减少重建数据量的冗余编码技术研究

马良荔[1] 柳 青[1,2]

(海军工程大学电子工程学院 武汉 430000)[1] (华中科技大学计算机科学与技术学院 武汉 430000)[2]

摘 要 为防止硬件故障或机器宕机导致的数据丢失,冗余编码技术被广泛应用于分布式存储系统中来保证数据的可靠性。然而,传统的冗余编码技术,如里德-所罗门码,存在着重建数据量大的问题。副本技术在重建丢失数据时只需要读取和传输丢失的数据,而冗余编码需要读取和传输更大的数据量,从而消耗更多的磁盘 I/O 带宽和网络带宽。因此,基于冗余编码的分布式存储系统在重建数据时将消耗更长的时间,从而将整个系统长时间暴露在一种降级的模式下,进而增加了发生永久性数据丢失的风险。为解决这个问题,减少重建数据量的冗余编码技术不断被提出,然而只有这些冗余编码与传统的里德-所罗门码的比较,缺少它们在存储系统的综合比较。系统地从减少重建数据量等几个重要方面研究了这些减少重建数据量的冗余编码技术,从而为实际系统中采用合适的编码提供重要参考和依据。

关键词 冗余编码,数据重建,存储系统,分布式系统

中图法分类号 TP302.8 **文献标识码** A

Researches of Redundancy Coding Technologies on Reducing Reconstruction Data Amount

MA Liang-li[1] LIU Qing[1,2]

(School of Electronic Engineering, Naval University of Engineering, Wuhan 430000, China)[1]

(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430000, China)[2]

Abstract In order to avoid data loss due to hardware failure or server breakdown, redundancy coding technology is widely employed in distributed storage systems for data reliability. However, traditional erasure codes, such as Reed-Solomon codes, bear the burden of huge rebuilding data amount. Compared with the replication technique, which only needs to read and transfer the lost data, the erasure coding requires to read and transfer a much large amount of data, thereby consuming much more disk I/Os and network bandwidth. Thus, a erasure code based distributed storage system would cost longer time for data reconstruction than a replication based system, and exposes the whole system in a long-term degraded stage, increasing the risk of the permanent data loss. To solve this problem, many repair-bandwidth-efficient codes were constantly proposed, but these codes are only compared with the traditional Reed-Solomon codes and lack the comprehensive comparisons on practical storage systems. We systematically analyzed these repair-bandwidth-efficient codes from the some significant aspects, such as amount reduction on reconstruction data and so on, thus providing valuable basis and references for choosing suitable erasure codes for practical systems.

Keywords Erasure codes, Data construction, Storage system, Distributed system

1 引言

在分布式存储系统中,常常采用冗余技术来保证数据的可靠性。副本和冗余编码是系统常用的两种冗余技术,而冗余编码技术相对于副本能够减少更多的存储空间。比如,在 Hadoop RAID 分布式存储系统[1]和 ceph[2]分布式存储系统中都采用里德-所罗门(Reed-Solomon, RS)码[3]代替副本冗余技术,减少了存储开销。然而,传统的冗余编码技术在实际的存储系统中遇到一个重要问题,即修复带宽问题(即修复丢失数据块时重建数据量大)。它指的是:在基于冗余编码技术的分布式存储系统中,往往把固定大小为 M 的数据等分为 k 块原始数据块,继而编码得到 m 块冗余数据块,最终将这 $n=k+m$ 个数据块分别存储在 n 个存储节点上;然而,重建任何一个存储节点上大小为 M/k 的数据块都需要 k 个数据块参与,需要从 k 个节点上读取并传输大小共计为 M 的数据量,因此需要从磁盘上读取和在网络上传输 k 倍的数据量。图 1 给出了双副本和 $(n=4,k=2)$-RS 码在修复第三个节点上相同数据量时所需要的重建数据量,(4,2)-RS 码需要重建的数据量是副本的 $k=2$ 倍。当 k 增加时,重建数据量也增加。

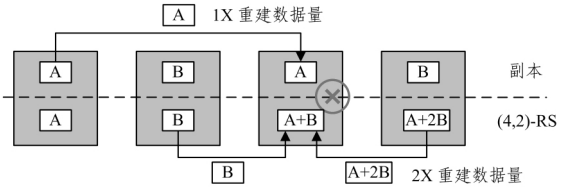


图 1 双副本和(4,1)-RS 码修复开销

马良荔(1968-),女,博士,教授,博士生导师,主要研究方向为系统结构、系统可靠性,E-mail:maliangl@163.com;柳 青(1986-),男,博士生,主要研究方向为存储系统、冗余编码、存储可靠性,E-mail:qing@hust.edu.cn。

在传统的磁盘阵列系统中，数据通过冗余编码技术编码后保存在多个磁盘中。当某个磁盘发生损坏导致数据丢失或者数据暂不可用时，系统处于一种不稳定的降级状态，如果有对丢失数据的读请求，则会产生重建/修复读操作，几倍于请求数据量的重建数据将从磁盘中被读取，消耗了大量的磁盘 I/O[4]，因此，重建时间是衡量磁盘阵列性能的重要参数之一。在基于网络的分布式存储系统中，重建数据的过程不仅消耗宝贵的磁盘 I/O，而且占用大量的网络带宽，修复开销更大[5]。修复过程中重建数据量越大，重建时间就越长，导致整个系统暴露在不稳定的降级状态下，任何进一步的数据丢失都可能导致不可逆的数据丢失，增加了数据永久性丢失的风险[6]。但也有研究指出，在现实中超过 95% 的数据节点失效都是单节点失效[7]，因此本文仅讨论单个存储节点发生数据丢失的情况，定义重建数据量为重建一个数据节点中丢失数据所需要的数据量。

