计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2020.20190675
57(11): 2419-2431, 2020

纠删码存储系统数据更新方法研究综述

张 耀 储佳佳 翁楚良
(华东师范大学数据科学与工程学院 上海 200062)
(zhangyao@stu.ecnu.edu.cn)

Survey on Data Updating in Erasure-Coded Storage Systems

Zhang Yao, Chu Jiajia, and Weng Chuliang
(*School of Data Science and Engineering, East China Normal University, Shanghai* 200062)

Abstract In a distributed storage system, node failure has become a normal state. In order to ensure high availability of data, the system usually adopts data redundancy. At present, there are mainly two kinds of redundancy mechanisms. One is multiple replications, and the other is erasure coding. With the increasing amount of data, the benefits of the multi-copy mechanism are getting lower and lower, and people are turning their attention to erasure codes with higher storage efficiency. However, the complicated rules of the erasure coding itself cause the overhead of the read, write, and update operations of the distributed storage systems using the erasure coding to be larger than that of the multiple copies. Therefore, erasure coding is usually used for cold data or warm data storage. Hot data, which requires frequent access and update, is still stored in multiple copies. This paper focuses on the data update in erasure-coded storage systems, summarizes the current optimization work related to erasure coding update from the aspects of hard disk I/O, network transmission and system optimization, makes a comparative analysis on the update performance of representative coding schemes at present, and finally looks forward to the future research trends. Through analysis, it is concluded that the current erasure coding update schemes still cannot obtain the update performance similar to that of multiple copies. How to optimize the erasure-coded storage system in the context of erasure coding update rules and system architecture, so that it can replace the multi-copy mechanism under the hot data scenario, and reducing the hot data storage overhead is still a problem worthy of further study in the future.

Key words erasure codes; distributed storage systems; data update; multiple copies; storage overhead

摘 要 在分布式存储系统中,节点故障已成为一种常态,为了保证数据的高可用性,系统通常采用数据冗余的方式.目前主要有2种冗余机制:一种是多副本,另一种是纠删码.伴随着数据量的与日俱增,多副本机制带来的效益越来越低,人们逐渐将目光转向存储效率更高的纠删码.但是纠删码本身的复杂规则导致使用纠删码的分布式存储系统的读、写、更新操作的开销相比于多副本较大.所以纠删码通常被用于冷数据或者温数据的存储,热数据这种需要频繁访问更新的场景仍然用多副本机制存储.专注于纠删码存储系统内的数据更新,从硬盘I/O、网络传输、系统优化3方面综述了目前纠删码更新相关的优化工作,对目前具有代表性的编码方案的更新性能做了对比分析,最后展望了未来研究趋势.通过分析

收稿日期:2019-09-16;**修回日期**:2019-12-30
基金项目:国家自然科学基金项目(61772204,61732014)
This work was supported by the National Natural Science Foundation of China (61772204, 61732014).
通信作者:翁楚良(clweng@dase.ecnu.edu.cn)

发现目前的纠删码更新方案仍然无法获得和多副本相近的更新性能.如何在纠删码更新规则和系统架构角度优化纠删码存储系统,使其能够替换掉热数据场景下的多副本机制,降低热数据存储开销仍是未来值得深入研究的问题.

关键词 纠删码;分布式存储系统;数据更新;多副本;存储开销

中图法分类号 TP309.3

在分布式存储系统中,节点故障已成为常态,根据文献[1],Facebook 数据仓库每天平均有 50 个节点不可用.通常分布式存储系统会采用数据冗余的方式来保证系统的可用性,以免机器宕机、网络分区等故障带来的数据丢失、请求不能正常响应情况的发生.目前数据冗余的方式主要有 2 种:一种是多副本,另一种是纠删码.随着大数据时代的到来,每天都会产生大量的数据,这给存储系统带来了更大容量的需求,随之产生的成本开销也越来越大.根据 Facebook 统计数据显示[2],数据中心中每 PB 的数据每个月所产生的经济开销大约是 200 万美元.多副本机制的使用无疑使这种情况更加恶化.随着 CPU、网络设备等新硬件技术的快速发展,人们逐渐把目光从多副本方式转向纠删码.相比于多副本方式,纠删码能够在保证相同甚至更高的系统可用性的同时,将存储开销降低为原来的 50%,甚至更低,这节省的经济开销无疑是巨大的.目前越来越多的分布式存储系统开始支持纠删码方式的存储,像 HDFS[3],Windows Azure[4],Ceph[5],f4[6]等.

但是纠删码本身也存在缺点.和多副本直接将原始数据完整拷贝多份作为冗余信息不同,纠删码之所以能够将存储开销降低,是因为其利用特定的编码规则对原始数据进行特定的结合,计算后生成少量的冗余信息,冗余信息和编码规则结合能够恢复出出错的数据.当原始数据中的一些数据被更新时,对应的冗余信息也同样需要更新来保证数据与冗余信息之间的一致性.可以看出,编码、解码、更新过程都需要计算消耗 CPU 资源.而且,解码和更新操作都需要读取相应的数据和冗余信息,这会消耗更多的网络带宽和硬盘 I/O.和多副本机制相比,纠删码的写、更新、恢复数据过程更加复杂,导致纠删码的性能相对较差.为了兼顾系统性能和存储效率,目前纠删码方式多是用于冷数据或者温数据的存储.像热数据这种需要频繁读、写、更新,并且对性能要求很高的场景还是用多副本方式存储.很多研究工作致力于改善纠删码的性能,这些工作大多只是专注于其中一个方面.但是对于热数据存储来说,读、写、更新 3 种操作对性能的要求都很高,只有当纠删码在这 3 个方面的性能达到和多副本同样性能甚至更好时,才能真正替换掉多副本机制,获得更高的存储效率.近年来不断发展的非易失性内存(non-volatile memory, NVM)技术能够极大地改善计算机的计算能力和存储性能,消除了硬盘 I/O 开销,使得纠删码用于热数据存储成为可能.

目前国内外存在少量的纠删码技术的研究综述.文献[7]从纠删码的编码、解码、更新 3 个方面对现有的研究工作做了概括性的总结分析,文献[8]仅关注了再生码和本地修复码 2 种编码的技术进展,文献[9]主要介绍编码方面的相关工作,文献[10]专注于数据恢复方面的工作,还没有专门一篇介绍纠删码更新这一重要工作领域的综述.文献[7]也仅仅是从更新规则角度介绍了纠删码几种基本的更新方法,并没有全面分析和纠删码更新有关的工作.所以本文将专注于纠删码存储系统中的数据更新操作,从硬盘 I/O、网络传输、系统优化 3 个方面对现有的纠删码更新的相关工作做了深入分析,并对目前具有代表性的编码方案的更新性能做了对比分析,最后指出纠删码未来的研究方向.

1 纠删码基本原理和背景知识

本节主要介绍了纠删码的编码、解码、更新的基本原理,分析了和多副本机制相比,纠删码更新的额外开销.

1.1 基本原理

在分布式存储系统中,数据通常以固定大小的块的形式存储起来,存储数据的块被称为数据块(data block),存储冗余信息的块被称为冗余块,在纠删码中也被称为校验块(parity block).存放数据块的节点和存放校验块的节点被分别称为数据节点和校验节点.纠删码存储系统中数据块和校验块被统称为编码块.存储开销和容错能力是分布式存储系统中非常重要的 2 个指标,本文对存储开销和容错能力的定义为:

1) 存储开销.指冗余块和数据块大小之和与数据块大小的比值,以 3 副本为例,其存储开销是 3.

2) 容错能力.指存储系统在能够正常提供服务的前提下允许发生故障的机器的最大数目,还是以 3 副本为例,其容错能力为 2.

