ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2017/11(10)-1531-14
doi: 10.3778/j.issn.1673-9418.1701044

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-89056056

纠删码存储系统中数据修复方法综述*

杨松霖,张广艳+
清华大学 计算机科学与技术系,北京 100084

Review of Data Recovery in Storage Systems Based on Erasure Codes*

YANG Songlin, ZHANG Guangyan+
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
+ Corresponding author: E-mail: gyzh@tsinghua.edu.cn

YANG Songlin, ZHANG Guangyan. Review of data recovery in storage systems based on erasure codes. Journal of Frontiers of Computer Science and Technology, 2017, 11(10): 1531-1544.

Abstract: Erasure codes have the advantage of low storage overhead. However, they also have the drawbacks of long recovery time and high impact on application performance. This paper presents the computation model of the time for data recovery with erasure codes, and identifies the key factors that affect the recovery performance. Thereafter, this paper chooses the computation overhead, read/write overhead, and transmission overhead as the evaluation criterion for the recovery performance. In addition, this paper analyzes how the latest efforts in this field reduce overheads from the aspects of computation, read/write, and transmission. Finally, this paper compares existing coding schemes from the aspects of recovery performance, reliability, as well as storage overhead, and then points out the future research directions.

Key words: erasure codes; multiple replicas; data recovery; performance improvement

摘 要: 纠删码技术具有存储开销低的优势,然而在进行数据修复时面临修复时间长和对前端应用性能影响高的缺陷。给出纠删码技术中数据修复完成时间的计算模型,指出影响修复性能的关键因素,进而选取计算开销、读写开销、传输开销作为修复性能的评价标准;分析了现有研究工作如何降低计算、读写和传输3种开销,重点讨论了其关键性技术的优缺点;最后从修复性能、可靠性、存储开销等方面对现有编码方案进行对比,

* The National Natural Science Foundation of China under Grant Nos. 61672315, F020803 (国家自然科学基金); the National Basic Research Program of China under Grant No. 2014CB340402 (国家重点基础研究发展计划(973计划)).
Received 2017-01, Accepted 2017-06.
CNKI网络优先出版: 2017-06-21, http://kns.cnki.net/kcms/detail/11.5602.TP.20170621.1105.002.html

并指出未来可能的研究方向。

关键词:纠删码;多副本;数据修复;性能优化

文献标志码:A **中图分类号**:TP393

1 引言

随着计算机技术的发展与互联网的普及,数据呈现爆发式增长,有研究表明,过去两年里产生的数据已经占世界数据总量的90%[1]。数据成为当今企业的核心资产,企业面临着数据丢失和数据不可用的风险,如何解决这些问题,对现代存储系统提出了巨大的挑战。多副本技术通过将数据的多个冗余副本分布在不同机器上,当存储节点发生故障时,自动将服务切换到其他副本,从而实现数据的高可靠和高可用。然而随着数据的持续增长,在PB级别的数据中心,多副本技术会引入极大的存储开销。比如,现有的分布式存储系统,如HDFS(Hadoop distributed file system)[2]、Ceph[3],普遍采用三副本的配置,这将占用原始数据的3倍存储空间。

面对如此情况,纠删码技术因其低存储开销的特点近年来受到越来越多的关注。其中,RS编码(Reed-Solomon code)[4]是目前被广泛使用的纠删码方案,它将 k 个数据块按照一定的编码规则,生成 m 个校验块,对于这 $k+m$ 个编码块,其编码性质保证通过任意的 k 个编码块均能重建出所有数据。以RS(4, 2)编码为例,它只需占用1.5倍的存储空间,就具有与三副本技术相同的容错能力。

然而当数据丢失时,纠删码技术数据修复的速度没有多副本快。为了修复丢失的数据,多副本技术只需拷贝对应数据的冗余副本。而纠删码需要读取 k 块数据,通过计算重建丢失的数据,相比之下,占用了更多的磁盘带宽和网络带宽,也引入了额外的计算开销。这不仅导致修复时间过长,而且也对前台应用带来干扰。在大规模集群环境下,磁盘、服务器和网络的错误已经成为常态,如在Facebook的数据中心平均每天发生50起机器不可用事件[5]。在这种情况下,为了保持数据可靠性,存储系统需要频繁进行修复操作,进一步加重了纠删码修复给系统带来的压力。

针对纠删码修复存在的性能缺陷,近年来涌现出很多理论设计和工程实现的工作。目前,国内外存在少量纠删码技术的研究综述[6-9],其中文献[8]主要关注编码方案上的进展,文献[9]主要围绕编码实现、数据修复和数据更新等方面的最新研究进展。本文将只聚焦于纠删码数据修复,从系统性能优化的一般性原则出发,分析影响纠删码数据修复性能的关键因素,如图1所示,围绕读写、计算、传输3个层级,对现有的优化技术进行讨论。其中,从减少计算量和加速计算执行两个角度介绍了计算优化方面的工作;从降低I/O总量和提高I/O并行度两个角度介绍了读写优化方面的工作;从减少单个节点的传输量和减少参与修复节点数量两个角度介绍了传输优化方面的工作。最后,对比分析了针对修复性能优化的编码方案,并指出未来可能的研究方向。

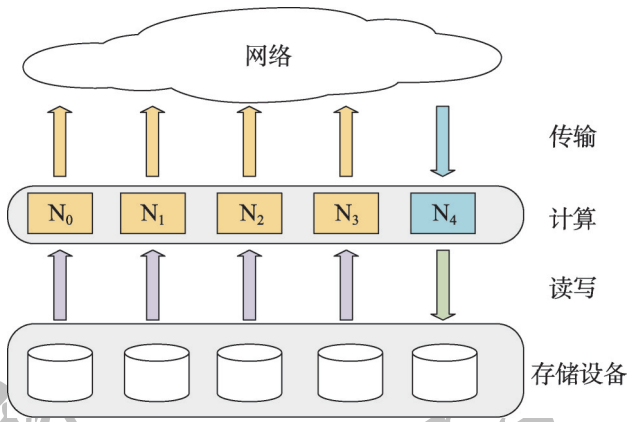


Fig.1 Hierarchy of recovery performance optimization

图1 修复性能优化层级图

本文组织结构如下:第2章介绍纠删码技术的相关背景;第3章论述从计算角度优化纠删码修复性能的关键技术;第4章论述从读写角度优化纠删码修复性能的关键技术;第5章论述从传输角度优化纠删码修复性能的关键技术;第6章讨论评价了现有的编码方案;第7章总结全文,并展望未来研究方向。

2 基础与关键因素

以下介绍纠删码的相关概念，阐述纠删码的修复过程，重点分析影响纠删码的数据修复性能的关键因素。

2.1 基本概念

为了便于理解，对本文出现的相关概念给出如下说明。

（1）数据块：原始用户数据被系统划分形成的最小编码单位。

（2）校验块：由数据块经过相关运算得到的校验数据。

（3）条带：多个数据块与其对应的校验块构成的冗余集合，如果一定数目的编码块丢失，可以通过对所在条带中剩余编码块进行运算而重新生成。

（4）容错能力：编码方案中最大允许出错的节点数，当出错的节点数超过其容错能力时，将无法修复丢失的数据。

（5）最大距离可分码（maximum distance separable，MDS）：满足 Singleton 边界[10]的线性编码方式。与其他编码相比，它在同等容错能力的情况下拥有最低的存储开销。