近十年来，针对分布式存储系统中减少单个失效节点重建数据量的理论研究不断发展。在理论方面，Dimakis 等人推导出重建数据量和其他相关参数的约束关系，指出重建数据量存在理论上的最小值[5]。在编码方面，减少重建单个失效节点数据量的冗余编码方法也不断被提出：Rashmi 等人和 Changho Suh 等人提出使用积矩阵(product-matrix)和干扰对齐(interference alignment)的方法减少网络上传输的重建数据量，代价却是更多的磁盘 I/O[7-8]；Rashmi 等人基于积矩阵的编码方法进一步优化了磁盘 I/O，使得从磁盘中读取的重建数据量也得到减少，但要求编码码率必须不大于 1/2 ($R=k/n\leqslant 1/2$)，这限制了该编码方法在一般存储系统中的应用[9]；再生码是能够使得网络传输的重建数据量达到理论上最小的一类编码方法，但同样存在由于以增加磁盘 I/O 为代价而在实际存储系统中局限性较大[7]的问题。胡燏翀等人提出了功能性最小存储再生码(Functional Minimum Storage Regenerating codes，FMSR)[10]，同时减少了磁盘读取和网络传输的重建数据量，码率可以任意大，即原始数据在编码后的数据所占比例可以任意大。黄诚等人首次将局部重建码[11](Locally Repairable Codes，LRC)应用于 Windows 云存储系统 AZure 中，减少了重建单个数据节点时参与节点的数目和重建数据量。Papailiopoulos 等人提出了简单再生码[12](Simple Regenerating Codes，SRC)，即数据通过两层冗余编码，在重建数据时只使用其中一种冗余编码即可恢复数据，从而减少了任意单节点故障的重建开销。Z 码[13]是能够在重建单个数据节点时达到理论上存储开销最小和修复开销最小的一类编码方法。旋转里德-所罗门码[4](Rotated Reed-Solomon，RRS)是一类基于 RS 码的改进编码方法，能够在保证存储开销和系统可靠性不变的情况下减少单个失效数据节点的重建数据量。

但这些编码方法或只是理论上的提出，或实现在不同的存储系统中，如何在统一的环境中从不同指标上衡量它们的性能，并在实际存储系统中根据不同需求给以指导意见，尚属空白。因此，本文系统地研究和评估了这些编码方法，并指出它们应用于实际系统时所需要注意的问题。

本文第 2 节介绍相关背景知识；第 3 节详细分析几类典型的减少重建数据量的冗余编码方法；第 4 节系统性地评估了减少重建数据量的冗余编码方法的性能；最后总结全文。

2 相关背景

本节介绍基于冗余编码技术的分布式存储系统的相关背景知识，以为后文讨论减少重建数据量的编码方法做准备。表 1 列出了文中要用到的一些符号和相应说明。

表 1 冗余编码常用符号及其说明

符号	说明
n	一个条带中所有数据块个数
k	一个条带中原始数据块个数
m	一个条带中冗余数据块个数
d	修复时参与的存储节点个数($k\leqslant d<n$)
r	一个数据块中包含多少数据片
M	原始数据总量
D_i	条带中第 i 个原始数据块($0\leqslant i<k$)
$D_{i,j}$	第 i 个原始数据块第 j 个数据片($0\leqslant j<r$)
C_i	条带中第 i 个冗余数据块($0\leqslant i<k$)
$C_{i,j}$	第 i 个冗余数据块第 j 个数据片($0\leqslant j<r$)
GM	生成矩阵
GM_i	冗余数据块 C_i 的生成矩阵($0\leqslant i<m$)
$GM_{i,j}$	生成矩阵 GM 第 i 行第 j 列元素($0\leqslant i<mr$，$0\leqslant j<kr$)

2.1 基于冗余编码的分布式存储系统

如图 2 所示，在一个典型的分布式存储系统中，所有数据以数据块的形式存储在存储节点中。数据块大小为固定的，可以分为原始数据块和冗余数据块。若干原始数据块和它们编码得到的冗余数据块可以组成一个条带，每个条带中的数据块分别分布在不同的存储节点。条带是数据编码和修复的基本单元，一个条带中包含 n 个数据块，其中原始数据块的个数为 k，冗余数据块的个数为 m。最大距离可分码[3](Maximum Distance Separable，MDS)是一类具有最优容错性质的编码方法，它指的是 n 个数据块中任意 k 个数据块都可以恢复出原始数据，即这类编码方法在一个条带中最多可以容忍任意 m 个数据块失效。另外，存储节点上每个数据块可以进一步等分为 r 个数据片。分片的个数和编码方式有关，比如 RS 码每个数据块即对应一个分片($r=1$)，而柯西里德-所罗门码[14](Cauchy Reed-Solomon codes)则有 w 个分片($r=w>1$)，其中 w 是柯西矩阵域 GF(2^w)中 w 的大小。

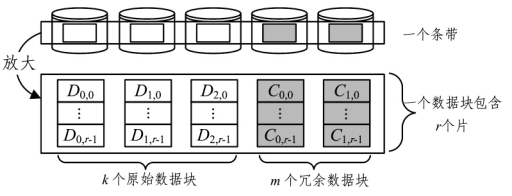


图 2 存储系统中一个条带的结构

2.2 基于生成矩阵的编码方法

冗余编码在编码理论中属于线性码，即它们将原始数据块线性组合为冗余数据块。所有的线性组合系数可以构成一个矩阵，称作生成矩阵[3](Generator Matrix，GM)，编码得到冗余数据块的过程就是计算生成矩阵和原始数据块乘积的过程。即 $GM\times[D_i]=[C_i]$，其中 $[D_i]$ 代表所有原始数据块的集合，$[C_i]$ 代表所有冗余数据块的集合。因为每个条带中有 k 个原始数据块和 m 个冗余数据块，所以生成矩阵是一个 $m\times k$ 的矩阵。对于 $r\neq 1$ 的编码，每个数据块可以继续分为 r 个数据片，那么对应的生成矩阵中的每个元素将是一个 $r\times r$ 的子矩阵，因此，(n,k)-编码方法的生成矩阵是一个 $mr\times kr$ 的矩阵。如果将生成矩阵 GM 按行等分为 m 份，则等分后的