纠删码作为数据存储中除了多副本之外的另一种保护数据可用性的方法,在提供相同容错能力的同时,能够极大地降低存储开销.因为目前大多数存储系统使用最广泛的是线性纠删码,所以本文将只关注线性纠删码.线性纠删码的编码、解码重构操作都能通过在有限域内的线性操作来完成.编码的基本原理是将原始数据以 w 位为单位在有限域(2^w)[11]内线性结合得到对应的校验信息,w 通常为 8,也就是 1 B[12].通常来说,存储系统会将被编码的数据划分成 k 个固定大小的数据块,通过编码生成 m 个校验块,然后将这 k 个数据块和 m 个校验块分别存储到不同的故障隔离域内.k 个数据块和 m 个校验块组成 1 个逻辑条带(stripe),这是维护数据可用性的基本单元,能够允许 1 个条带内同时出现 m 个编码块损坏(容错能力为 m),这样一种编码被称为($k+m$,k)码.($k+m$,k)码的基本原理的数学形式为

$$(p_0,p_1,\cdots,p_{m-1})^{\mathrm{T}}=\boldsymbol{G}\times(d_0,d_1,\cdots,d_{k-1})^{\mathrm{T}},\quad(1)$$

其中,p_i 为校验块($0\leqslant i\leqslant m-1$),$d_j$ 为数据块($0\leqslant j\leqslant k-1$),$\boldsymbol{G}$ 为生成矩阵.在大多数存储系统中采用的纠删码都有这样一个特性:条带内的数据块以原生的形式存放,这样使得系统能够不用执行任何编码操作直接响应请求.图 1 是著名的 RS(Reed-Solomon)编码[13]的编码过程的图形化表示.假设第 1 个和第 2 个数据块出现损坏,利用生成矩阵构造出解码矩阵(去除生成矩阵第 1 行和第 3 行后的方阵的逆矩阵),执行相应的矩阵运算即可恢复出损坏的数据块.

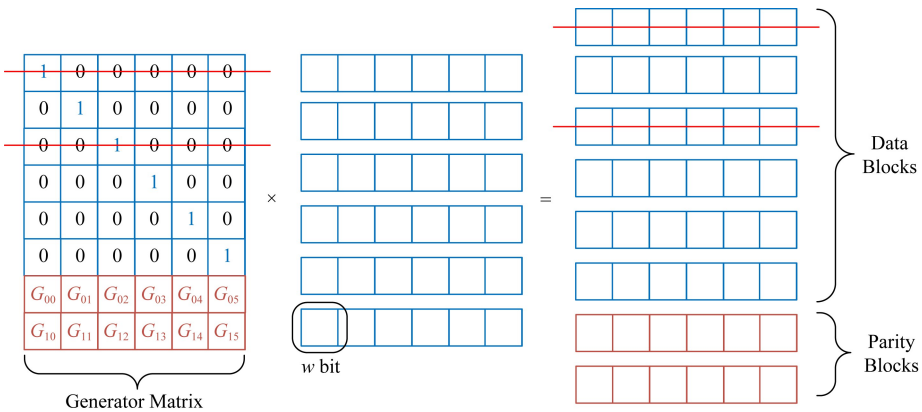

Fig. 1　A pictorial representation of erasure coding encoding principle

图 1　纠删码编码原理的图形化

1.2　纠删码基本更新方法

从图 1 可以看出,校验块的生成是每个数据块对应位置的数据在有限域内的线性结合,当数据块发生变化时,校验块也需要相应地更新,如此才能在后期数据出现损坏或不可访问时利用相应的解码规则恢复出正确的数据.

最直接也是最传统更新校验块的方法,就是重新执行一遍编码.重新编码是一个非常耗时的过程,这需要将条带内所有的数据块读出,重新计算出校验块.以 RS(5,3)为例,当更新数据块 d_0 时,需要消耗 5x 网络 I/O(读出 d_1,d_2,写回新编码块 d'_0,p'_0,p'_1),5x 硬盘读写(读出 d_1,d_2,写入 d'_0,p'_0,p'_1),但是在 3 副本中只需要 3x 的网络 I/O 和硬盘写.当被更新的数据很小时,这种差距更明显.这种更新方法适合于条带内大多数数据都被更新的情况.如果只有 1 个数据块,甚至是 1 个数据块内的一部分被更新,这种方式的开销无疑是巨大的.

第 2 种更新方法是采用增量更新.已知 $p_i=\sum_{j=0}^{k-1}G_{ij}d_j$,假设第 k 个数据块 d_{k-1} 被更新为 d'_{k-1},新校验块计算为:

$$p'_i=\sum_{j=0}^{k-2}G_{ij}d_j+G_{i(k-1)}d'_{k-1}=$$

$$\sum_{j=0}^{k-2}G_{ij}d_j+G_{i(k-1)}d_{k-1}-G_{i(k-1)}d_{k-1}+$$

$$G_{i(k-1)}d'_{k-1}=\sum_{j=0}^{k-1}G_{ij}d_j+G_{i(k-1)}\Delta d_{k-1}=$$

$$p_i+G_{i(k-1)}\Delta d_{k-1}.\quad(2)$$

根据其推导过程可以看出,系统只需要计算出更新增量来更新校验块.图 2 展示了增量更新的流程,

客户端将新数据发送到存储系统,存储系统需要读出原始数据块,计算出增量,并将增量发送到相应的校验节点.校验块节点再读出原始校验块,根据式(2)计算出最新校验块并持久化,这时存储系统才能将更新的数据覆盖原始数据,响应客户端.和重新编码的更新方式相比,这种方式消除了很多不必要的网络 I/O、硬盘 I/O,很适合被更新的数据较小的负载.但是可以看出,更新流程仍然很复杂,和基于多副本的数据更新性能有较大差距.多副本机制的数据更新过程类似于数据写入过程,只需要将更新数据发送到相应节点写入即可.而纠删码机制的更新过程涉及到数据块和校验块2种数据,会产生额外的网络 I/O、硬盘 I/O 和更多计算.所以,针对基于纠删码机制的分布式存储系统,需要设计一个高效的数据更新方法来改善更新效率.

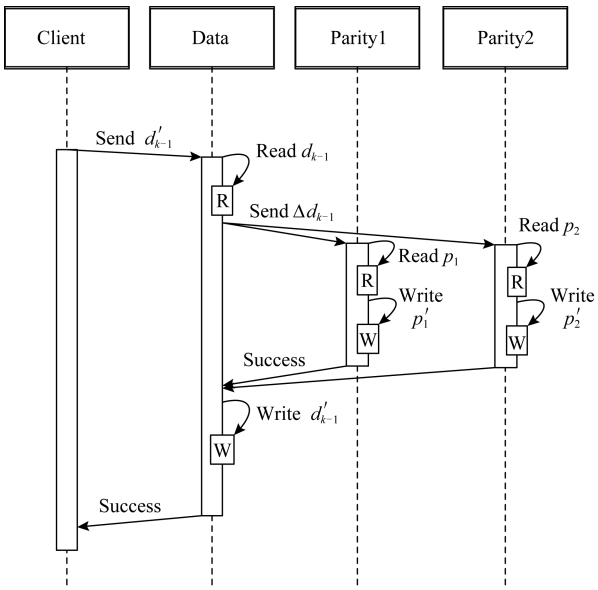

Fig. 2 Incremental update process

图 2 增量更新流程

2 纠删码更新优化的研究进展

纠删码本身并不是针对存储系统设计的容错技术,最早是用在通信行业解决数据传输中的数据损耗问题[14-15],所以分布式存储系统中的一些限制因素比如硬盘 I/O、网络 I/O 并没有被考虑进来,导致纠删码在用于分布式存储系统中时不可避免地产生很多冗余开销.这些开销主要体现在读、写、更新 3 种操作的性能方面.读、写操作主要因为编码、解码所产生的开销,已有相关综述对这些方面做了详细的分析总结,在此不再赘述.本节接下来将专注于更新方面的研究工作,根据这些研究工作的优化角度可将其划分为 3 类:第 1 类是针对硬盘 I/O 开销的优化工作,从图 2 可以看出纠删码更新流程中都有“读后写”操作,这是导致硬盘 I/O 开销比多副本大的一个主要原因,这类研究工作主要针对这一问题进行优化;第 2 类是针对网络传输的优化工作,这类工作主要根据纠删码数据更新的特点减少网络中的数据传输量,从而改善纠删码更新性能;第 3 类工作是针对自身分布式存储系统特点利用纠删码基本更新方法优化系统更新性能,这部分工作虽然改善了自身存储系统的更新性能,但是并没有对纠删码本身的更新性能做出改进优化,本文将这类工作称为系统优化.基于这 3 个分类标准,表 1 对纠删码更新优化工作中具有代表性的工作进行了分类总结,本节接下来会详细介绍.