（6）参与节点：参与数据修复的节点。它需要读取本地数据来参与数据重建。

（7）修复节点：重建丢失数据的节点。它通过从参与修复的节点中收集所需数据，计算得到丢失的数据。

（8）存储利用率：原始用户数据量与编码后所占存储空间的比值。由于提供容错保护需要存储冗余信息，从而纠删码的存储利用率总是小于1。

2.2 编码修复过程

在现有的分布式存储系统中，RS编码是一种常见的纠删码编码方案，它支持任意数目的数据块与校验块参数配置，其生成矩阵采用范德蒙矩阵或柯西矩阵进行构造。如图2所示，RS编码利用生成矩阵 $\boldsymbol{G}$ 与原始数据向量的乘积，生成整个条带的数据。当数据块出现丢失，首先从生成矩阵中去掉丢失数据块对应的行，构造出残余矩阵；然后求解残余矩阵的逆矩阵，并取出丢失数据块所对应的行向量；

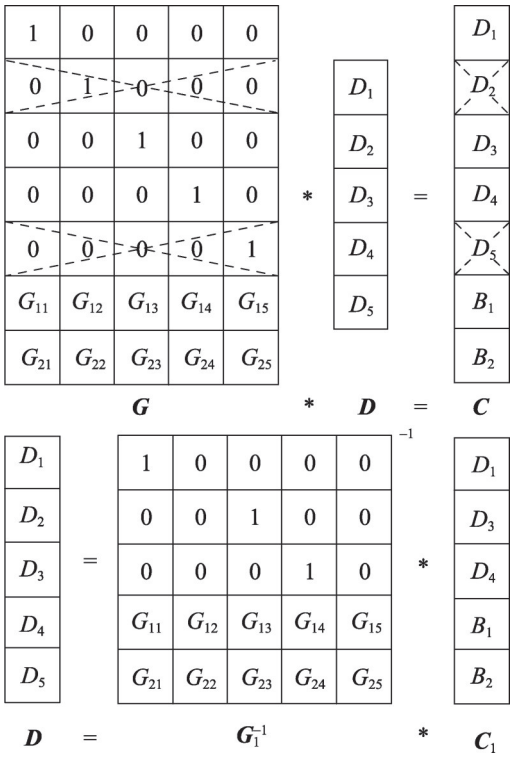


Fig.2 Encoding and recovery process of RS code

图2 RS编码的编码和修复过程

最后将行向量与存活数据相乘，恢复出丢失数据。

目前，多数分布式存储系统采用集中式编码实现[9]，数据修复过程一般包含4个步骤：

（1）参与节点从本地磁盘读取所需数据。

（2）为了实现减少数据传输量等目标，参与节点对数据进行本地合并后生成传输数据。

（3）参与节点将传输数据通过网络传输到修复节点。

（4）修复节点接收到修复需要的所有数据，通过计算重建出丢失数据。

2.3 修复性能的影响因素

当分布式存储系统中出现数据失效时，两种操作将触发数据修复：降级读（degrade read）和主动修复（proactive repair）。前者是用户访问失效数据时，触发存储系统对相应数据进行修复；后者是存储系统检测到节点失效或数据失效，主动进行的数据修复行为。对于前者，用户请求的响应需要等待数据修复完成，显著增加了请求延迟。而后者的修复操

作极大占用系统的计算资源、磁盘资源和网络资源，对用户的性能造成干扰。由此可见，加快数据修复过程，降低修复开销是纠删码修复面临的主要问题。

RS编码的修复中传输数据与磁盘读取数据等同，不需要步骤(2)的数据加工处理(见2.2节)。但是，也存在不少编码方案[11-12]，参与节点需要对磁盘读取数据进行线性组合，减少传输的数据量。因此，数据修复所花费的时间可以如下表示：

$$T_{\text{repair}}=\frac{D_{\text{disk}}}{S_{\text{disk}}}+T_{\text{com}}D_{\text{disk}}+\frac{kD_{\text{net}}}{S_{\text{net}}}+T_{\text{com}}(kD_{\text{net}})$$

其中，D_{disk}、D_{net}分别代表参与节点读取的数据量和网络传输的数据量；S_{disk}、S_{net}分别代表磁盘的读取速度和网络的传输速度；T_{com}代表计算所花费的时间；k代表修复过程中访问的节点数量。

在基于流水线的修复模式下，最大修复带宽受限于上述4个步骤中的最小处理带宽，修复时间受到计算开销、磁盘读写开销和网络传输开销的共同影响。由于大部分分布式系统都搭建在廉价的服务器上，CPU计算能力、磁盘性能和网卡速度均有可能成为制约修复带宽的瓶颈。如果能够降低计算处理、磁盘读取和网络传输的数据量，加快计算执行、磁盘读取和网络传输的速度，将减少纠删码的修复时间，并降低数据修复对应用性能的影响。

基于以上分析，计算操作、磁盘读写和网络传输是影响纠删码修复性能的关键因素。因此，从这三方面入手，对现有的优化技术进行归纳，并选取计算开销、读写开销、传输开销作为纠删码修复性能的评价标准。

3 计算优化

RS编码的运算法则建立在迦罗瓦域GF(2^w)上，修复过程不仅涉及残余矩阵的求逆运算，同时涉及复杂的有限域运算。GF(2^w)上的加法等同于异或运算，而乘法的计算相对复杂，涉及多项式乘法和取余运算。目前针对计算的优化主要分为两类：减少计算量和加速计算执行。

3.1 减少计算量

在柯西编码的生成矩阵中，“1”的数量反映计算过程中需要的异或次数，因此大量的研究工作试图通过寻找低密度生成矩阵的编码方案来减少计算量[13-15]。然而对于低密度的编码矩阵，其解码矩阵可能非常稠密，无法在修复时也同样减少计算量。

为了降低矩阵密度对计算量的直接影响，已有工作采用计算调度的方式来减少计算过程中的计算量[16]。其基本思想是调整计算的执行顺序，最大可能地利用计算过程中的中间结果来减少重复的计算。如图3所示，在$\boldsymbol{B}=\boldsymbol{AX}$的计算中，为了计算$\boldsymbol{B}$中4个目标块的值，传统的计算方式如图3中右边所示，整个计算需要12次异或操作。如果利用$b_2=b_1+x_1$，可以将整体的操作次数从12降低至9。然而，如何根据给定的生成矩阵寻找最优的计算调度方案，已经被Huang等人推测是NP完全问题[17]，现有的解决方式主要基于贪心原则和启发式搜索寻找次优的调度方案。

基于贪心的原则，Hafner等人[18]提出了CSHR(code specific hybrid reconstruction)调度算法，其思想是让向量$\boldsymbol{B}$中已经计算出的目标块b_i参与待求解

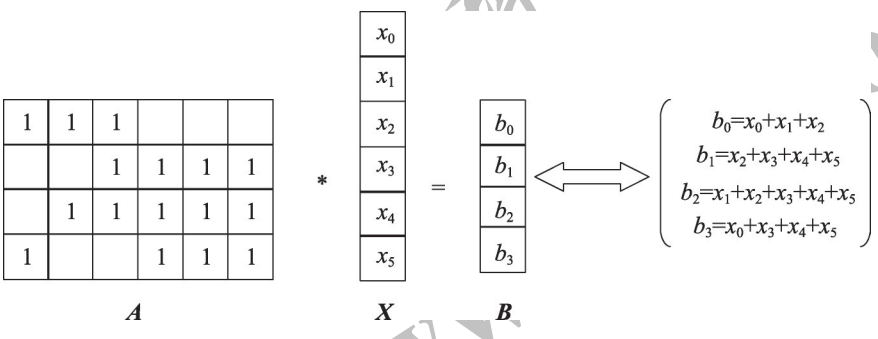
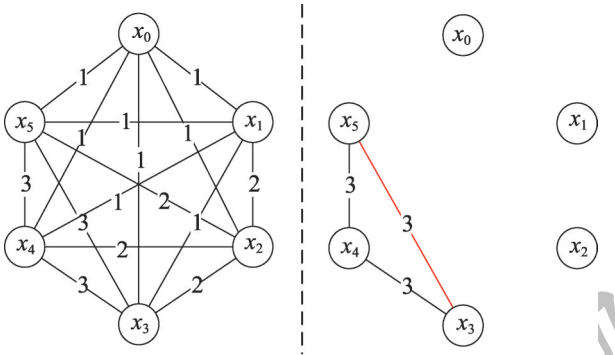