第 i 个 $r\times kr$ 子矩阵 GM_i 可以看作是第 i 个冗余节点上 r 个数据片的生成矩阵，即 GM_i 和所有数据块乘积的结果为第 i 个冗余节点上 r 个冗余数据片。

图 3 给出了生成矩阵和 kr 个原始数据片与 mr 个冗余数据片的对应关系：矩阵中的每个行向量对应着一个冗余数据片，即该冗余数据片是将所有数据分片以该行向量中元素为组合系数线性组合得到的；矩阵中每个列向量对应着一个原始数据片，即该原始数据片以列向量中每个元素为组合系数线性组合到 mr 个冗余数据片中。

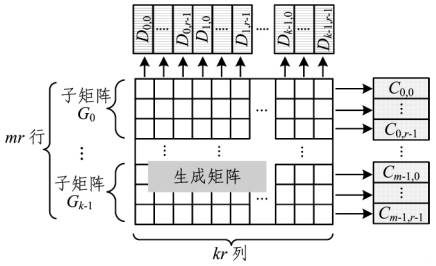


图 3　生成矩阵和原始/冗余数据片的对应关系

冗余编码所有的运算都是基于有限域 GF(2^w)的，2^w 代表域上元素的个数，当 $w=1$ 时，域 GF(2)上只有两个元素(0 和 1)，基于域 GF(2)上的冗余编码也就是常见的异或码，异或码基于的域 GF(2)上所有的运算都可以通过异或(XOR)来实现，比如 RAID5 和 CRS 码。而基于域 GF(2^w)($w>1$)上的编码运算规律更复杂，不能只通过异或来实现编码和解码，被称为非异或码，常见的非异或码基于有限域的大小为 GF(2^w)，$w=4,8,16,32$。一般来说，域越大(w 越大)，域上的冗余编码速度就越慢[3]。Planks 等人提出使用 CPU 单指令多数据流 SIMD 指令集加速有限域的运算，使得非异或码速度得到极大提升[15]。

2.3　分布式存储系统中的数据修复

在基于冗余编码的分布式存储系统中，为保证数据的可靠性和冗余度，当某数据节点失效或者节点上某数据块损坏时，需要对丢失数据进行修复。数据节点失效表示由于服务器宕机或者磁盘损坏导致该节点上所有数据块发生永久性丢失，在本文中将数据节点失效和数据块失效视为相同，因为修复失效节点需要将节点上每个丢失的数据块都进行重建。

修复节点是重建出失效数据块的存储节点，修复节点通过网络读取一些存储节点中相应的数据块/片，并通过计算重建出损坏的失效数据块。那些在修复过程中提供数据块/片以辅助修复过程的存储节点被称作辅助节点。

通过分析以往的分布式存储系统可知，超过 95% 的数据失效都是单点失效[7]，即可看作一个条带中仅有一个数据块不可用(损坏数据量为 M/k)。但即使如此，传统的冗余编码(如 RS 码)仍需要读取相当于原始数据量即 k 倍于丢失数据块大小的重建数据(M)用于恢复丢失的数据块。Dimakis 等人[5]首次通过信息流图的方式推导出修复单节点故障重建数据量所需要满足的条件，如下式所示：

$$M\leqslant\sum_{i=0}^{k-1}\min\{\alpha,(d-i)\beta\} \tag{1}$$

其中，M 代表原始数据大小，α 表示每个节点存储数据量的大小，d 表示辅助节点个数，β 表示每个辅助节点为辅助完成修复过程需要在网络上传输的数据量，那么总的重建数据量则是 $d\beta$。从式(1)不难看出，要保证原始数据大小 M 不变的情况下减少重建数据量，只有增大每个节点存储的数据量 α 或者增加辅助节点个数 d。

MDS 性质是冗余编码最优容错的一种性质，它使得具有 k 个原始数据块和 $n-k$ 个冗余数据块的(n，k)编码可以容忍任意 $n-k$ 个数据块失效，即编码后 n 个数据块中的任意 k 个可以恢复出原始数据。

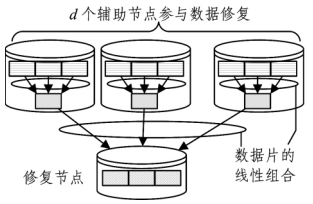
3　冗余编码的分类

根据不同冗余编码的编码方法和减少重建数据量的原理不同，可以将减少重建数据量的冗余编码分为再生码、分层码和阵列码 3 类。

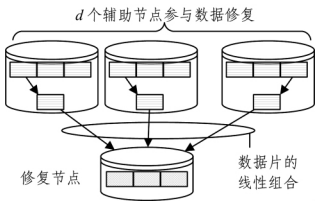
3.1　再生码

再生码是一类能够使得网络上传输重建数据量达到理论最小值的冗余编码方法。再生码不仅可以在存储最优的情况下达到理论最优重建数据量[5]，而且可以通过增加每个节点的存储数据量来进一步减少重建数据量，前者称为最小存储再生码(Minimum Storage Regenerating codes，MSR)，后者称为最小带宽再生码(Minimum Bandwidth Regenerating codes)。但再生码在构建时仅仅考虑到了网络带宽和网络流量，并使得在网络上传输的重建数据量最小，却增加了需要从磁盘上读取的数据量。具体来说，大部分的再生码采用一种两步骤的编码修复方法(见图 4(a))：1)每个参与数据修复的存储节点(也称作辅助节点)从磁盘上读取出若干数据片进行线性组合计算，得到线性组合后的组合分片；2)通过网络把这些组合分片传输到修复节点，修复节点将这些组合分片再次线性组合得到丢失的数据。这样，即使是网络上传输的数据量得到了减少，但由于每个存储节点需要读取更多的数据片参与线性组合，从磁盘读取的数据量也仍旧很大。基于 Product-matrix 的 MSR(Product-Matrix MSR，PMMSR)再生码是这类编码方法的代表。