Table 1 Classification Summary of Erasure Coding Update Related Work

表 1 纠删码更新相关工作分类总结

Classify	Related Work	Characteristic
Disk I/O	PL	Log for parity increments to mitigate write-after-read for parity block.
	PLR	Keep parity increments next to parity blocks to mitigate disk seeks.
	PARIX	Speculative write to mitigate write-after-read for parity increments.
Network Transmission	PUM-P, PDN-P	Disperse calculations to the storage nodes to reduce network traffic.
	T-Update, TA-Update	Use a tree structure to transmit data to eliminate star structure bottlenecks.
	Group-U	Use group method to organize nodes to improve multi-point update.
	CASO	Use data correlation to organize strip to reduce network I/O.
	CAU	Use cross-rack-aware mechanism to mitigate the cross-rack update traffic.
System Optimization	FINGER, SEDP, OptDUM	Perform erasure coding within a block to reduce HDFS/Cloud storage update traffic.
	Cocytus	Use a hybrid replication to eliminate metadata update issues for in-memory KV-store.
	MemEC	Use a all-encoding data model to improve storage efficient for small objects memory store.

2.1　硬盘 I/O 优化

如 1.2 节所述,纠删码主要有重新编码和增量更新 2 种方法.重新编码会消耗大量的网络带宽、硬盘 I/O,不适合于被更新数据长度较小的情况.对 MSR Cambridge traces[16] 数据集分析可以发现平均每个更新操作涉及的数据长度普遍很小,超过 60%的更新操作的数据长度小于 4 KB.Harvard NFS traces 数据集也具有类似的特征[17].所以目前的研究工作基本上都是基于增量更新做优化,尽可能减少更新过程中网络 I/O、硬盘 I/O 的消耗.

增量更新方法一般使用就地更新方式,即新数据块和校验块直接写入旧块所在位置[18-19].相比于重新编码,增量更新方法能够很大程度上减少更新过程中参与的数据量,并且就地更新方式能够有效保证条带内数据的一致性.但是每一次更新操作都需要读取原始数据块和校验块,计算新校验块后方可写入新数据,这些"读后写"操作使得其性能仍然比多副本更新性能差.

为了在更新操作的关键路径上避免冗余的硬盘读开销,存储系统可以利用日志追加形式将新数据和校验块更新增量顺序写入到硬盘内来消除更新路径上原校验块的硬盘读,同时将随机写转换为顺序写,改善更新效率.这种方法被用于很多企业集群的存储系统内,像 GFS[20],Azure.但是这种方式对读操作不友好,每次读取数据时都需要查找随机分布在日志中的所有更新增量得到最新数据,无法顺序读取数据,严重影响读操作的性能[21].此外,当有数据损坏需要执行纠删码恢复操作时,需要先合并数据,削弱了纠删码的恢复性能[22-24].日志追加的形式导致系统中同时存在着旧数据和新数据,也一定程度上降低了存储效率.

为了兼顾数据读操作和更新操作的性能,可以对校验块更新采取日志追加形式,数据块仍然采用就地更新方式.这种方式最先被 Stodolsky 等人[25]用于解决磁盘冗余阵列中的小写问题.他们提出 PL(parity logging)方法,对校验块增量利用日志追加形式写入到硬盘内,对新数据使用就地更新方式,从而消除硬盘查找新数据和读取原校验块时间.纠删码存储系统也可以应用这种方法,但是这种方法仍然存在着恢复性能差的问题,因为恢复时需要查找并读取校验块增量来得到最新校验块信息.为了能进一步改善这个问题,Chan 等人[26]提出 PLR(parity logging with reserved space)方法.这种机制会在硬盘内紧挨着校验块的位置分配一些额外空间来存储未来的校验增量,确保每个校验块和它的更新增量可以被顺序访问,从而消除硬盘查找开销,将随机读转化为顺序读,改善恢复性能.这种方式虽然消除了校验块的"读后写"问题,但是数据块的"读后写"问题仍然存在,成为纠删码更新性能的一个瓶颈.

为了解决数据块的"读后写"问题,进一步减少更新过程中的硬盘 I/O 开销,Li 等人[27]提出 PARIX 方法,是一种预测部分写模式的方法,将"读后写"问题转变为单纯写操作.假设 1 个条带内的某个数据块 d_j 被更新多次,$d_j^{(r)}$ 代表数据块 d_j 第 r 次更新,$p_i^{(r)}$ 代表 $d_j^{(r)}$ 对应的校验块.通过式(3):

$$p_i^{(r)}=\sum_{j=0}^{k-1}G_{ij}d_j^{(r)}=p_i^{(r-1)}+G_{ij}(d_j^{(r)}-d_j^{(r-1)})=$$
$$p_i^{(r-2)}+G_{ij}(d_j^{(r-1)}-d_j^{(r-2)})+G_{ij}(d_j^{(r)}-d_j^{(r-1)})=$$
$$p_i^{(0)}+G_{ij}(d_j^{(r)}-d_j^{(0)}),\tag{3}$$

可以看出,当前校验块 $p_i^{(r)}$ 可以通过 $p_i^{(0)}$,$d_j^{(r)}$,$d_j^{(0)}$ 三个变量计算出来.对于每次写操作,系统只需要预测性地将最新值 $d_j^{(r)}$ 发送到校验节点而不需要先读出 $d_j^{(0)}$,只有当校验节点显式请求 $d_j^{(0)}$ 时(预测失败),对应的数据节点才会读出 $d_j^{(0)}$ 并将其发送到校验节点.在 d_j 的 r 次更新的整个过程中,只有第 1 次被更新时($d_j^{(1)}$)需要读出 $d_j^{(0)}$,显著减少了硬盘读操作,进一步改善了纠删码更新性能.但是在预测失败时需要更多的网络 I/O 和硬盘写操作,使得系统更新性能不稳定.此外,由于校验块的更新仍是以记日志的形式,对纠删码恢复性能还是有一定影响.

上述工作都是致力于改善纠删码更新过程中的"读后写"问题,可以看出相比于基本的纠删码更新方法,这些工作都对纠删码更新的硬盘 I/O 开销做出了一定程度的优化,但是"读后写"问题并没有被彻底解决,和多副本方式的更新性能相比还是有一定差距.如何进一步优化硬盘 I/O 开销,获得和多副本相近的更新性能仍需要研究人员继续探索.