Fig.3 Data encoding process under 01 matrix

图3 01矩阵下的数据编码过程

目标块 b_j 的计算。比如，b_2 的求解可以借助于 b_1。相比传统计算方式下，目标块的求解固定使用一种编码等式，CSHR算法大大增加了编码等式可选的方案数量。得益于目标块之间存在公共的计算部分，CSHR算法在不使用额外空间的情况下，能够减少总的计算次数。

然而，CSHR算法只用目标块 b_i 的值优化其他目标块的求解，却忽略了多个目标块的编码等式中存在更细粒度的公共计算部分。比如图3中的例子，$x_3+x_4+x_5$ 均出现在 b_1、b_2 和 b_3 的编码等式中。如果预先计算并保存 $x_3+x_4+x_5$ 的值，那么只需额外3次计算即可求解出 b_1、b_2 和 b_3。文献[17]中，Huang等人提出的Subex（common subexpressions）算法恰好解决了这个问题。Subex算法首先统计任意计算单元组合 (x_i,x_j) 在所有编码等式中成对出现的次数，在图4(a)的完全图中展示了生成矩阵 $\boldsymbol{A}$ 中所有计算单元对的统计情况，边权代表两个端点构成的计算单元对出现的次数，此时边权最大值为3。然后移除边权低于3的边，在剩余图中利用最大权匹配算法选出最多的非交叉边，如图4(b)所示。Subex算法优先计算出这些边对应的计算组合，并用来替换目标块需要的运算，重复以上步骤，经过多次迭代，最终求解出所有目标块。



(a) Statistic for calculation unit pairs
(a) 计算单元对统计

(b) Selection for calculation unit pairs
(b) 计算单元对选取

Fig.4 Illustration of Subex

图4 Subex算法说明

除了单一从算法设计对调度进行优化，Zhang等人[19]提出的CaCo算法将低密度编码设计和计算调度结合在一起。该算法分为矩阵选取和调度选取两个阶段，首先使用多种矩阵生成算法得到多个低密度的柯西矩阵，然后对每个生成矩阵尝试不同的计算调度算法得到多种调度方案，最后在所有方案中选取计算量最小的调度方案。CaCo算法将现有的矩阵生成算法和计算调度算法融入到统一的计算框架中，相比单一算法，能够产生更多的选择，进而得到计算量更低的调度方案。

3.2 加速计算执行

有限域的乘法运算存在计算复杂的问题，为了加快执行速度，目前有两种优化方式：位运算转换法和查表法。位运算转换法[20]基于有限域上的元素都可以采用二进制矩阵进行表示，它将生成矩阵转换为GF(2)域空间上的01矩阵，因此整个计算过程只存在异或运算，降低了计算的复杂度。

查表法是工程实现中一种常见方式，它通过预先生成有限域乘法表，在计算过程中避免复杂的多项式计算，只需查询相应的计算结果。在 $GF(2^w)$ 中，二维乘法表需要耗费 $O((2^w)^2)$ 的存储空间，随着 w 的增大，该方式不再适用。由于有限域的非零元素均能用本原元的指数形式表示，进而有限域的乘法能够转换为指数上的加法运算。因此，在选定本原元的情况下，借助对数表（数字到指数的映射）和反对数表（指数到数字的映射），两数的乘法运算只需两次对数表查询得到相应的指数数值，然后通过求和运算获得乘法结果的对应指数数值，最后通过一次反对数表查询，即可得到乘法运算的结果。该方式只需 $O(2^w)$ 的存储空间。查表法用访存操作替代计算，将执行速度的瓶颈从处理器转移到内存。最近，由于英特尔SIMD（single instruction multiple data）技术[21]的发展，处理器中已经普遍支持SIMD指令集，Plank等人[22]利用SIMD技术，将查询表放入寄存器中，一次同时查询多个乘法计算结果，使得乘法速度能够达到8 GB/s。

随着多核CPU和GPU的发展，并行执行也被用于纠删码的计算加速[23]。针对多核CPU，文献[24-26]分别完成了柯西RS编码、EVENODD编码[27]、RDP（row-diagonal parity）编码[28]的并行设计，相比传统

RDP编码，修复能够提高40%的解码速度。并行加速的难点在于如何进行任务切分，使负载能够均衡分布在每一个核上。目前，主要存在块级别和等式级别的两种切分方式：前者按块对数据切分，采用数据并行的方式达到并行执行的效果；而后者采用分配编码等式的方式，使得每个核负责一定数量的编码等式。相对于块级别的并行，等式级别的并行可以使数据的访问聚集在每一个核中的特定区域，能够达到更均衡的分配。除此之外，在非对称编码的并行修复中，文献[29]提出了一种新划分方式，其发现修复过程中存在独立的出错块，并据此对校验矩阵切分，从而实现出错的数据块的并行修复。

由于RS编码中涉及大量的矩阵向量乘运算，相比CPU，GPU的数据矢量化和并行处理能力更加适合这种计算模型。Gibraltar[30]是基于GPU实现的RS编码库，相比Jerasure[31]，Gibraltar能够达到更高的吞吐量，并且解码性能几乎与编码性能一样。

4 读写优化

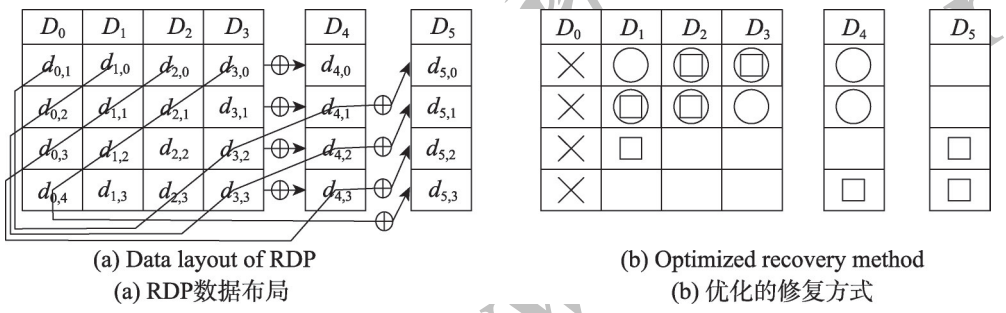
磁盘读取的时间开销受读取的数据总量和并行度控制，通过降低I/O总量和提高I/O并行度两种方式，均可以有效降低单盘负载，从而减少磁盘读取的时间开销。

4.1 降低I/O总量

阵列码EVENODD和RDP是常见的RAID6（redundant arrays of independent disks）编码，在其编码布局中均存在水平校验组和对角线校验组。当单个数据盘出错时，传统修复方式采用水平校验组对数据进行修复。如图5（a）所示的RDP(6, 2)数据布局，一共6块磁盘，其中数据盘4块。当磁盘D_0损坏时，采用水平校验组，需要从D_1、D_2、D_3、D_4读取16块数据，该修复方式不仅未使用磁盘D_5，同时忽略了两种校验组间存在的数据重叠。