为了解决一般再生码磁盘 I/O 大的问题，胡燏翀等人提出了功能性最小存储再生码(FMSR codes)。FMSR 码的原理都是使用非编码修复方法，即不在参与数据修复的辅助节点上进行线性组合运算，而只在修复节点上进行最终的数据重建，这样就不用读取多余的数据片，使得磁盘 I/O 和网络上的重建数据量都得到减少。图 4 中比较了编码修复和非编码修复在分布式存储系统上的实现。



(a)编码修复



(b)非编码修复

图 4　编码修复和非编码修复的区别

由图 4 可知，相对于编码修复，非编码减少了在每个辅助

节点中读取更多数据片并进行计算的开销，节省了磁盘 I/O。虽然它们均减少了磁盘 I/O 的重建数据量，但两种编码在其他特性上却有所取舍。FMSR 使用的是一种非系统码的编码方式，即编码后数据不包含原始数据块，条带中所有的数据块都是冗余数据块，这样即使是正常读取原数据，也需要解码操作。PMMSR 虽然是系统码（编码后包含原始数据），但它要求码率不大于 1/2，即一个条带中冗余数据块个数不少于原始数据块。

再生码的另外一个重要分类是最小带宽再生码（MBR）。最小带宽指的是在修复一个存储节点失效时所需要的重建数据量最小，即重建数据量等于该失效节点所存储的数据量。图 5 给出了一个精确修复的最小带宽再生码（Exact MBR，EMBR）的例子，图 5(a)是原始数据块编码得到冗余数据块，图 5(b)是每个数据块的放置方式。每个存储节点保存 4 个数据块，重建数据量同为 4 个数据块。首先，数据通过一个 $(\frac{n(n-1)}{2},\frac{n(n-1)}{2}-\frac{m(m-1)}{2})$-MDS 码（如 RS 码）编码得到 $\frac{n(n-1)}{2}-\frac{m(m-1)}{2}$ 个原始数据块和 $\frac{m(m-1)}{2}$ 个冗余数据块，然后将这 $\frac{n(n-1)}{2}$ 个数据块按照完全图的形式放置于 n 个存储节点上，每个节点放置 $n-1$ 个数据块。放置规则为：将 n 个存储节点视为完全图的所有节点，节点之间用无向边相连，共 $\frac{n(n-1)}{2}$ 条边，每条边对应且唯一对应着一个原始数据块或冗余数据块，每个节点保存的数据块为所连接边上对应的数据块。在修复一个存储节点的失效时，其余 $n-1$ 个节点只需要读取和传输与该失效节点连接的边所对应的数据块。如图 5(b)中右上和右下节点所连接的边代表的数据块为 D_6，那么当这两个节点中任意一个失效时，一个节点传输给另一个节点的数据块为 D_6。

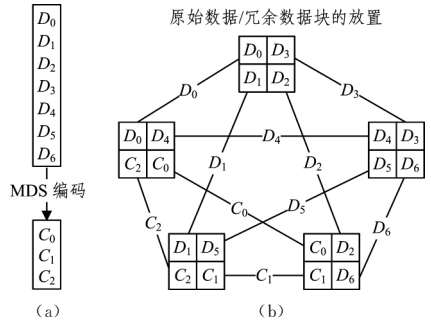


图 5 $(n=5,k=3)$-EMBR 最小带宽再生码

最小带宽再生码通过增加每个节点的存储开销（即增大式(1)中 α）减少重建数据量，其代价之一就是增加了存储系统所需占用的存储空间。因此，EMBR 码比较适合于带宽受限较大但存储空间充裕的存储系统（比如 P2P 存储系统），而不适合按存储容量计费的云存储系统。

3.2 分层码

分层码采用一种背负式（piggybacking）技术将一种冗余编码方案应用在另一种冗余编码方案上，最早应用于 P2P 网络中。本节分析局部修复码（Locally Repairable Codes，LRC）和简单再生码（Simple Regenerating Codes，SRC），它们分别是水平分层码和垂直分层码。虽然 SRC 码使用的是再生码名称，但其原理仍属于分层码。

LRC 码属于水平分层码，水平指的是同一条带原始数据块和编码生成的冗余数据块分别水平地保存在物理位置不同的存储节点。LRC 码减少重建数据的基本原理是将条带内的原始数据块分组，在组内生成一个局部冗余数据块，使得组内的单个数据块重建在组内就可以完成，而不需要其他组的数据块参与。另一方面，为了保证容忍组内多个数据块失效，LRC 码还保留了整个条带的全局冗余数据块作为所有数据块的编码结果，只有在一组内两个以上（冗余）数据块失效时才使用全局冗余数据块进行数据恢复。LRC 码不但减少了修复单个数据块的重建数据量，而且减少了需要参与修复的节点个数 d，但它也以牺牲编码的 MDS 性质为代价，即它不能够容忍最大程度的数据丢失。图 6 给出了一个(9,6,2)-LRC 码的例子，6 个数据块分为两组，每组生成一个局部冗余数据块 L_i，分别为 L_0 和 L_1，全部 6 个数据块共同计算得到全局冗余数据块 C_0。当第一组数据块 D_1 丢失时，仅需要 D_0，D_2 和 L_0 即可完成修复；当组内有更多数据块发生失效时，则需要全局冗余数据块参与修复。LRC 码不能容忍最大数目的数据块失效，同样可以从图中看出，如 $(n=9,k=6,2)$-LRC 并不能像其他 MDS 码一样容忍任意 3 个数据块的失效。