2.2　网络传输优化

2.1 节提到的工作都只是专注于条带内的单个数据块更新操作的优化.当 1 个条带内有多个数据块被更新或者多个条带同时被更新时,这些更新可以共享数据传输或者数据计算来减少网络传输的数据量,改善更新效率.此外,以上提到的优化工作所涉及的更新模式都采用星型结构来完成更新流程,所有被更新的数据节点连接到校验节点,数据节点或者校验节点会成为星型结构的核心,这会成为更新时的计算瓶颈或者网络瓶颈.Zhang 等人[28]从负责计算任务处理的节点位置角度出发,提出 PUM-P,

PDN-P 两种方式.通过将计算子任务尽可能放在距离数据近的节点上处理,来减少在网络中需要传输的数据量,从而改善更新性能.PUM-P 方法将计算任务交给 1 个中心节点 update manager,其负责计算出校验块增量,然后将校验块增量发送到各个校验节点上,由校验节点负责新的校验块的计算.这种方式相比于将所有计算任务交给 update manager 的方式,节省了原始校验块的网络传输.PDN-P 方式将计算校验块增量的任务交给被更新数据块所在节点,减少原始数据块和更新后的数据块的网络传输,进一步减少更新过程中数据的网络传输.

PUM-P 和 PDN-P 更新流程采用的便是典型的星型结构模式,相关研究[29-30]表明即使存在可用带宽更高的空闲链路,星型结构也无法充分利用其来改善数据传输效率.Pei 等人[31]针对此问题提出一种基于树形结构的自顶向下的更新模式 T-Update,来充分利用节点间的网络带宽,改善更新效率.T-Update 通过一种机架感知的树形构建算法 RA-TREE 构建一棵最优更新树,其核心思想是尽可能选择较为临近的节点作为连接节点,以保证节点之间数据传输的高效性.Wang 等人[32]在 T-Update 的基础上设计了一种更加通用的自适应更新模式,TA-Update,来改善 T-Update 的通用性.精心构建的树形结构相比于星型结构能够充分利用网络带宽来改善数据传输效率,但是这些工作仍然只是针对单点数据更新场景下的优化,单点问题依然存在.

Pei 等人[33]针对多点更新提出一种分组流水线的更新模式 Group-U,进一步优化更新效率,减少更新开销.Group-U 将被更新的节点分成若干组,每组选择 1 个数据节点负责组织组内从其他数据节点发往校验节点的数据流,通过这种方式,将数据传输和计算限制在每一个分组内,从而改善更新效率.Huang 等人[34]也采用了分组的思想,通过改善多个数据块更新的并行性来改善分布式内存存储系统的更新延迟.

Shen 等人[35]给出了不一样的解决纠删码多个数据块更新的思路.在真实的存储工作负载中,数据之间存在着关联性,这些关联的数据在访问负载中占有很大比例,通常会被一起访问.如果关联的数据分散在多个不同的条带中,那么对这些数据的 1 次更新操作就需要更新相应条带中的所有校验块.将这些关联的数据放在同一个条带内的话,就可以减少需要更新的校验块数目,降低网络带宽的占用,从而改善更新性能,所以文献[35]提出了一种基于数据关联性的条带组织算法 CASO,通过检测数据块的被访问特征确认数据块的关联性,然后将数据块划分为关联和不关联 2 类.对于关联的数据块,CASO 构造一个关联图来评估它们的相关程度,将条带组织问题转化为一个图分区问题.

在数据中心中,机架与机架之间的通信开销相比于机架内部的通信开销较大,通常 1 个节点的可用跨机架带宽在最坏情况下是机架内部带宽的 1/5 到 1/20,这会成为限制系统性能的一个瓶颈.在纠删码存储系统中,为了增强容错能力,1 个条带内的数据块和校验块通常会放置在不同机架内的节点上,这会导致在更新时产生大量的机架间数据传输.受限的跨机架网络带宽成为影响纠删码更新操作性能的一个瓶颈.针对这个问题,Shen 等人[36]提出一种跨机架感知的更新机制——CAU,来减少跨机架的数据通信量.由式(2)可以看出,系统可以将条带内每个被更新的数据块的数据增量发送到校验节点更新校验块,也可以让条带内 1 个数据节点负责计算出校验增量,然后发送给校验节点更新校验块.这 2 种方式在不同的数据分布和更新负载下会有不同的跨机架数据通信量.所以 CAU 采用选择性的校验块更新方法,基于当前的数据分布和更新模式选择一个最优方式进行更新.此外,CAU 会进行数据重定位,将同一个条带内更新的数据块尽可能部署在 1 个机架内,进一步减少机架间的数据通信.

以上这些研究工作优化了纠删码存储系统更新过程中需要网络传输的数据量,减少了网络带宽的开销,进一步改善了更新性能.如果将分组、跨机架感知、数据关联等特性融合在一起,相信会有一个更好的性能改进.

2.3 系统优化

还有一些研究工作是从分布式存储系统的自身特点出发,结合纠删码更新特点,优化更新性能.

Esmaili 等人[37]在采用 CORE 编码[38]的 HDFS 中实现了增量更新方法,并和重新编码方法做了对比.CORE 是一种将 RS 码和奇偶校验码简单耦合的编码,在数据恢复时优先使用奇偶校验码,改善恢复性能.在数据更新时,需要将更新增量发送到所有校验节点更新校验块.实验结果表明当 1 个条带中有小于一半的数据块被更新时,增量更新性能比重新编码好,反之比重新编码差,所以在实现系统更新策略时需要结合当前更新负载特征,灵活选择合适的更新方法来更新数据.

此外,在 HDFS 中写入的文件会被划分成多个

固定大小的块,这些块通常很大,比如 64 MB 或者 128 MB.这使得更新操作发生时,需要读取的数据量很大,极大地增加了网络 I/O 和硬盘 I/O 开销.Subedi 等人[39]针对此问题提出了一种 FINGER(fine grained erasure coding scheme)的想法来改善 HDFS-RAID 中的写操作和更新操作的性能.为了在不增加元数据大小的同时改善更新性能,他们在块内执行纠删编码而不是块与块之间,减少写和更新过程参与的数据量.Zhou 等人[40]提出的采用数据页(page)粒度的 SEDP 模型和 Zakerinasab 等人[41]提出的基于字节优化的 OptDUM 都是采用了类似的思想来处理云系统内的频繁更新,减少更新过程中的计算开销和网络 I/O、硬盘 I/O 开销,改善更新性能.

Chen 等人[42]观察到分布式内存键值存储系统中大量零散的元数据需要被频繁更新.如果直接将纠删码机制移植到这类系统中,必将带来零散元数据更新开销大的问题.所以他们采用了一种混合存储方式,对元数据采用多副本机制,对数据采用纠删码方式,避免了元数据被频繁更新的开销,同时获得更高的存储效益.但是当数据对象也很小时[43],混合存储方式带来的存储效率收益就很低了,所以 Yiu 等人[12]针对由小对象负载主导的分布式内存存储设计了 MemEC,按照固定块大小的方式来组织对象和元数据,并设计了 2 阶段索引来改善数据访问性能.Chen 等人[44]采用了类似的思想设计了一个分布式文件系统来存储大量的小文件(比如应用中的大量图片).

本节介绍的这些研究工作结合了分布式存储系统的不同特点和纠删码更新特点,采用了不同对策来改善系统性能,但是纠删码本身的更新问题依然存在.

3　现有编码方案的更新性能

目前已有的一些研究工作通常是考虑优化纠删码的一个方面.但是某些场景下,比如热数据的存储,系统对读、写、更新 3 种操作的性能需求都很高.有些工作对纠删码数据恢复性能做了优化工作,但是使得更新操作的流程更加复杂.为了能够在降低热数据存储开销的同时达到和多副本一样的性能,需要从纠删码的编码、解码、更新 3 个方面去考虑优化.本节对目前一些比较有代表性的编码方案进行了分析,这些编码方案大多是为了优化数据恢复性能而专门设计的.为了衡量它们的更新性能,本节主要从网络开销和硬盘 I/O 开销 2 方面与多副本做对比.网络和硬盘 I/O 开销具体指的是对条带内 1 个数据块就地更新时所需要传输和读写的数据量,如表 2 所示.不失一般性,更新模式采用 PUM-P 方式,利用 1 个专门的中心节点 update manager 来处理更新请求.