文献[32]中，Xiang等人对RDP恢复进行了优化，首次提出同时使用两种校验组的混合修复方式。如图5（b）所示，对于丢失的4个数据块，其中一半数据块的修复采用水平校验组，另一半数据块修复采用对角线校验组。该修复方式下，读取的数据块存在4块重叠，需要读取的数据块数量降低为12，从而减少了25%的I/O读取总量。混合修复方式通过寻找读取过程中最大的重叠区域来降低数据读取总量，基于同样的思想，文献[33-35]分别针对EVENODD编码、X-Code编码[36]和HDP（horizontal-diagonal parity）编码[35]提出了单节点故障下的快速恢复算法。然而，上述的双容错编码由于其特定的数据布局，只需考虑两种校验组，当将该方式扩展到容错能力为3的STAR编码[37]或者更通用的柯西RS编码时，复杂的数据布局导致使用指数级的搜索时间来寻找最大的重叠区域[38]。为了降低时间开销，Zhu等人提出一种更替修复算法[39]，其通过启发式爬山算法来寻找次优的修复方案。

为了将数据重叠的思想引入到RS编码，Khan等人通过对标准RS编码进行改造，设计出一种新型编码——循环RS编码[38]。如图6所示，标准RS编码中多个校验块的生成来自对数据块不同系数的选取，而循环RS编码采取不同的生成方式，借助多个组合



(a) Data layout of RDP
(a) RDP数据布局

(b) Optimized recovery method
(b) 优化的修复方式

Fig.5 Data layout and recovery method of RDP(6, 2)

图5 RDP(6, 2)编码的数据布局及修复方式

在一起的条带，从相邻条带内选取不同的数据块形成新的校验块。当磁盘 D_0 出错时，标准RS编码恢复3个条带需要读取18个数据块，而循环RS编码由于存在3块数据重叠，只需15个数据块。

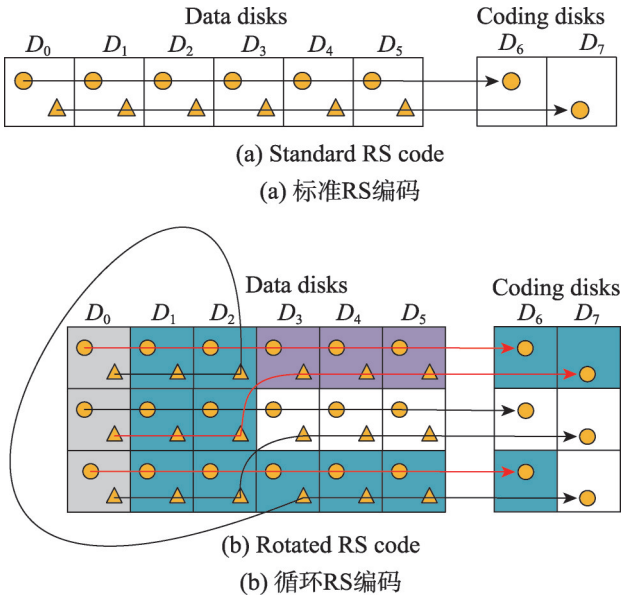


Fig.6 Comparison between rotated RS code and RS code

图6 循环RS编码与RS编码的对比

4.2 提高I/O并行度

传统磁盘阵列在运行过程中，为了及时替换出错的磁盘，往往配备了热备盘。然而在没有发生磁盘失效的正常状态下，热备盘只能空闲等待。针对这个问题，文献[40]中，Menon等人提出了分布式空闲块的思想，不再设置专门的热备盘，而使用磁盘中的空闲区域作为修复使用的存储空间。在正常状态下，该方式让热备盘也加入使用，所有的磁盘都能参与服务，提高了I/O处理能力；失效修复时，更多磁盘的参与降低了修复时间。

除了热备盘的利用，分簇技术（parity declustering）[41]在恢复数据读取总量不变的情况下，通过使用更多的磁盘，显著降低了修复的时间开销。以S²-RAID[42]编码设计为例，S²-RAID在条带大小不变的情况下，允许加入更多的磁盘参与修复。如图7所示，当磁盘 D_4 出错时，由于条带的大小为3，{1.1, 3.1, 8.1}3个数据块的修复需要读取6个数据块，传统磁盘阵列的修复需要从两块磁盘上各读取3块数据，而S²-RAID使用6块磁盘参与数据读取，每块磁盘只需读取1个数据块，大大降低了单盘上的负载。

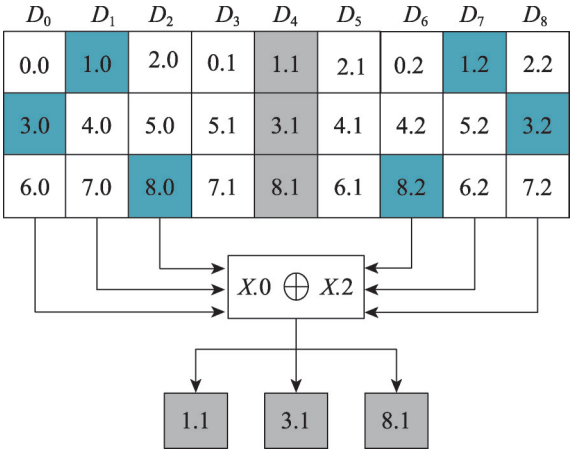


Fig.7 Data layout of S²-RAID

图7 S²-RAID的数据布局

5 传输优化

传统RS (k,m) 编码对 k 个数据块生成 m 个校验块，在修复某块数据时需要将 k 个参与节点上的数据传输到修复节点进行数据重建。以RS(8, 2)为例，数据块大小为128 MB，在数据修复过程中，整个网络需要传输1 GB的数据量。这给分布式存储系统带来了较大的网络负担。目前，传输优化主要有减少单个参与节点传输量和减少参与节点数量两种方式。

5.1 减少单个参与节点传输量

RS编码将节点上存储的数据作为单一整体进行计算，因而修复过程需要参与节点读取并传输所有数据。如果将节点上的数据划分成子块，以子块作为最小编码单位，则参与节点可以只传输部分子块数据用于修复。

为了降低单个参与节点的数据传输量，Dimakis等人提出了一类新型编码——再生码（regenerating code，RGC）[11-12,43-44]。参数为 (n,k,d) 的再生码，满足任意 k 个节点能够重建整个数据，单节点出错时，从任意 d（$k\leqslant d\leqslant n-1$）个节点上传输数据进行修复。与RS编码相比，虽然RGC编码需要更多的节点参与修复，但是每个节点的数据传输量低，从而能够降低整体的网络传输量。目前，RGC编码主要关注两类：

MSR(minimum storage regenerating)编码和MBR(minimum bandwidth regenerating)编码。MSR编码与RS编码同等存储开销下,具有更低的网络开销,而MBR编码允许存储更多的数据来获得更低的网络开销。

干扰对齐(interference alignment)技术是构造MSR编码的一种主要方式,其思想是同时消去多个不需要的变量。如图8所示的EMSR(4, 2, 3)(exact-repair MSR)编码[43],当节点0出错时,剩余的3个存活节点读取本地存放的两个编码子块,通过求和的方式将两个子块合并成单个子块,并将其传输到修复节点。此时,修复节点得到$\{B_1+B_2, A_1+2A_2+B_1+B_2, 2A_1+A_2+B_1+B_2\}$ 3个编码子块。利用编码子块B_1+B_2可以同时消去剩余两个编码子块中的B_1和B_2,从而得到$\{A_1+2A_2, 2A_1+A_2\}$,通过对这两个编码子块运算即可求解出丢失的数据子块A_1和A_2。在该过程中,数据修复只需传输3个编码子块,而如果只利用节点1和节点2的数据进行修复,需要传输4个编码子块。相比之下,EMSR编码降低了25%的网络传输量。Wu等人[11]最早将该技术用于MDS编码的精确修复,不过只解决了$k=2$的MSR精确编码。随后,Suh等人[44]进行改进,解决了$d\geqslant 2k-1$的MSR精确编码。