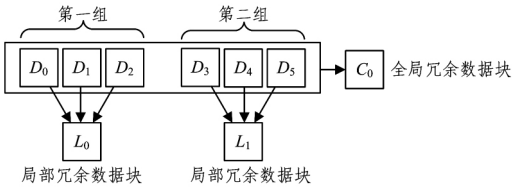


图 6 (9,6,2)-LRC 码

SRC 码是垂直分层码，即同一个条带中原始数据块和冗余数据块垂直放置，同一个节点既存储了原始数据块，也存储了冗余数据块。SRC 的基本原理也是将原始数据分组编码，再在组间通过异或码生成组间编码，在进行单个节点失效数据的重建时，只需使用组间编码的关系即可。以(5,3,2)-SRC 码为例，如图 7(a)是数据编码过程，原始数据分为两组，分别使用(5,3)-MDS 码（如(5,3)-RS 码）编码得到 5 个数据片，然后两组间数据片使用异或编码得到一组新的全冗余数据片，再将所有数据片按照图 7(b)所示放置：一组一行，最后一行放置组间奇偶校验得到的全冗余数据片，且组间进行校验的原始数据块和校验块放在不同节点（如 D_0，D_3 和 D_0+D_3 分别存放于节点 1，2 和 3）。当有一个节点发生失效时（图 7(b)节点 3 失效），只需要在组间进行奇偶校验的数据块参与修复即可。

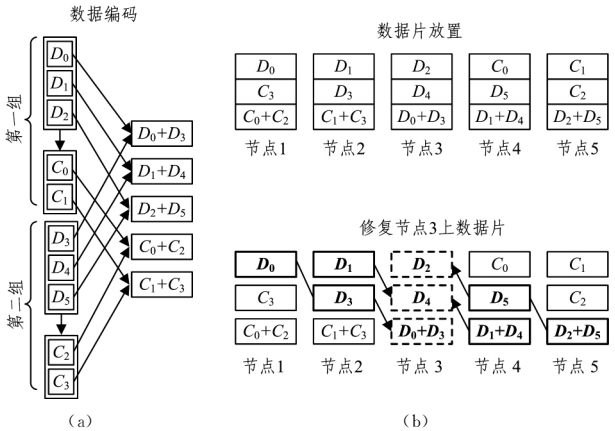


图 7 (5,3,2)-SRC 码编码、数据放置和修复过程

相对于再生码提出了新的编码方法，分层码借鉴已有的传统冗余编码方法，使用多层次的编码方案并优化数据布局，从而减少了重建数据量。分层码的编解码和修复性能一方面依赖于所使用的传统冗余编码，另一方面依赖于构建分层码的方法。另外，分层码从系统结构上契合了存储系统中分层存储的思想，在系统实现和实际效果上比其余两种编码方法更具亲和力。

3.3　阵列码

阵列码自磁盘阵列诞生起便随之发展，它是一种应用于多磁盘阵列的高性能冗余编码。基于磁盘阵列特性，阵列码具有速度快、存储效率高和 MDS 性质等特点。但阵列码的容错数量往往较少，如容一错的奇偶校验码，容两错的 EVENODD 码、X 码、RDP 码[3]和容三错的 STAR 码[16]。得益于磁盘阵列在过去 20 年中得到的广泛应用，围绕阵列码的相关方向也得到广泛且深入的研究，包括基于减少重建数据量的单磁盘失效快速重建方法。这类基于磁盘阵列的快速重建方法为在分布式存储系统中减少重建数据量提供了很好的思路和方向。如何将这类减少重建数据量的方法应用于分布式存储系统中，可以基于两个方向进行研究：1）如何进一步减少重建数据量，因为现有阵列码快速重建一般依赖于穷举修复策略寻找最优的重建方案，但由于编码方法的限制，减少的重建数据量有限；2）如何在进一步减少重建数据量的同时容更多节点失效。

为了达到容更多错和进一步减少重建数据量的目标，研究者一般牺牲了阵列码的一些特性，如 MDS 性质、存储效率等。Butterfly 码是一种容忍两个错的阵列码，优点是修复单个原始数据块失效能够达到理论最小重建数据量，且 Butterfly 码还为 XOR 码，也满足 MDS 性质，不足之处在于其只能容忍两个错。Plank 等人提出了 RRS 码，即一种基于 RS 码的改进，相对于 RS 码每个节点的数据块只包含一个数据片（$r=1$），RRS 码中每个节点数据块可包含更多的数据片（$r=2,4,8$ 等），通过合理选择组合方法和组合系数减少了单个数据块的重建数据量。RRS 码能够容忍 2 个以上节点失效，但 RRS 码不是 XOR 码，计算速度慢，且 RRS 码是否满足 MDS 性质尚无理论证明。与 Butterfly 码一样，Z 码也能在修复单个原始数据块时达到理论最小重建数据量，而且也是 XOR 码。Z 码基于置换矩阵，具有一般性的构造方法，能够容忍任意数量节点失效。Z 码虽然不是 MDS 码，但如果将其应用于有限域 $GF(2^w)(w>1)$ 中，则有可能满足 MDS 性质。