Table 2　Comparison of Update Cost of Codes

表 2　各编码方案更新开销对比

Codes	Network	Disk I/O	Fault-Tolerance Capacity
3-Replication	3x	3x(W)	2
RS(4,2)	4x	3x(R)+3x(W)	2
LRC(6,2,2)	5x	4x(R)+4x(W)	3
EMSR(4,2,3)	4x	3x(R)+3x(W)	2
Hitchhiker-XOR(4,2)	5x	4x(R)+4x(W)	2

RS 编码是最流行的一种 MDS 码,在很多分布式存储系统中使用,比如 Atlas[45],Colossus[46],f4.在 RS(4,2)编码中,更新条带内的 1 个数据块,通常采用增量更新方式来更新校验块.update manager 从相应的数据节点读出原数据块计算数据增量,将增量发送到 2 个校验节点,每个校验节点分别读出原校验块,计算最新校验块后写到硬盘内,整个过程产生 3 个编码块的硬盘读和写以及 4 个编码块的网络传输,比 3 副本多了 3 个编码块的硬盘读操作和 1 个数据块的网络传输.

由于 RS 编码将条带内 k 个数据块进行线性结合生成校验块,数据恢复时需要读取剩余的 k 个块才能执行解码操作.为了减少数据恢复过程中需要的数据量,局部修复码(local reconstruction codes,LRC)[47]被提出.LRC 在 RS 编码的基础上引入局部校验块,在恢复数据时优先使用局部校验块,从而减少恢复过程需要的数据量.

图 3 展示了 LRC(6,2,2)编码,其中 $k=6$ 代表 6 个数据块(d_0,d_1,d_2,d_3,d_4,d_5),$l=2$ 代表 2 个局部校验块(p_0,p_1),$r=2$ 代表 2 个全局校验块(p_2,p_3).6 个数据块被分为 2 组,分别对分组内数据块线性结合得到局部校验块 p_0(由第 1 组 d_0,d_1,d_2 线性结合得到),p_1(由第 2 组 d_3,d_4,d_5 这 3 个数据块线性结合得到).p_2,p_3 由所有数据块计算得到.在修复时优先使用局部校验块,从而减少数据恢复需要的数据量,改善纠删码恢复性能.当某个数据块被更新时,需要更新对应的局部校验块和全局

校验块,其产生的开销和 RS(6,3)编码一样.由于 LRC 只需要在标准 RS 编码的基础上引入局部校验块,实现简单,所以已被 Azure[48],Facebook[49] 使用.需要注意的是,LRC 编码本身不是 MDS 码,数据恢复只能访问固定的节点集合.相比于 RS 编码,LRC 编码引入的局部校验块虽然可以降低恢复开销,但是当不同分组内的数据块被同时更新时,会引入额外的硬盘 I/O 和网络数据传输开销.

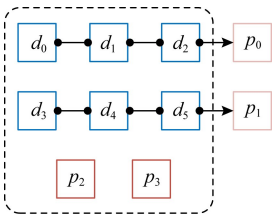

Fig. 3 LRC(6,2,2) code

图 3 LRC(6,2,2)编码

再生码(regenerating codes)[50-51] RGC(n,k,x)是另一种减少纠删码恢复过程中网络数据传输开销的编码,其能够通过任意 k 个块恢复整个条带的数据,任意 $x(k\leqslant x\leqslant n-1)$ 个节点能够恢复单个节点故障.虽然 RGC 编码需要更多的节点参与数据恢复,但是总的数据传输量更少,从而减少网络开销.由于在恢复过程中需要从更多的节点上读取数据,虽然总的网络传输的数据量减少,但是显著增加了硬盘 I/O 开销和计算开销.RGC 编码主要分为 2 类:一类是 MSR 码,与 RS 编码相比有同等存储开销,具有更低的网络开销;另一类是 MBR 码,通过存储更多冗余数据获得更低的网络开销.由于 MBR 的存储利用率较低,所以对其不做考虑.EMSR(4,2,3)码[52]是一种 MSR 码,利用干扰对齐技术同时消去多个不需要的变量,如图 4 所示,每个数据块划分为 2 个子块,每个校验块也包含 2 个校验子块.如果数据块 d_0 损坏,d_1,p_0,p_1 三个块分别将子块求和合并成单个子块,然后发送到修复节点,修复节点通过对接收到的 3 个子块进行计算得到 d_0.整个修复过程中,网络中只需要传输 3 个子块,若是 RS(4,2)编码,则需要传输 4 个子块.若 d_0 块被更新,与其子块 a_0,b_0 相关的所有校验子块都需要被更新.虽然 EMSR(4,2,3)在更新过程中传输和读写的数据量和 RS(4,2)相同,但是由于在校验节点上更新校验子块的操作更复杂,导致其比 RS(4,2)编码消耗更多计算资源和硬盘 I/O 资源.

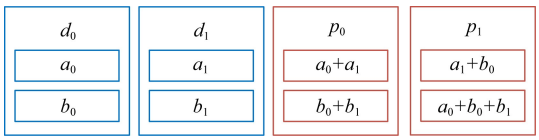

Fig. 4 EMSR(4,2,3) code

图 4 EMSR(4,2,3)编码

Hitchhiker-XOR 码[18]是在现有的 RS 码基础上对其改造,将 RS 编码中的 2 个条带组合成 1 个更大的条带.2 个子条带之间通过一些冗余信息产生关联,从而减少恢复时所需要的数据量.如图 5 所示,Hitchhiker-XOR(4,2)码先对每个子条带进行 RS(4,2)编码,然后将第 2 个子条带内的第 2 个校验块与第 1 个子条带内的数据块进行异或运算.若 a_0,b_0 两个数据块出现损坏,可以先用 b_1,$f_1(b)$ 恢复出 b_0,然后用 b_0 和 b_1 计算出 $f_2(b)$,最后利用 $f_2(b)$,a_1 和 $f_2(b)+a_0+a_1$ 恢复出 a_0.整个过程需要 4 个块参与,这虽然与 RS(4,2)编码相同,但是这仅限于此实例.当子条带内块数量较大时,比如说 10 个数据块、4 个校验块时,恢复 a_0,b_0 只需要 13 个块,相比于 RS(14,10),数据量减少了 35%.由于一个子条带内的某些数据块信息会关联到另一个子条带内的校验块,所以更新开销会相应增大.比如说更新 a_0,需要更新 $f_1(a)$,$f_2(a)$,$f_2(b)+a_0+a_1$ 三个校验块,更新 b_0 则只需要更新 $f_1(b)$,$f_2(b)+a_0+a_1$ 两个校验块.虽然 Hitchhiker-XOR 码减少了数据恢复所需要的数据量,降低了网络传输开销和硬盘 I/O 开销,但是却导致了更新性能的削弱.

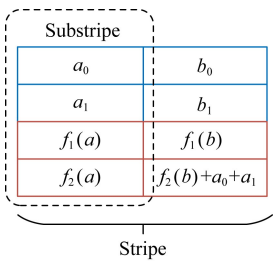

Fig. 5 Hitchhiker-XOR(4,2) code

图 5 Hitchhiker-XOR(4,2)编码

因为多副本机制的数据更新过程类似于数据写入过程,只需要将更新数据发送到相应节点写入即可.以 3 副本为例,更新过程只需要 3x 的网络传输和硬盘写操作.表 2 中对比了各编码方案的更新开销,可以看出,优化了纠删码恢复性能的各类编码都导致了纠删码更新性能的削弱,使得纠删码更新性

能比多副本机制更差.由于目前纠删码的读、写、更新3种操作的性能都弱于多副本,使得热数据存储这种对性能要求较高的场景无法使用纠删码容错机制,只能使用多副本来保证数据高可用,导致热数据的存储开销很大.

