分布式存储中的纠删码容错技术研究

王意洁 许方亮 裴晓强

(国防科学技术大学并行与分布处理国家重点实验室 长沙 410073)

(国防科学技术大学计算机学院 长沙 410073)

摘 要 大数据规模上体量大和增长速度快的特点对存储系统的性能和可扩展性提出了严峻挑战.使用普通商用服务器构建的分布式存储系统服务能力强、成本低廉且极易扩展,在大数据的存储管理中得到了极为广泛的应用.分布式存储系统庞大的节点数量导致节点失效情况频发,必须采用一定的容错技术来保证数据可靠性.常用的容错技术主要包括多副本技术和纠删码技术两种.与多副本容错技术相比,纠删码容错技术能够以低得多的存储开销提供相同甚至更高的数据可靠性.随着近年来数据规模的爆炸式增长,纠删码容错技术受到了业界的广泛关注.该文综述了分布式存储中纠删码容错技术的研究现状.首先,介绍了纠删码容错技术的基本原理和概念,指出了纠删码容错技术在大规模分布式存储中面临的主要技术挑战;然后,从编码实现、纠删码设计、数据修复和数据更新等方面阐述了分布式存储中纠删码容错技术的研究进展,重点研究分析了各项关键技术的特点和局限性,并依据主要评价指标对现有纠删码的编码性能和修复性能进行了对比和分析;最后,基于最新研究动态指出了分布式存储中纠删容错技术未来的研究方向,包括同步编码实现技术、低冗余再生码设计和数据失效预测技术等.

关键词 分布式存储;纠删码;编码实现;数据修复;数据更新

中图法分类号 TP391 **DOI 号** 10.11897/SP.J.1016.2017.00236

Research on Erasure Code-Based Fault-Tolerant Technology for Distributed Storage

WANG Yi-Jie XU Fang-Liang PEI Xiao-Qiang

(*National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha* 410073)

(*College of Computer, National University of Defense Technology, Changsha* 410073)

Abstract Storing and managing big data, whose volume is extremely large and keeps growing rapidly, is a big challenge. Distributed storage systems built from inexpensive commodity hardware, which are able to offer extremely high performance and high scalability with low economic cost, are widely used for storing and managing big data. However, the large amount of storage nodes in distributed storage systems makes node failures common in their daily operations. This makes it essential to introduce data redundancy so that data reliability is guaranteed. Replication and erasure coding are two common approaches used to protect data from node failures. Compared to replication, erasure coding incurs much lower storage overheads and can offer the same or even higher data reliability at the same time. For this reason, with the rapid growth of data, erasure coding has gained comprehensive attention recently. This paper summarizes the research status of erasure coding in distributed storage systems. Firstly, we introduce the basic idea and main concepts of erasure coding, and point out the main technical challenges of integrating erasure coding into

收稿日期:2016-06-01;在线出版日期:2016-09-19.本课题得到国家自然科学基金(61379052)、国家重点研发计划项目(2016YFB1000101)、国家"八六三"高技术研究发展计划项目(2013AA01A213)、湖南省自然科学杰出青年基金项目(14JJ1026)、高等学校博士学科点专项科研基金资助课题(20124307110015)资助.**王意洁**,女,1971 年生,博士,教授,博士生导师,中国计算机学会(CCF)杰出会员,主要研究领域为分布式存储、大数据分析和云计算等.E-mail:wangyijie@nudt.edu.cn.**许方亮**,男,1989 年生,博士研究生,主要研究方向为纠删码、分布式存储.**裴晓强**,男,1986 年生,博士研究生,主要研究方向为分布式存储、纠删码.

large-scale distributed storage systems. Secondly, we provide a comparison and analysis of the latest research in the field from the aspects of data encoding technologies, design of erasure codes, data repair technologies, data update technologies and so on. We also provide a comprehensive comparison of common erasure codes from the aspects of data encoding and data repair. Finally, we point out some future work that can promote the further development of erasure coding in distributed storage systems, including synchronous data encoding, regenerating codes with low redundancy and data failure forecasting.