4　冗余编码的评估

4.1　重建数据量及其开销

从 Dimakis 关于重建数据量的理论最小值式（1）可以看出，增加每个节点的存储开销 α 和修复度 d 即参与重建数据的辅助节点个数，是两种减少重建数据量的方法。表 2 列出了几种常见的减少重建数据量冗余编码的重建数据量的参数（以 RS 码作为参照）。除了分层码中的 LRC 码和 SRC 是通过牺牲 MDS 性质或增加存储开销达到减少重建数据量的目的外，几乎所有的编码方法都通过增加修复度来减少重建数据量。

表 2　常见的减少重建数据量冗余编码方法及其特点

编码方法	码率	每个节点的存储量	修复度	重建数据量
RS	$\frac{k}{k+m}$	$\frac{M}{k}$	k	M
RRS	$\frac{k}{k+m}$	$\frac{M}{k}$	$k+1$	$(\frac{1}{2}+\frac{1}{2k}\lceil\frac{k}{m}\rceil)M$
FMSR	$\frac{k}{k+m}$	$\frac{M}{k}$	$n-1$	$\frac{n-1}{k(n-k)}M$
LRC	$\frac{k}{k+m+l}$ †	$\frac{M}{k}$	l	$\frac{l}{k}M$
SRC	$\frac{kf}{n(f+1)}$ ‡	$\frac{(f+1)}{kf}M$	$2f$	$\frac{f+1}{k}M$
EMBR	$\frac{k(n+m-1)}{2n(n-1)}$	$\frac{2(n-1)}{k(n+m-1)}M$	$n-1$	$\frac{2(n-1)}{k(n+m-1)}M$
Z	$\frac{k}{k+m}$	$\frac{M}{k}$	$n-1$	$\frac{n-1}{k(n-k)}M$
PMMSR	$\frac{k}{k+m}\leqslant\frac{1}{2}$	$\frac{M}{k}$	$n-1$	$\frac{n-1}{k(n-k)}M$

注：† l 表示 LRC 码中的分组个数；‡ f 表示 SRC 码中的分组个数。

图 8 比较了几种编码方法在具体的两种参数（$n=6,k=4$ 和 $n=12,k=9$）下的重建数据量和存储开销。存储开销是数据总量（原始数据量与冗余数据之和）与原始数据量的比值，重建数据量是指修复单个节点失效所需重建数据量与原始数据量的比值。点线是理论上修复单个存储节点所需要的最小重建数据量。在存储开销最小的情况下，FMSR 码和 Z 码可以达到理论最小的重建数据量。同样是最小存储开销，RRS 码则需要更多重建数据量，当参数 n 和 k 变大时，RRS 重建数据量和理论最小重建数据量的差距也在增加。局部冗余数据块的增加导致 LRC 码的存储开销略大于最小存储开销，但其重建数据量比较接近于最小重建数据量。由于 LRC 码和 Z 码都牺牲了 MDS 性质，对系统可靠性的影响未知，因此，在下节将通过分析它们的平均数据丢失时间来评估对系统可靠性的影响。

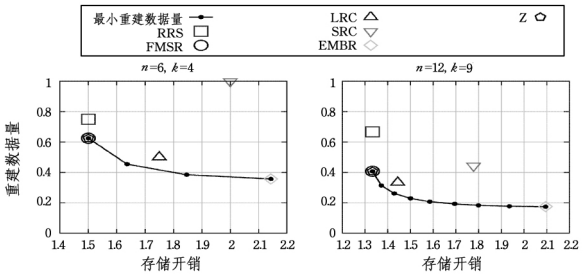


图 8　不同编码方法的存储开销和重建数据量

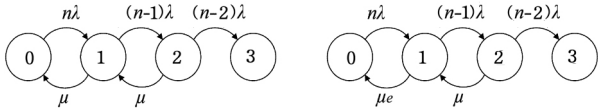
4.2　系统可靠性

为分析基于减少重建数据量冗余编码的分布式存储系统的可靠性，建立马尔科夫状态模型，通过计算基于不同冗余编码的系统平均数据丢失时间（Mean Time To Data Loss，MTTDL）来衡量基于这些编码的存储系统的可靠性。

MTTDL 的含义是存储系统从开始运行到发生数据丢失的时间期望。基于某种冗余编码的存储系统 MTTDL 不仅与该编码方法有关，还与单个存储节点的失效率 λ 和单个存储节点的修复率 μ 有关。一般来说，失效率 λ 越高，数据越容易发生丢失，MTTDL 越小，系统可靠性越低。相反地，修复率 μ 越高，数据越快地被重建和恢复，MTTDL 越大，系统可靠性越高。由于减少修复带宽的编码能够减少恢复单个失效节点的重建数据量，因此单个节点的修复率为 μ_e，且 $\mu_e<\mu$。

下面以（$n=4,k=2$）-MDS 码为例，分析基于冗余编码的存储系统 MTTDL。图 9 给出了基于普通冗余编码和减少重建数据量冗余编码存储系统的马尔科夫状态模型。图中圆圈

表示状态，状态 0 为起始状态，表示所有存储节点都正常；状态 i 代表有 i 个存储节点失效，有向箭头表示状态之间的转移，值表示转移概率。状态 3 为该模型的终态，因为当有 3 个存储节点失效时，数据发生不可逆的丢失。通过计算非终态的其他状态概率，可以得到系统处于所有非终态状态的时间期望，即 MTTDL。