4　未来研究趋势

从第3节的分析可以看出,目前纠删码更新操作的性能仍然无法与多副本机制相媲美,所以纠删码大多用于冷数据或者温数据这种一次写入后很少被访问和更新的场景,需要频繁访问和更新的热数据仍然使用多副本机制来存储.如何应用纠删码改善热数据存储开销的同时,满足系统性能需求是一个值得深入研究的方向.

伴随着非易失性内存技术的不断发展,计算机计算能力和存储性能得到了质的提升.比如新兴的电阻式随机存取存储器(RRAM)[53-54]不仅可以提供非易失性存储,还可以提供矩阵向量乘法的内在逻辑.磁畴壁存储器(domain-wall memory)[55],又被称为赛道存储器(racetrack memory)[56],它在提供非易失性特性的同时,能够提供和DRAM非常接近的访问延迟、密度和功耗.笔者现有工作[57-58]利用其构建了内存计算架构极大地改善了系统的吞吐量、延迟和功耗.可以看出NVM的特性能够很好地消除纠删码更新过程中的硬盘I/O开销,降低纠删码存储系统的计算开销和功耗,使纠删码用于热数据存储成为可能.但是NVM本身也具有一些缺点,比如在数据一致性方面,当节点因故障重启后,NVM内部之前的数据还在,这些数据中有可能存在脏数据,这要求使用NVM的系统需要专门的管理机制来管理数据.再加上纠删码自身的编码规则和特点,这势必会增加系统设计的复杂度.如何结合NVM新硬件和纠删码自身特点,构建一个安全、高效的分布式存储系统是一个值得深入研究的方向.

目前很多分布式存储系统,尤其是云系统都是采用追加式更新的方式来更新数据.更新数据以新数据的形式写入,原有数据被标记为无效数据.这样将随机写转化为顺序写操作,来优化写的性能.这种日志结构存储方式避免了纠删码更新所带来的问题,但是引入了一个新的问题,就是无效数据所占用的空间回收问题.为了释放掉无效数据占用的空间,垃圾回收功能被引入进来.垃圾回收是一个代价比较昂贵的操作,需要消耗大量的计算机资源,所以垃圾回收功能通常在节点空闲时被触发,避免对系统性能产生影响.但是在热数据的存储场景中,更新操作频繁,如果触发垃圾回收的周期太长,就会使得大量无效数据聚集,降低存储效率,频繁触发垃圾回收又会影响系统性能.而且,采用纠删码机制的日志结构存储系统中的垃圾回收和传统的基于多副本机制的日志结构存储系统的垃圾回收有所不同.为了方便描述,本文将前者称为ECGC(erasure coding garbage collection),后者称为MRGC(multiple replication garbage collection).MRGC和ECGC主要有3点不同:

1) 在MRGC中,垃圾回收的粒度通常以段(segment)或者块(block)为单位.segment内的有效数据拷贝到新的位置后即可回收旧segment.但是在ECGC中,垃圾回收需要以条带为粒度.由于1个条带内数据块和校验块之间存在着关联性,为了保证条带内数据的高可用,只有当条带内所有有效的数据都安全地拷贝到新的位置后才可以回收旧条带内的数据.

2) 条带内的数据块通常位于不同节点,如图6所示,2个条带内的数据块和校验块均匀分布在集群内的每个节点上,如果对这2个条带内的无效数据进行回收,就需要将条带内的每个有效数据块从相应节点上读出,重新编码持久化到不同节点,然后才可以删除原来的条带.这个过程涉及到集群内的每个节点,如果不仔细设计垃圾回收机制,ECGC有可能影响到整个集群处理上层请求的能力.MRGC则不存在这个问题.

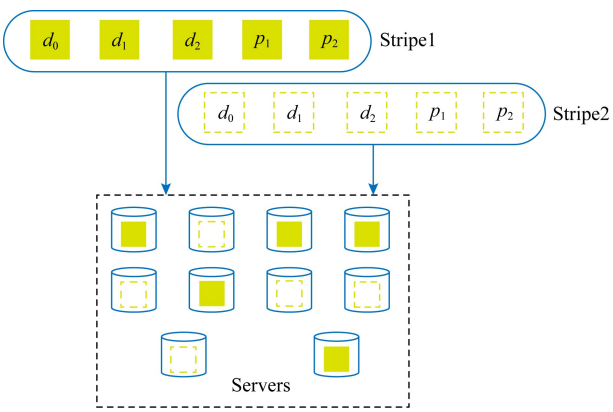

Fig. 6　Data stripe distribution in erasure-coded storage systems

图6　纠删码存储系统数据条带分布

3) MRGC中1次回收的有效数据没有长度限制,但是在ECGC中1次回收的有效数据的长度通常

需要 k 个数据块的整数倍,因为只有满足(n,k)编码的长度要求时才能进行编码并持久化.这要求 ECGC 在选择条带进行垃圾回收时需要考虑到该特点.

为了不影响系统的延迟和吞吐量,使用纠删码机制的日志结构存储系统需要一个针对纠删码优化的高效的垃圾回收机制,但是目前这方面的研究工作非常少.Li 等人[59]设计了一个启发式垃圾回收算法,其基本想法是将无效数据占用比例最大的条带内的有效数据迁移到无效数据占用比例最小的条带内,尽可能减少垃圾回收过程中数据块的移动数量.为了进一步减少数据块的移动,改善垃圾回收效率,他们利用工作负载中键倾斜分布的特点,对客户端发来的数据按照热度进行排序,将热度相近的数据放置在相同的条带内.因为热度相同的数据更容易被一起访问,被更新时无效数据就会更多地聚集在 1 个条带内.如图 7 所示,条带 1 内有 2 个无效数据块,条带 2 内有 1 个无效数据块,将条带 1 内剩余的 1 个有效数据块迁移到条带 2 内,然后即可删除条带 1.启发式垃圾回收一定程度上减少了垃圾回收过程中数据的传输和读写,但是垃圾回收过程中所产生的数据迁移,相当于触发了更多的数据更新请求,这些请求会和上层应用请求交错在一起,占用系统网络带宽、硬盘 I/O 等资源,使系统吞吐量和延迟产生波动.

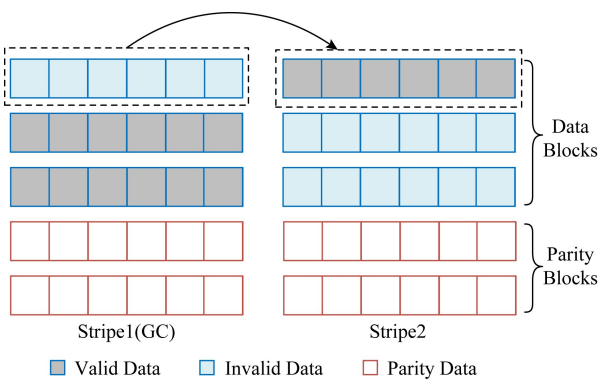

Fig. 7 Heuristic garbage collection

图 7 启发式垃圾回收

以日志结构方式来更新数据消除了纠删码更新所导致的性能削弱,但是纠删码条带内部数据一致性的特征导致垃圾回收活动有可能波及到存储系统内的每个节点,导致系统性能波动,尤其是在热数据存储场景下.针对纠删码的特点设计一个高效的垃圾回收机制,在不影响系统响应上层请求能力的同时,降低存储开销值得深入探讨.