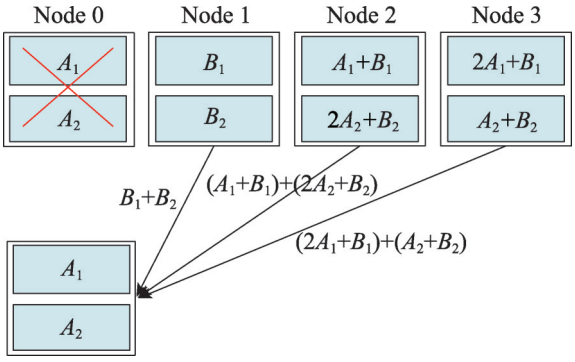


Fig.8 Recovery process of EMSR(4, 2, 3) code

图8 EMSR(4, 2, 3)编码修复过程

矩阵乘(product matrix, PM)[12]是构造再生码的另一种主要方式,它可以同时构造出MSR编码和MBR编码。再生码的矩阵构造由编码矩阵和数据矩阵构成,式(1)展示了PM-MBR(6, 3, 4)的矩阵构造,与RS编码相比,PM编码采用数据矩阵替换了数据向量,并且在数据矩阵中存在重复的数据子块。

$$\boldsymbol{M}=\begin{bmatrix}1&1&1&1\\1&2&4&1\\1&3&2&6\\1&4&2&1\\1&5&4&6\\1&6&1&6\end{bmatrix}*\begin{bmatrix}u_1&u_2&u_3&u_7\\u_2&u_4&u_5&u_8\\u_3&u_5&u_6&u_9\\u_7&u_8&u_9&0\end{bmatrix} \tag{1}$$

图9展示了PM-MBR(6, 3, 4)编码的具体修复过程。当节点出错时,首先选择出4个节点参与修复,读取每个参与节点上的4个编码子块。然后,利用出错节点对应的编码向量将每个参与节点的编码子块线性组合成单个传输子块。最后,修复节点利用所有参与节点的编码向量构成解码矩阵,通过其逆矩阵与传输子块的乘积计算得到丢失的数据。在此过程中,节点修复只需传输4个编码子块,而传统RS编码需要传输9个编码子块,网络传输量降低了55.6%。相比MSR编码,PM-MBR能够实现更低的网络传输量。

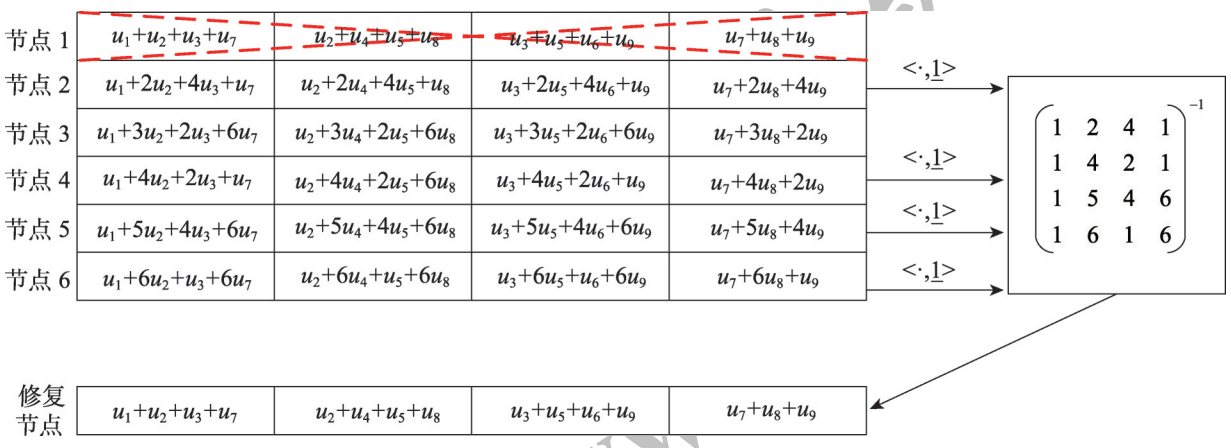


Fig.9 Recovery process of PM-MBR (6, 3, 4) code

图9 PM-MBR(6, 3, 4)编码修复过程

在数据传输之前，再生码需要参与节点读取所有的编码子块，然后计算得到传输子块。修复过程中，再生码降低网络开销的同时，却显著增加了磁盘读写开销和计算开销。

为了解决上述问题，Rashmi等人[45]设计的PM-RBT（product matrix reconstruct-by-transfer）编码，通过预先存储计算结果的方式，有效减少了在线修复时所需的计算和读取。Chen等人[46]设计了F-MSR（functional MSR）编码，修复出的数据为原始数据子块的线性组合，能够继续保持数据容错能力不变。对于单节点的修复，F-MSR编码只需从剩余所有存活节点上直接读取一个编码子块，无需参与节点进行计算。Shah等人[47]设计了MBR-RBT（MBR reconstruct-by-transfer）编码，传输的数据即为磁盘读取的数据，不仅降低了磁盘读取开销，而且也避免了计算开销。

上述编码解决了单节点故障下的传输优化，然而在真实业务场景中，节点故障往往相互关联，从而存在多节点同时故障的风险[48-49]。在并发故障模型下，存储系统可以延迟修复操作，累积故障节点数量超过一定阈值时，才开始执行修复[50-51]。这种方式能够在短暂故障下出现数据不可用时，避免无效的数据修复，同时增加了在传输节点上合并相关数据，减少传输量的可能性。除了从编码设计上来降低单个节点的传输开销，Subedi等人[52]从修复的机制出发，提出了一种结合缓存，共同合作，并主动重建的数据修复机制——CoARC（cooperative，aggressive recovery and caching）。在现有的分布式系统中，比如Hadoop，其客户端触发的降级读操作独立于自主的修复过程。客户端在使用完数据后，将丢弃降级读操作产生的重建数据。此时，单个客户端对相同出错条带的访问，或者多个客户端触发相同数据块的修复，都将给系统的网络和计算带来极大的负荷。CoARC通过对修复的数据进行缓存，过滤掉相同的重建操作，并且它在每次降级读触发的修复中，主动重建出错条带上所有不可用的数据。在这种提前修复和缓存的方式下，能够显著减少数据重建的次数，避免单个节点上相同数据的多次传输，从而有效降低整个网络的传输量。

5.2 减少参与节点数量

传统RS编码将多个数据块通过线性组合的方式生成全局校验块，数据修复时需要读取的数据量与参与构建的数据块数量相等。在全局校验块数量固定的情况下，增加全局校验组中数据块的数量，能够降低编码的存储开销比率，却会导致参与修复的节点数量增多，带来网络传输开销和计算开销的增加。

局部修复码（local reconstruction code，LRC）[53]在传统RS编码的基础上，引入局部分组的思想，将编码块划分成多个小分组，分组内部生成局部校验块进行容错保护。数据修复不再完全依赖于全局校验组，优先使用局部分组内的数据进行重建，从而通过减少参与节点的数量，降低了修复过程中的网络传输开销和磁盘读取开销。