(a)普通冗余编码 (b)减少重建数据量冗余编码

图 9 $(n=4,k=n-2)$-MDS 码和减少重建数据量冗余编码的马尔科夫状态模型

$$MTTDL=\frac{\mu^m+P_1(n)\mu^{m-1}\lambda+\cdots+P_{m-1}(n)\mu\lambda^{m-1}+P_m(n)\lambda^m}{\prod_{i=0}^{m}(n-i)\lambda^{m+1}}\approx\frac{\mu^m}{\prod_{i=0}^{m}(n-i)\lambda^{m+1}} \tag{2}$$

在式(2)中，可以做近似计算是因为：修复率远大于故障率即 $\mu\gg\lambda$，且 $P_i(n)$ 是一个关于 n 的 i 次多项式。同样地，计算得到减少重建数据量冗余编码的存储系统 MTTDL(记作 $MTTDL_e$)。

$$MTTDL_e\approx\frac{\mu^{m-1}\mu_e}{\prod_{i=0}^{m}(n-i)\lambda^{m+1}} \tag{3}$$

其中，μ_e 即减少修复带宽冗余编码的单节点修复率。从式(2)和式(3)可以看出，系统可靠性与修复率(μ)成正比，与故障率(λ)和节点总个数(n)成反比。

图 10 在相同 n 与 k 参数的情况下比较了几种减少重建数据量冗余编码的归一化 MTTDL。归一化 MTTDL 是以相同参数的(n,k)-RS 码的 MTTDL 为基准，计算得到其他冗余编码的 MTTDL 与其的比值结果。所有的编码方法都具有 MDS 性质，即最多能容忍 m 个存储节点失效。从以 MTTDL 作为可靠性衡量的结果不难看出，所有基于减少重建数据量冗余编码的存储系统的可靠性均不低于 RS 码的可靠性。对于更大的参数对(n,k)，能够减少的重建数据量更多，修复率 μ_e 更大，系统可靠性也越高。EMBR 和 SRC 码以存储开销换取重建数据量的减少，但 EMBR 能够减少更多重建数据量，MTTDL 也就越高。RRS，FMSR 和 GZ 码都是最小存储(与 RS 码相同存储开销)的冗余编码，其中 GZ 是将 Z 码应用于有限域 $GF(2^8)$ 得到的编码方法，是满足 MDS 性质的冗余编码。3 种编码中，FMSR 码能够减少所有节点的重建数据量，因此其可靠性高于只能够减少数据节点重建数据量的 GZ 码和 RRS 码的可靠性。而相比 RRS，GZ 码能够减少更多的重建数据量，因此可靠性更高。

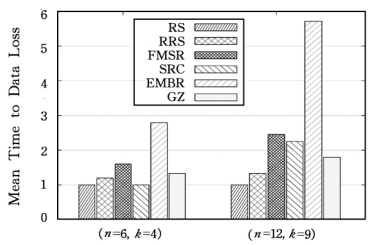


图 10 不同的减少重建数据量冗余编码的归一化 MTTDL

LRC 码虽然减少了重建数据量，但却以牺牲 MDS 性质为代价。表 3 比较了基于(6，4)-RS，(7，4，2)-LRC 和(7，4)-RS 码存储系统的 MTTDL。首先，(7，4)-RS 码的 MTTDL 高出(6，4)-RS 3 个数量级，极大地提高了系统可靠性，由此可见，增加冗余节点是提高存储系统可靠性的重要方法；其次，(7，4，2)-LRC 码具有 3 个冗余节点，其中 2 个是局部冗余节点，1 个是全局冗余节点，最少容任意两个节点失效，最多容忍 3 个特定节点失效。结果其可靠性要优于最多容两错的(6，4)-RS 码，却低于最多容三错的(7，4)-RS 码。

表 3 基于(6，4)-RS，(7，4，2)-LRC 和(7，4)-RS 码存储系统的 MTTDL/h

	(6，4)-RS	(7，4，2)-LRC	(7，4)-RS
MTTDL	8.367×10^8	3.288×10^9	4.762×10^{11}

4.3 编码性能

在一个具有 Intel Core i7-4770K 的 CPU 和 16GB 内存的单机上测试了不同编码的编码速度和修复速度。编码速度是原始数据编码为数据块与冗余数据块的过程，其值为原始数据大小与所消耗时间的比值；修复速度指的是单个数据块的重建速度，即是单个数据块大小与从数据片和重建时间开销的比值。为加速有限域上的计算，使用 GF-Complete 库加速计算过程。

图 11 和图 12 分别给出了不同冗余编码的编码速度和修复速度。原始数据大小为 1GB，每次测试循环 50 次，取平均值为最终结果。所有冗余编码的参数(n,k)分别为(4，2)，(6，3)和(8，4)。由于 RRS 码仅在 $m=2$ 或 3 时存在，因此未画出 RRS 码在(8，4)参数下的编码速度和修复速度。没有考虑 EMBR，LRC 和 SRC 码的速度，是因为这 3 个冗余编码是基于已有编码方法的，如 RS 和 CRS 码，这 3 种编码的速度取决于这些已有冗余编码的速度。

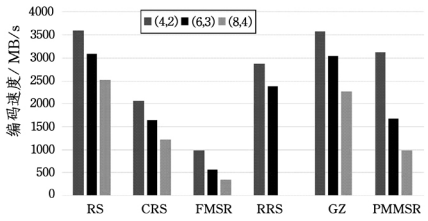


图 11 不同冗余编码的编码速度

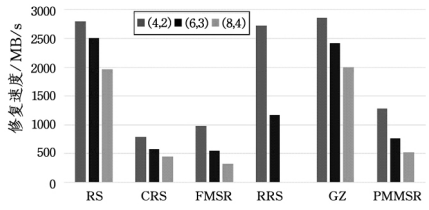


图 12 不同冗余编码的单数据块修复速度

因为使用 GF-Complete 加速有限域上的计算，所以基于有限域上的 RS 码要比异或码 CRS 码的编码和修复速度快。但 FMSR 由于是非系统码，所有编码后的数据以冗余数据存在，因此编码速度在所有冗余编码中最慢。RRS，GZ 和 PMMSR 码的编码速度均能达到 GB/s，但 PMMSR 修复速度较慢，原因是其两步修复和需要读取较多数据。但所有冗余编码的编码速度和修复速度均能够超过一般磁盘的读写速度，这使得计算将不会成为存储系统的瓶颈。