5 总 结

随着数据量的与日俱增,分布式存储系统的开销越来越大,多副本容错机制使得这种情况更加严重.伴随着硬件技术的不断发展,越来越多的存储系统将目光从多副本机制转向纠删码,但是纠删码本身复杂的规则导致其性能较差.本文专注于纠删码存储系统中的数据更新操作,从硬盘 I/O、网络传输、系统优化 3 个方面对相关工作进行了总结分析,并对目前具有代表性的编码方案的更新性能做了对比分析,最后展望了未来研究趋势.与多副本机制相比,现有纠删码更新方案会产生更多网络 I/O、硬盘 I/O 开销,导致纠删码存储系统的更新操作的性能较差.如何结合新硬件技术特点,从纠删码编码规则本身和系统架构角度对其优化,还存在巨大技术挑战.目前较优的一种避免纠删码更新开销的方案是采用日志结构追加式更新,但是如何设计一个高效的垃圾回收机制来应对热数据这种频繁更新的场景仍需探索.

参 考 文 献

[1] Rashmi K V, Shah N B, Gu Dikang, et al. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster [C/OL] //Proc of the 5th Workshop on Hot Topics in Storage and File Systems. Berkeley, CA: USENIX Association, 2013 [2019-07-10]. https://www.usenix.org/system/files/conference/hotstorage13/hotstorage13-rashmi.pdf

[2] Annamalai M, Ravichandran K, Srinivas H, et al. Sharding the shards: Managing datastore locality at scale with Akkio [C] //Proc of the 13th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2018: 445-460

[3] Borthakur D. HDFS architecture guide [OL]. [2019-06-29]. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.pdf

[4] Calder B, Wang Ju, Ogus A, et al. Windows Azure storage: A highly available cloud storage service with strong consistency [C] //Proc of the 23rd ACM Symp on Operating Systems Principles. New York: ACM, 2011: 143-157

[5] Weil S A, Brandt S A, Miller E L, et al. Ceph: A scalable, high-performance distributed file system [C] //Proc of the 7th Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2006: 307-320

[6] Muralidhar S, Lloyd W, Roy S, et al. f4: Facebook's warm BLOB storage system [C] //Proc of the 11th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2014: 383-398

[7] Wang Yijie, Xu Fangliang, Pei Xiaoqiang. Research on erasure code-based fault-tolerant technology for distributed storage [J]. Chinese Journal of Computers, 2017, 40(1): 236-255 (in Chinese)
(王意洁, 许方亮, 裴晓强. 分布式存储中的纠删码容错技术研究[J], 计算机学报, 2017, 40(1): 236-255)

[8] Li Jun, Li Baochun. Erasure coding for cloud storage systems: A survey [J]. Tsinghua Science and Technology, 2013, 18(3): 259-272

[9] Luo Xianghong, Shu Jiwu. Summary of research for erasure code in storage system [J]. Journal of Computer Research and Development, 2012, 49(1): 1-11 (in Chinese)
(罗象宏, 舒继武. 存储系统中的纠删码研究综述[J]. 计算机研究与发展, 2012, 49(1): 1-11)

[10] Yang Songlin, Zhang Guangyan. Review of data recovery in storage systems based on erasure codes [J]. Journal of Frontiers of Computer Science and Technology, 2017, 11(10): 1531-1544 (in Chinese)
(杨松霖, 张广艳. 纠删码存储系统中数据修复方法综述[J]. 计算机科学与探索, 2017, 11(10): 1531-1544)

[11] Plank J S, Luo Jianqiang, Schuman C D, et al. A performance evaluation and examination of open-source erasure coding libraries for storage [C] //Proc of the 7th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2009: 253-265

[12] Yiu M M T, Chan H H W, Lee P P C. Erasure coding for small objects in in-memory kv storage [C/OL] //Proc of the 10th ACM Int Systems and Storage Conf. New York: ACM, 2017 [2019-07-10]. https://arxiv.org/pdf/1701.08084.pdf

[13] Reed I S, Solomon G. Polynomial codes over certain finite fields [J]. Journal of the Society for Industrial and Applied Mathematics, 1960, 8(2): 300-304

[14] Luby M G, Mitzenmacher M, Shokrollahi M A, et al. Efficient erasure correcting codes [J]. IEEE Transactions on Information Theory, 2001, 47(2): 569-584

[15] Widmer J, Le Boudec J Y. Network coding for efficient communication in extreme networks [C] //Proc of the 2005 ACM SIGCOMM Workshop on Delay-Tolerant Networking. New York: ACM, 2005: 284-291

[16] Narayanan D, Donnelly A, Rowstron A. Write off-loading: Practical power management for enterprise storage [C] // Proc of the 6th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2008: 253-267

[17] Ellard D J, Seltzer M I. Trace-based analyses and optimizations for network storage servers [D]. Cambridge, MA: Harvard University, 2004

[18] Rashmi K V, Shah N B, Gu Dikang, et al. A Hitchhiker's guide to fast and efficient data reconstruction in erasure-coded data centers [J]. ACM SIGCOMM Computer Communication Review, 2015, 44(4): 331-342

[19] Rawat A S, Vishwanath S, Bhowmick A, et al. Update efficient codes for distributed storage [C] //Proc of the 2011 IEEE Int Symp on Information Theory. Piscataway, NJ: IEEE, 2011: 1457-1461

[20] Ghemawat S, Gobioff H, Leung S T. The Google file system [C] //Proc of the 19th ACM Symp on Operating Systems Principles. New York: ACM, 2003: 29-43

[21] Matthews J N, Roselli D, Costello A M, et al. Improving the performance of log-structured file systems with adaptive methods [C] //Proc of the 16th ACM Symp on Operating Systems Principles. New York: ACM, 1997: 238-251

[22] Aguilera M K, Janakiraman R, Xu Lihao. Using erasure codes efficiently for storage in a distributed system [C] //Proc of the 2005 Int Conf on Dependable Systems and Networks. Piscataway, NJ: IEEE, 2005: 336-345

[23] Mazumdar A, Chandar V, Wornell G W. Update-efficiency and local repairability limits for capacity approaching codes [J]. IEEE Journal on Selected Areas in Communications, 2014, 32(5): 976-988

[24] Aguilera M K, Janakiraman R, Xu Lihao. On the erasure recoverability of mds codes under concurrent updates [C] // Proc of the Int Symp on Information Theory. Piscataway, NJ: IEEE, 2005: 1358-1362

[25] Stodolsky D, Gibson G, Holland M. Parity logging overcoming the small write problem in redundant disk arrays [C] //Proc of the 20th Annual Int Symp on Computer Architecture. New York: ACM, 1993: 64-75

[26] Chan J C W, Ding Qian, Lee P P C, et al. Parity logging with reserved space: Towards efficient updates and recovery in erasure-coded clustered storage [C] //Proc of the 12th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2014: 163-176

[27] Li Huiba, Zhang Yiming, Zhang Zhiming, et al. PARIX: Speculative partial writes in erasure-coded systems [C] //Proc of the 2017 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2017: 581-587

[28] Zhang Fenghao, Huang Jianzhong, Xie Changsheng. Two efficient partial-updating schemes for erasure-coded storage clusters [C] //Proc of the 2012 IEEE Int Conf on Networking, Architecture, and Storage. Piscataway, NJ: IEEE, 2012: 21-30

[29] Li Jun, Yang Shuang, Wang Xin, et al. Tree-structured data regeneration with network coding in distributed storage systems [C/OL] //Proc of the 17th Int Workshop on Quality of Service. Piscataway, NJ: IEEE, 2009 [2019-07-10]. https://www.eecg.utoronto.ca/~bli/papers/junli-iwqos09.pdf