目前存在多种LRC编码方案，如图10所示的Pyramid编码[53]，数据块被划分成多个分组，每个分组内引入一个局部校验块。由于分组之间形成嵌套关系，当同一个小分组内出现多个数据块丢失时，Pyramid编码可以借助其上层分组进行修复，从而在多节点失效时依然可以降低参与节点的数量。由于多节点同时失效的概率较低，在Azure系统中目前只使用了一层局部校验组[54]。

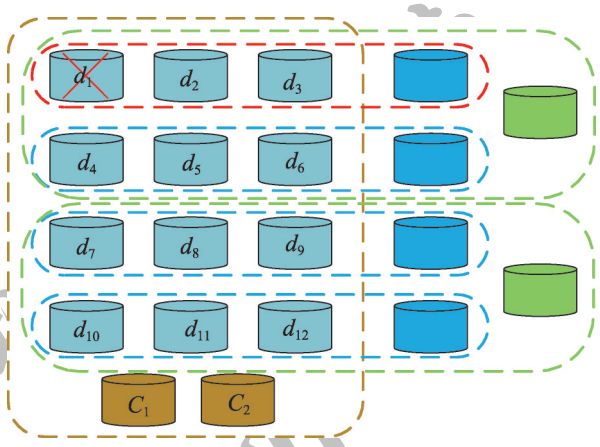


Fig.10 Pyramid code

图10 Pyramid编码

Pyramid编码只对数据块引入了局部分组，全局校验块的修复依然需要读取所有的数据。为了解决这个问题，Sathiamoorthy等人[55]在HDFS-Xorbas系统中对全局校验块也引入了局部分组，并选取特定的

系数对存储开销进行优化。如图11所示，HDFS-Xorbas系统为RS(10, 4)编码生成了3个局部校验块S_1、S_2和S_3，通过特定系数的选取使$S_1+S_2+S_3=0$，从而S_3的值可以根据$-S_1-S_2$计算得到，S_3作为隐式校验块无需存储。在该方式下，任意单个节点的修复，只需访问5个节点的数据，同时全局校验块的保护没有引入额外的存储开销。实验表明，HDFS-Xorbas以额外的14%(2/(10+4))的存储开销将网络流量减少了50%。

LRC编码通过局部分组的方式减少了修复访问的节点数量，显著降低了网络带宽和磁盘带宽的占用。由于其实现简单，只需在标准RS编码的基础上引入多个局部校验块，因而已被用于Microsoft[54]、Facebook[55]的存储系统中。为了结合LRC编码和RGC编码各自的优势，Xia等人[56]将PM编码和LRC编码进行混合使用，根据数据访问热度的不同，对于热数据采用数据修复快的编码，从而减少修复过程对前台应用带来的延迟增大，而对于冷数据采用低存储开销的编码，能够有效减少存储系统的空间开销。在这种混编方式，整个存储系统能够同时获得快速修复能力和高存储空间利用率。

6 讨论

服务领域的存储系统需要提供连续不停机的存储服务，因此数据修复操作必须在线进行。一方面，基于纠删码的存储系统的修复完成时间受到计算开销、磁盘读写开销和网络传输开销等因素的共同影响。另一方面，因为数据修复操作和前台I/O操作共享甚至竞争存储系统的I/O资源，数据修复操作对计算资源、磁盘带宽和网络带宽的占用又会给前台应用性能带来明显干扰。因此，计算开销、读写开销和传输开销是影响纠删码修复性能的三大关键因素。

为了提高纠删码存储系统中数据修复效率，研究者主要从计算优化、读写优化和传输优化等三方面开展工作。

(1)计算优化方面：计算硬件的发展允许通过SIMD技术实现数据级并行，允许对多个数据单元同时执行相同的操作，显著加快了基于有限域运算的编码计算。而通过调整计算的执行顺序，避免了计算过程中不必要的重复计算，能够有效减少计算量，进一步降低计算开销。

(2)读写优化方面：通过合理选择修复使用的条带，让数据恢复所需的数据出现重叠，使得读取同一块数据能够用于多块数据的修复，减少了数据读取总量。另一方面，在不减少数据读取总量的情况下，通过将更多磁盘引入到数据恢复中来，降低单盘上的读取负载，从而减少整体的读取时间。

(3)传输优化方面：为了降低网络传输量，研究者分别设计出RGC编码和LRC编码。RGC编码通过降低单个参与节点的数据传输量，减少修复过程中的网络开销。LRC编码通过引入局部分组，解除了修复参与节点数量与全局校验组之间的关联，以存储开销的增加换取修复参与节点数量的减少，从而降低修复过程中的磁盘读写开销和网络传输开销。

为了全面对比现有的编码方案，表1在以计算开销、读写开销、传输开销作为修复性能评价标准的基础上，结合容错能力、存储利用率等指标对现有的编码方案进行了对比。可以看出，没有一种方案可以

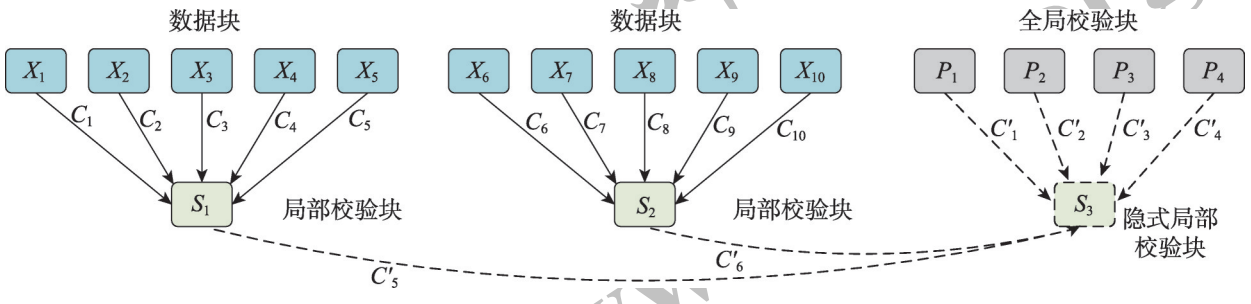


Fig.11 Encoding structure of HDFS-Xorbas

图11 HDFS-Xorbas编码结构

Table 1 Comparison of various code scheme

表1 各编码方案的对比评价

编码方案	计算	读写	传输	参数受限	MDS编码	存储利用率/%	容错能力	编码种类
RS(6,3)	—	—	—		√	66.70	3	传统编码
循环RS(6,3)	—	↓	↓	√	√	66.70	3	RS变种
EMSR(4,2,3)	↑	↑	↓	√	√	50.00	2	MSR编码
PM-RBT(12,6,11)	↓	↓	↓	√	√	50.00	6	MSR编码
PM-MBR(6,3,4)	↑	↑	↓		√	37.50	3	MBR编码
MBR-RBT(5,3,4)	↓	↓	↓	√	√	45.00	3	MBR编码
Azure LRC(6,2,2)	↓	↓	↓			60.00	3	LRC编码
Xorbas LRC(12,2,2)	↓	↓	↓			75.00	3	LRC编码
RDP(6,2)	—	↓	↓	√	√	66.70	2	阵列码
EVENODD(7,2)	—	↓	↓	√	√	71.40	2	阵列码

很好地满足所有指标，LRC编码的修复虽然在计算开销、读写开销和传输开销上都取得良好表现，但它不是MDS编码，节点的修复只能访问固定的节点集合，而且局部校验块的计算也在其编码阶段引入了额外的计算量。MSR编码和MBR编码虽然能够降低网络开销，但是却引入了更多的计算开销和读写开销。虽然PM-RBT(12, 6, 11)和MBR-RBT(5, 3, 4)引入“修复即传输”的思想来降低计算开销和读写开销，但是PM-RBT编码是基于MSR编码的改进，参数的选取受到一定程度的限制。而MBR-RBT编码的修复需要从剩余所有节点上传输数据，在面临多个节点失效时，无法降低网络传输开销。阵列码RDP(6, 2)、EVENODD(7, 2)和循环RS(6, 3)两类编码在磁盘开销和网络开销方面都有降低，但均存在参数受限，容错能力不高的缺陷。