结束语　随着存储系统中数据量的急剧增加和冗余编码在存储系统中越来越广泛地被应用，存储失效导致的数据重建问题越来越引起冗余编码领域的重视。本文对近年提出的减少重建数据量的冗余编码进行了系统分析，从编码的实现原理、基于冗余编码系统的特性与开销和不同冗余编码的编码性能方面进行了讨论，为不同减少重建数据量的冗余编码在实际系统中的应用提供了参考。

参 考 文 献

[1] BORTHAKUR D, SCHMIDT R, VADALI R, et al. Hdfs raid [C]// Hadoop User Group Meeting. 2010.

[2] 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. USENIX Association, 2006: 307-320.

[3] PLANK J S, DING Y. Note: Correction to the 1997 tutorial on Reed-Solomon coding[J]. Software: Practice and Experience, 2005, 35(2): 189-194.

[4] 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]// FAST. 2012: 20.

[5] DIMAKIS A G, GODFREY P, WU Y, et al. Network coding for distributed storage systems[J]. IEEE Transactions on Information Theory, 2010, 56(9): 4539-4551.

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

[7] RASHMI K V, SHAH N B, GU D, 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]// Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems. 2013.

[8] 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.

[9] RASHMI K V, NAKKIRAN P, WANG J, et al. Having your cake and eating it too: jointly optimal erasure codes for I/O, storage, and network-bandwidth[C]// 13th USENIX Conference on File and Storage Technologies (FAST 15). 2015: 81-94.

[10] HU Y, CHEN H C H, LEE P P C, et al. NCCloud: applying network coding for the storage repair in a cloud-of-clouds[C]// FAST. 2012: 21.

[11] HUANG C, SIMITCI H, XU Y, et al. Erasure coding in windows azure storage[C]// Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12). 2012: 15-26.

[12] PAPAILIOPOULOS D S, LUO J, DIMAKIS A G, et al. Simple regenerating codes: Network coding for cloud storage[C]// 2012 Proceedings IEEE INFOCOM. IEEE, 2012: 2801-2805.

[13] LIU Q, FENG D, JIANGY H, et al. Z Codes: General Systematic Erasure Codes with Optimal Repair Bandwidth and Storage for Distributed Storage Systems[C]// 2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS). IEEE, 2015: 212-217.

[14] PLANK J S, XU L. Optimizing Cauchy Reed-Solomon codes for fault-tolerant network storage applications[C]// Fifth IEEE International Symposium on Network Computing and Applications, 2006(NCA 2006). IEEE, 2006: 173-180.

[15] PLANK J S, GREENAN K M, MILLER E L. Screaming fast Galois field arithmetic using intel SIMD instructions[C]// FAST. 2013: 299-306.

[16] HUANG C, XU L. STAR: An efficient coding scheme for correcting triple storage node failures[J]. IEEE Transactions on Computers, 2008, 57(7): 889-901.

（上接第 441 页）

结束语　本文针对提高矩阵分解技术的推荐质量和在大数据环境下突破计算时间、计算资源瓶颈等问题进行研究，提出了融入邻居信息的矩阵分解算法，并且克服了融入邻居信息后计算复杂度高和难以并行化等问题。最后，在 MapReduce 并行计算框架下实现了所提并行计算。

参 考 文 献

[1] ADOMAVICIUS G, TUZHILIN A. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions[J]. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(6): 734-749.

[2] SALAKHUTDINOV R, MNIH A. Probabilistic Matrix Factorization[M]// Advances in neural information processing systems, NIPS'08. Cambridge, Massachusetts, USA, MIT Press, 2008: 1257-1264.

[3] ZHANG Z J, LIU H. Social Recommendation Model Combining Trust Propagation and Sequential Behaviors[C]// Applied Intelligence. 2015.

[4] LIU Q, WANG C W, XU C F. A modified PMF modelincorporating implicit item associations[C]// Proceedings of 24th International Conference on Tools with Artificial Intelligence. Athens, Greece, 2012.

[5] CHAKROUN I, Haber T, AA T V. Exploring Parallel Implementations of the Bayesian Probabilistic Matrix Factorization [C]// Parallel, Distributed, and Network-Based Processing (PDP). 2016.

[6] GEMULLA R, HAAS P J, NIJKAMP E, et al. Large-Scale matrix factorization with distributed stochastic gradient descent [C]// Proc. of the 17th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. ACM Press, 2011: 69-77.

[7] RECHT B, RÉ C. Parallel stochastic gradient algorithms for large-scale matrix completion[J]. Mathematical Programming Computation, 2013, 5(2): 201-226.

[8] 印鉴，王智圣，李琪，等. 基于大规模隐式反馈的个性化推荐[J]. 软件学报，2014，25(9)：1953-1966.

[9] LÄmmel R. Google's MapReduce programming model-revisited [J]. Science of Computer Programming, 2007, 68(3): 208-237.

[10] 王全民，苗雨，何明，等. 基于矩阵分解的协同过滤算法的并行化研究[J]. 计算机技术与发展，2015，25(2)：55-59.

[11] Cloudera. Cloudera[EB/OL]. http://www.cloudera.com.