[30] Li Jun, Yang Shuang, Wang Xin, et al. Tree-structured data regeneration in distributed storage systems with regenerating codes [C] //Proc of the 29th Conf on Information Communications. Piscataway, NJ: IEEE, 2010: 2892-2900

[31] Pei Xiaoqiang, Wang Yijie, Ma Xingkong, et al. T-Update: A tree-structured update scheme with top-down transmission in erasure-coded systems [C/OL] //Proc of the 35th Annual IEEE Int Conf on Computer Communications. Piscataway, NJ: IEEE, 2016 [2019-07-10]. https://ieeexplore.ieee.org/document/7524347

[32] Wang Yijie, Pei Xiaoqiang, Ma Xingkong, et al. TA-Update: An adaptive update scheme with tree-structured transmission in erasure-coded storage systems [J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 29(8): 1893-1906

[33] Pei Xiaoqiang, Wang Yijie, Ma Xingkong, et al. Efficient in-place update with grouped and pipelined data transmission in erasure-coded storage systems [J]. Future Generation Computer Systems, 2017, 69(C): 24-40

[34] Huang Jianzhong, Xia Jie, Qin Xiao, et al. Optimization of small updates for erasure-coded in-memory stores [J]. The Computer Journal, 2019, 62(6): 869-883

[35] Shen Zhirong, Lee P P C, Shu Jiwu, et al. Correlation-aware stripe organization for efficient writes in erasure-coded storage systems [C] //Proc of the 2017 IEEE Symp on Reliable Distributed Systems. Piscataway, NJ: IEEE, 2017: 134-143

[36] Shen Zhirong, Lee P P C. Cross-rack-aware updates in erasure-coded data centers [C] //Proc of the 47th Int Conf on Parallel Processing. New York: ACM, 2018: 1-10

[37] Esmaili K S, Chiniah A, Datta A. Efficient updates in cross-object erasure coded storage systems [C] //Proc of the 2013 IEEE Int Conf on Big Data. Piscataway, NJ: IEEE, 2013: 28-32

[38] Esmaili K S, Pamies-Juarez L, Datta A. CORE: Cross-object redundancy for efficient data repair in storage systems [C] //Proc of the 2013 IEEE Int Conf on Big Data. Piscataway, NJ: IEEE, 2013: 246-254

[39] Subedi P, Huang Ping, Young B, et al. FINGER: A novel erasure coding scheme using fine granularity blocks to improve hadoop write and update performance [C] //Proc of the 2015 IEEE Int Conf on Networking, Architecture and Storage. Piscataway, NJ: IEEE, 2015: 255-264

[40] Zhou Hang, Yang Yahui, Li Weiping. An erasure code-based approach to improve data recovery and update capability [C/OL] //Proc of the 2018 Int Conf on Mechanical, Electronic, Control and Automation Engineering. Paris: Atlantis Press, 2018 [2019-07-10]. https://www.atlantis-press.com/proceedings/mecae-18/25893742

[41] Zakerinasab M R, Wang M. Practical network coding for the update problem in cloud storage systems [J]. IEEE Transactions on Network and Service Management, 2017, 14(2): 386-400

[42] Chen Haibo, Zhang Heng, Dong Mingkai, et al. Efficient and available in-memory KV-store with hybrid erasure coding and replication [C] //Proc of the 14th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2016: 167-180

[43] Atikoglu B, Xu Yuehai, Frachtenberg E, et al. Workload analysis of a large-scale key-value store [C] //Proc of the 12th ACM SIGMETRICS/PERFORMANCE Joint Int Conf on Measurement and Modeling of Computer Systems. New York: ACM, 2012: 53-64

[44] Chen Xinhai, Liu Jie, Xie Peizhen. Erasure code of small file in a distributed file system [C] //Proc of the 2017 IEEE Int Conf on Computer and Communications. Piscataway, NJ: IEEE, 2017: 2549-2554

[45] Lai Chunbo, Jiang Song, Yang Liqiong, et al. Atlas: Baidu's key-value storage system for cloud data [C/OL] //Proc of the 31st Symp on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE, 2015 [2019-07-10]. http://ranger.uta.edu/~sjiang/pubs/papers/lai15-atlas.pdf

[46] Fikes A. Colossus, successor to Google file system [OL]. [2019-06-29]. https://cloud.google.com/files/storage_architecture_and_challenges.pdf

[47] Huang Cheng, Chen Minghua, Li Jin. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems [C] //Proc of the 6th IEEE Int Symp on Network Computing and Applications. Piscataway, NJ: IEEE, 2007: 79-86

[48] Huang Cheng, Simitci H, Xu Yikang, et al. Erasure coding in windows Azure storage [C] //Proc of the 2012 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2012: 15-26

[49] Sathiamoorthy M, Asteris M, Papailiopoulos D, et al. Xoring elephants: Novel erasure codes for big data [J]. Proceedings of the VLDB Endowment, 2013, 6(5): 325-336

[50] Jiekak S, Kermarrec A M, Le Scouarnec N, et al. Regenerating codes: A system perspective [J]. ACM SIGOPS Operating Systems Review, 2013, 47(2): 23-32

[51] Li Jie, Tang Xiaohu, Parampalli U. A framework of constructions of minimal storage regenerating codes with the optimal access/update property [J]. IEEE Transactions on Information Theory, 2015, 61(4): 1920-1932

[52] Shah N B, Rashmi K V, Kumar P V, et al. Interference alignment in regenerating codes for distributed storage: Necessity and code constructions [J]. IEEE Transactions on Information Theory, 2011, 58(4): 2134-2158

[53] Akinaga H, Shima H. Resistive random access memory (ReRAM) based on metal oxides [J]. Proceedings of the IEEE, 2010, 98(12): 2237-2251

[54] Kim K H, Gaba S, Wheeler D, et al. A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications [J]. Nano Letters, 2012, 12(1): 389-395

[55] Parkin S S P, Hayashi M, Thomas L. Magnetic domain-wall racetrack memory [J]. Science, 2008, 320(5873): 190-194

[56] Thomas L, Yang S H, Ryu K S, et al. Racetrack memory: A high-performance, low-cost, non-volatile memory based on magnetic domain walls [C/OL] //Proc of the 2011 Int Electron Devices Meeting. Piscataway, NJ: IEEE, 2011 [2019-07-10]. https://ieeexplore.ieee.org/document/6131603

[57] Ni Leibin, Wang Yuhao, Yu Hao, et al. An energy-efficient matrix multiplication accelerator by distributed in-memory computing on binary RRAM crossbar [C] //Proc of the 21st Asia and South Pacific Design Automation Conf. Piscataway, NJ: IEEE, 2016: 280-285

[58] Wang Yuhao, Yu Hao, Ni Leibin, et al. An energy-efficient nonvolatile in-memory computing architecture for extreme learning machine by domain-wall nanowire devices [J]. IEEE Transactions on Nanotechnology, 2015, 14(6): 998-1012

[59] Li Shenglong, Zhang Quanlu, Yang Zhi, et al. BCStore: Bandwidth-efficient in-memory KV-store with batch coding [C/OL] //Proc of the 33rd Int Conf on Massive Storage Systems and Technology. 2017 [2019-06-15]. https://storageconference.us/2017/Papers/BCStore-Bandwid-thEfficientKV-Store.pdf

Zhang Yao, born in 1993. Master. His main research interests include parallel and distributed systems.

Chu Jiajia, born in 1993. PhD. Her main research interests include parallel and distributed systems.

Weng Chuliang, born in 1976. PhD, professor, PhD supervisor. Member of CCF. His main research interests include parallel and distributed systems, storage systems, OS, and system security.