] := 0$
  $b.local\_parity[i] := 0$
  **for** each data chunk $j$ in $G_i$ **do**
    $a.local\_parity[i] := a.local\_parity[i] \oplus a.data[j]$
    $b.local\_parity[i] := b.local\_parity[i] \oplus b.data[j]$
  **end for**
**end for**
**for** $i := 2$ to $m$ **do**
  $b.global\_parity[i] := s_b.parity[i] \oplus a.local\_parity[i-1]$
**end for**

图 7

### 3.3 混合局部性

纠删码中一个 stripe 中包含 n 个数据节点，可以容忍任意 n-k 个节点的失效。但是，通常来说纠删码的参数会在中等大小来取值，这是因为纠删码的修复惩罚的限制。如果纠删码的参数取得过大，在修复单个节点失效的时候，就需要传输过多的节点，造成更多的磁盘 I/O 和网络流量。因此，stripe 的大小 n 通常不会大于 20。但是，如果 stripe 的大小可以取得很大，就能够减小存储冗余，从而获得更好的存储效率，进而在存储系统投产的时候节省成千上万的成本。

宽带存储确实可以获得比较极限的存储效率，但是也会造成更加严重的修复惩罚。现存的很多工作都是通过最大化利用局部性来减小修复过程中的带宽占用的。局部性可以分为两种

1） 校验局部性，它通过引入额外的局部校验块来减少单个节点失效时需要访问的数据块数量。

2） 拓扑局部性，它着重考虑系统结构的性质，并且通过本地的修复操作来减少跨聚类的通信带宽。

但是，通过研究可以发现，现存的工作仍然是选取一个比较小的 k 值，因此不可避免的增加了存储冗余和修复性能。这个现象的原因是，当宽带存储的冗余接近极限的时候，就会抵消掉校验局部性或拓扑局部性带来的优化。

介于此，提出了混合局部性，它可以系统地将校验局部性和拓扑局部性来解决宽带条件下纠删码的问题。混合局部性将本地检验块和一小部分数据块子集联系起来，并且在数量很小的聚类中本地化一个修复操作。因此，它能够在存储冗余和修复性能之间做到更好的平衡，从而能比当前最优算法做得更好。以下是混合局部性的参数说明。

| Notation | Description |
|---|---|
| $n$ | total number of chunks of a stripe |
| $k$ | number of data chunks of a stripe |
| $r$ | number of retrieved chunks to repair a lost chunk |
| $z$ | number of racks to store a stripe |
| $c$ | number of chunks of a stripe in a rack |
| $f$ | number of tolerable node failures of a stripe |
| $\gamma$ | maximum allowed redundancy |

图 8

为了获得混合局部性的最佳性能，我们分析可知，它在修复一个数据块的时候，需要下载 r-1 块数据块外加一块校验块。既然混合局部性会把 r 块数据中的某一些放在校验聚类中，它就能在每个聚类中进行一个本地修复操作，从而减少了跨聚类的通信带宽。显然，如果 c 增大，即一个聚类中同一个 stripe 的数据块的数量增大，本地的修复就能覆盖更多的数据块，从而进一步减小带宽占用。因此，对于每一种编码，只要找打尽可能大的 c 值即可。

## 4 总结

本文研究了近几年对纠删码存储系统中做出的一些改进工作。其中，针对 RC 编码的子块划分

机制，因为固定大小的数据块不能充分发挥 RC 编码的优势，不能在降级读取时间和修复效率之间获得很好的平衡，所以 Shan 等人提出了几何划分，将存储对象划分为大小不同的数据块，并通过流水线技术来减小降级读取时间以及优化修复效率。Wang 等人发现，一个存储系统的降级读取延迟主要取决于高频读取的一小部分数据，而整个系统的存储效率则取决于低频读取的数据，这部分数据在系统中占比很大，于是提出了一种自适应的规则。采用 LRC 编码去对高频数据产生冗余，而采用 HH 编码对低频数据产生冗余。并提出了两种编码之间转换的算法。

## 参 考 文 献

[1] Yingdi Shan, Kang Chen, Tuoyu Gong, Lidong Zhou, Tai Zhou, Yongwei Wu. Geometric Partitioning: Explore the Boundary of Optimal Erasure Code Repair 2021 Symposium on Operating Systems Principles | October 2021

[2] Z. Wang, H. Wang, A. Shao and D. Wang, "An Adaptive Erasure-Coded Storage Scheme with an Efficient Code-Switching Algorithm," 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), 2020, pp. 1177-1178, doi: 10.1109/ICDCS47774.2020.00129.

[3] Hu Y, Cheng L, Yao Q , et al. Exploiting Combined Locality for Wide-Stripe Erasure Coding in Distributed Storage[C]// File and Storage Technologies. 2021.

[4] Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, et al. 2020. Delta lake: high-performance ACID table storage over cloud object stores. Proceedings of the VLDB Endowment 13, 12 (2020), 3411–3424.

[5] H. Shi and X. Lu, "INEC: Fast and Coherent In-Network Erasure Coding," SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1-17, doi: 10.1109/SC41405.2020.00070.