7 总结与展望

本文分析了影响纠删码修复的关键因素，从计算、读写、传输三方面对优化纠删码修复性能的关键技术进行了探讨。现有的编码方案中大多未能兼顾计算、读写、传输三方面，如何从理论设计和系统实现上优化实现这三目标，还存在巨大的技术挑战。

理论设计方面，同等容错能力下，MSR编码能够在不增加额外存储开销的情况下，显著降低网络传输量。但是MSR编码在高容错能力时还存在参数选取受限，计算复杂和存储利用率过低等缺陷。如何设计高编码率、高可靠和低复杂度的MSR编码将会是未来的一个重要研究方向。

系统实现方面，结合设备特性，对编码方案进行适配和调优。编码的理论效果与工程实现存在巨大的鸿沟，以磁盘读取为例，再生码修复过程需要完整读取磁盘上的数据，是否存在只读取部分子块的可能性？如果存在，当将原有的大块连续读取转变为多次跳读之后，虽然读取总量有所降低，但其带宽却可能下降。针对不同的编码方案，如何在工程实现中对计算、读取、传输三方面进行适配和调优，也将是未来研究的重点。

References:

[1] Big data and what it means[EB/OL]. [2016-11-27]. http://www.uschamberfoundation.org/bhq/big-data-andwhat-it-means.

[2] Shvachko K, Kuang Hairong, Radia S, et al. The hadoop distributed file system[C]//Proceedings of the 26th Symposium on Mass Storage Systems and Technologies, Lake Tahoe, USA, May 3-7, 2010. Washington: IEEE Computer Society, 2010: 1-10.

[3] Weil S A, Brandt S A, Miller E L, et al. Ceph: a scalable, high-performance distributed file system[C]//Proceedings of the 7th Symposium on Operating Systems Design and Implementation, Seattle, USA, Nov 6-8, 2006. Berkeley, USA: USENIX Association, 2006: 307-320.

[4] Reed I S, Solomon G. Polynomial codes over certain finite fields[J]. Journal of the Society for Industrial & Applied

Mathematics, 1960, 8(2): 300-304.

[5] 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]//Proceedings of the 5th Workshop on Hot Topics in Storage and File Systems, San Jose, USA, Jun 27-28, 2013. Berkeley, USA: USENIX Association, 2013: 1-5.

[6] Dimakis A G, Ramchandran K, Wu Yunnan, et al. A survey on network codes for distributed storage[J]. Proceedings of the IEEE, 2011, 99(3): 476-489.

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

[8] 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.

[9] 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.

[10] MacWilliams F J, Sloane N J A. The theory of error-correcting codes[M]. Amsterdam: Elsevier North-Holland, Inc, 1977.

[11] Wu Yunnan, Dimakis A G. Reducing repair traffic for erasure coding-based storage via interference alignment[C]// Proceedings of the 2009 International Symposium on Information Theory, Seoul, Jun 28-Jul 3, 2009. Piscataway, USA: IEEE, 2009: 2276-2280.

[12] Rashmi K V, Shah N B, Kumar P V. Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction[J]. IEEE Transactions on Information Theory, 2011, 57(8): 5227-5239.

[13] Blaum M, Roth R M. On lowest density MDS codes[J]. IEEE Transactions on Information Theory, 1999, 45(1): 46-59.

[14] Plank J S, Luo Jianqiang, Schuman C D, et al. A performance evaluation and examination of open-source erasure coding libraries for storage[C]//Proceedings of the 7th Conference on File and Storage Technologies, San Francisco, USA, Feb 24-27, 2009. Berkeley, USA: USENIX Association, 2009: 253-265.

[15] Plank J S, Xu Lihao. Optimizing Cauchy Reed-Solomon codes for fault-tolerant network storage applications[C]// Proceedings of the 5th International Symposium on Network Computing and Applications, Cambridge, USA, Jul 24-26, 2006. Washington: IEEE Computer Society, 2006: 173-180.

[16] Plank J S. XOR's, lower bounds and MDS codes for storage [C]//Proceedings of the Information Theory Workshop, Paraty, Brazil, Oct 16-20, 2011. Piscataway, USA: IEEE, 2011: 503-507.

[17] Huang Cheng, Li Jin, Chen Minghua. On optimizing XOR-based codes for fault-tolerant storage applications[C]//Proceedings of the Information Theory Workshop, Tahoe City, USA, Sep 2-6, 2007. Piscataway, USA: IEEE, 2007: 218-223.

[18] Hafner J L, Deenadhayalan V, Rao K K, et al. Matrix methods for lost data reconstruction in erasure codes[C]//Proceedings of the 4th Conference on File and Storage Technologies, San Francisco, USA, Dec 13-16, 2005. Berkeley, USA: USENIX Association, 2005: 15-30.

[19] Zhang Guangyan, Wu Guiyong, Wang Shupeng, et al. CaCo: an efficient cauchy coding approach for cloud storage systems[J]. IEEE Transactions on Computers, 2016, 65(2): 435-447.

[20] Blomer J, Kalfane M, Karp R, et al. An XOR-based erasure-resilient coding scheme[C]//Proc ACM SIGCOMM, 1995.

[21] Intel Corporation. Intel® 64 and IA-32 architectures software developer manuals.combined volumes: 1, 2A, 2B, 2C, 3A, 3B and 3C. Order Number: 325462-044US, 2012.

[22] Plank J S, Greenan K M, Miller E L. Screaming fast Galois field arithmetic using intel SIMD instructions[C]//Proceedings of the 11th Conference on File and Storage Technologies, San Jose, USA, Feb 12-15, 2013. Berkeley, USA: USENIX Association, 2013: 299-306.

[23] Chen H B, Fu Song. Parallel erasure coding: exploring task parallelism in erasure coding for enhanced bandwidth and energy efficiency[C]//Proceedings of the 2016 International Conference on Networking, Architecture and Storage, Long Beach, USA, Aug 8-10, 2016. Washington: IEEE Computer Society, 2016: 1-4.

[24] Sobe P. Parallel Reed/Solomon coding on multicore processors[C]//Proceedings of the 2010 International Workshop on Storage Network Architecture and Parallel I/Os, Incline Village, USA, May 3, 2010. Washington: IEEE Computer Society, 2010: 71-80.

[25] Feng Jun, Chen Yu, Summerville D. EEO: an efficient MDS-like RAID-6 code for parallel implementation[C]//Proceedings of the 33rd Conference on Sarnoff, Princeton, USA, Apr 12-14, 2010. Piscataway, USA: IEEE, 2010: 16-20.

[26] Feng Jun, Chen Yu, Summerville D, et al. An extension of RDP code with parallel decoding procedure[C]//Proceedings of the Consumer Communications and Networking Conference, Las Vegas, USA, Jan 14-17, 2012. Piscataway, USA: IEEE, 2012: 154-158.

[27] Blaum M, Brady J, Bruck J, et al. EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures[J]. IEEE Transactions on Computers, 1995, 44(2): 192-202.

[28] Corbett P, English B, Goel A, et al. Row-diagonal parity for double disk failure correction[C]//Proceedings of the 3rd Conference on File and Storage Technologies, San Francisco, USA, Mar 31-Apr 2, 2004. Berkeley, USA: USENIX Association, 2004: 1-14.

[29] Li Shiyi, Cao Qiang, Wan Shenggang, et al. PPM: a partitioned and parallel matrix algorithm to accelerate encoding/decoding process of asymmetric parity erasure codes[C]//Proceedings of the 44th International Conference on Parallel Processing, Beijing, Sep 1-4, 2015. Washington: IEEE Computer Society, 2015: 460-469.

[30] Curry M L, Skjellum A, WardH L, et al. Gibraltar: a Reed-Solomon coding library for storage applications on programmable graphics processors[J]. Concurrency and Computation: Practice and Experience, 2011, 23(18): 2477-2495.

[31] Plank J S, Simmerman S, Schuman C D. Jerasure: a library in C/C++ facilitating erasure coding for storage applications-Version 1.2, CS-08-627[R]. Knoxville: University of Tennessee, 2008.

[32] Xiang Liping, Xu Yinlong, Lui J C S, et al. Optimal recovery of single disk failure in RDP code storage systems[J]. ACM SIGMETRICS Performance Evaluation Review, 2010, 38(1): 119-130.

[33] Wang Zhiying, Dimakis A G, Bruck J. Rebuilding for array codes in distributed storage systems[C]//Proceedings of the Global Communication Conference Exhibition and Industry Forum, Miami, USA, Dec 6-10, 2010. Piscataway, USA: IEEE, 2011: 1905-1909.

[34] Xu Silei, Li Runhui, Lee P P C, et al. Single disk failure recovery for X-code-based parallel storage systems[J]. IEEE Transactions on Computers, 2014, 63(4): 995-1007.

[35] Wu Chentao, He Xubin, Wu Guanying, et al. HDP code: a horizontal-diagonal parity code to optimize I/O load balancing in RAID-6[C]//Proceedings of the 41st International Conference on Dependable Systems Networks, Hong Kong, China, Jun 27-30, 2011. Washington: IEEE Computer Society, 2011: 209-220.

[36] Xu Lihao, Bruck J. X-code: MDS array codes with optimal encoding[J]. IEEE Transactions on Information Theory, 1999, 45(1): 272-276.

[37] Huang Cheng, Xu Lihao. STAR: an efficient coding scheme for correcting triple storage node failures[J]. IEEE Transactions on Computers, 2007, 57(7): 889-901.

[38] Khan O, Burns R C, Plank J S, et al. Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads[C]//Proceedings of the 10th Conference on File and Storage Technologies, San Jose, USA, Feb 14-17, 2012. Berkeley, USA: USENIX Association, 2012: 251-264.

[39] Zhu Yunfeng, Lee P P C, Xu Yinlong, et al. On the speedup of recovery in large-scale erasure-coded storage systems[J]. IEEE Transactions on Parallel & Distributed Systems, 2014, 25(7): 1830-1840.

[40] Menon J, Mattson D. Distributed sparing in disk arrays[C]//Proceedings of the 37th International Conference on Compion, San Francisco, USA, Feb 24-28, 1992. Piscataway, USA: IEEE, 1992: 410-421.

[41] Holland M, Gibson G A. Parity declustering for continuous operation in redundant disk arrays[C]//Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, USA, Oct 12-15, 1992. New York: ACM, 1992: 23-35.

[42] Wan Jiguang, Wang Jibin, Yang Qing, et al. S^2-RAID: a new RAID architecture for fast data recovery[C]//Proceedings of the 26th Symposium on Mass Storage Systems and Technologies, Incline Village, USA, May 3-7, 2010. Washington: IEEE Computer Society, 2010: 1-9.

[43] 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, 2012, 58(4): 2134-2158.

[44] Suh C, Ramchandran K. Exact-repair MDS codes for distributed storage using interference alignment[C]//Proceedings of the 2010 IEEE International Symposium on Information Theory, Austin, USA, Jun 13-18, 2010. Piscataway, USA: IEEE, 2010: 161-165.

[45] Rashmi K V, Nakkiran P, Wang Jingyan, et al. Having your

cake and eating it too: jointly optimal erasure codes for I/O, storage, and network-bandwidth[C]//Proceedings of the 13th Conference on File and Storage Technologies, Santa Clara, USA, Feb 16-19, 2015. Berkeley, USA: USENIX Association, 2015: 81-94.

[46] Chen H C H, Hu Yuchong, Lee P P C, et al. NCCloud: a network-coding-based storage system in a cloud-of-clouds[J]. IEEE Transactions on Computers, 2014, 63(1): 31-44.

[47] Shah N B, Rashmi K V, Kumar P V, et al. Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff[J]. IEEE Transactions on Information Theory, 2012, 58(3): 1837-1852.

[48] Ford D, Labelle F, Popovici F I, et al. Availability in globally distributed storage systems[C]//Proceedings of the 9th Conference on Operating Systems Design and Implementation, Vancouver, Canada, Oct 4-6, 2010. Berkeley, USA: USENIX Association, 2010: 61-74.

[49] Schroeder B, Gibson G A. Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you?[C]// Proceedings of the 5th Conference on File and Storage Technologies, San Jose, USA, Feb 13-16, 2007. Berkeley, USA: USENIX Association, 2007: 1-16.

[50] Pei Xiaoqiang, Wang Yijie, Ma Xingkong, et al. Repairing multiple failures adaptively with erasure codes in distributed storage systems[J]. Concurrency and Computation: Practice and Experience, 2016, 28(5): 1437-1461.

[51] Li Runhui, Lin Jian, Lee P P C. Enabling concurrent failure recovery for regenerating-coding-based storage systems: from theory to practice[J]. IEEE Transactions on Computers, 2015, 64(7): 1898-1911.

[52] Subedi P, Huang Ping, Liu Tong, et al. CoARC: co-operative, aggressive recovery and caching for failures in erasure coded hadoop[C]//Proceedings of the 45th International Conference on Parallel Processing, Philadelphia, USA, Aug 16-19, 2016. Washington: IEEE Computer Society, 2016: 288-293.

[53] Huang Cheng, Chen Minghua, Li Jin. Pyramid codes: flexible schemes to trade space for access efficiency in reliable data storage systems[J]. ACM Transactions on Storage, 2013, 9(1): 3.

[54] Huang Cheng, Simitci H, Xu Yikang, et al. Erasure coding in windows Azure storage[C]//Proceedings of the Annual Technical Conference, Boston, USA, Jun 13-15, 2012. Berkeley, USA: USENIX Association, 2012: 15-26.

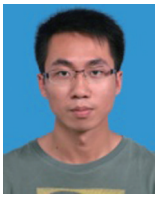
[55] Sathiamoorthy M, Asteris M, Papailiopoulos D, et al. Xoring elephants: novel erasure codes for big data[J]. Proceedings of the VLDB Endowment, 2013, 6(5): 325-336.

[56] Xia Mingyuan, Saxena M, Blaum M, et al. A tale of two erasure codes in HDFS[C]//Proceedings of the 13th Conference on File and Storage Technologies, Santa Clara, USA, Feb 16-19, 2015. Berkeley, USA: USENIX Association, 2015: 213-226.

附中文参考文献：

[8] 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.

[9] 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.



YANG Songlin was born in 1992. He is an M.S. candidate at Tsinghua University. His research interests include network storage and big data processing, etc.
杨松霖(1992—),男,清华大学硕士研究生,主要研究领域为网络存储,大数据处理等。

ZHANG Guangyan was born in 1976. He is an associate professor and M.S. supervisor at Tsinghua University. His research interests include network storage, distributed computing and big data processing, etc.
张广艳(1976—),男,清华大学副教授,主要研究领域为网络存储,分布式计算,大数据处理等。