Keywords　distributed storage; erasure codes; data encoding; data repair; data update

1　引　言

进入大数据时代，数据在规模方面的显著特点是体量庞大和增长迅速[1-2]. 例如，欧洲中等范围天气预报中心在 2015 年 11 月存储的原始数据达到 87 PB，并且每月增长约 3 PB①，年化增长率为 41%；根据 2016 年 3 月的数据，社交网站 Facebook 上用户分享的图片在过去 6 年间增长了 20 倍②，平均每年增长 65%；从 2012 年到 2016 年 3 月，云存储平台 Dropbox 上的数据量从 40PB 增长到了 500PB③，年复合增长率高达 88%.

规模庞大并且仍然在不断迅速增长的数据，对构建良好的存储系统提出了重大挑战. 系统既要有极高的数据存取性能，也要有良好的可扩展性，可以在不影响系统正常运行的前提下动态地增加系统规模并获得相应的性能提升. 庞大的数据规模对系统的经济成本也提出了严格要求. 传统基于 RAID (Redundant Array of Independent Disks)[3] 的单点存储系统或基于 SAN(Storage Area Network)[4] 的网络存储系统等都无法同时满足大数据存储在性能、可扩展性和经济成本等方面的要求. 构建于大量廉价商用硬件之上的分布式存储系统，不仅可以通过并行访问提供极高的数据存取性能，也可以通过增加存储节点增大规模并提升性能，并且成本低廉. 因此，分布式存储系统在大数据的存储管理中得到了极为广泛的应用. 例如，谷歌的文件系统 GFS(Google File System)[5] 和 Hadoop 中被广泛采用的存储系统 HDFS(Hadoop Distributed Storage System)[6] 就是该类存储系统的典型代表.

数据容错是大规模分布式存储中一项不可或缺的关键技术. 由于数据量极为庞大，该类存储系统往往包含几千甚至几万个存储节点[5,7-8]. 例如，百度公司单个集群的节点数量在 2014 年就超过了 10000 台[9]. 庞大的节点数量使得节点失效成为常态[5,7,10-11]. 近年来一些大型系统中的统计数据表明，平均每天都会有 1%～2% 的节点发生失效[8,12]. 因此，采用一定的数据容错技术，从而保证在部分存储节点失效的情况下数据仍然能够被正常地访问就显得尤为重要.

目前，常见的数据容错技术有两种：一种是多副本容错技术[13]，通过复制进行容错；另一种是纠删码容错技术[14]，通过编码进行容错. 与多副本容错技术相比，纠删码容错技术可以在显著降低存储空间消耗的同时提供相同甚至高得多的数据容错能力[15-16]. 例如，与被广泛采用的三副本相比，(14,10)-RS(Reed-Solomon)纠删码[17] 既可将存储空间消耗降低 53%，也可将容错能力提高一倍. 随着大数据时代数据规模的爆炸式增长，容错能力强且存储成本低的纠删码容错技术受到了广泛关注，成为了存储领域的一个研究热点.

目前，国内外关于纠删码容错技术研究的综述文献较少，主要是关于纠删码在 RAID 存储系统中的应用[18-19]. 但是与 RAID 存储系统相比，大规模分布式存储系统具有不同的特点，其中纠删码容错技术面临的技术挑战也不尽相同. 除了纠删码设计，近年来也涌现出了很多从编码实现、数据修复和数据更新等其它方面优化纠删码容错技术的研究工作. 本文将介绍大规模分布式存储中纠删码容错技术面临的主要技术挑战，以及围绕这些技术挑战的最新研究进展，并指出大规模分布式存储中纠删码容错技术的未来研究方向.

本文第 2 节介绍纠删码容错技术的基本原理并明确常用的相关概念；第 3 节介绍纠删码容错技术在大规模分布式存储中面临的主要技术挑战；第 4

① http://www.ecmwf.int/en/earthserver-2
② https://code.facebook.com/posts/1737605303120405
③ https://blogs.dropbox.com/tech/2016/03/magic-pocket-infrastructure/

节介绍应对编码实现挑战的研究进展;第 5 节和第 6 节分别介绍从纠删码设计和数据修复技术两个方面应对数据修复挑战的研究进展;第 7 节介绍应对数据更新挑战的研究进展;第 8 节介绍有关纠删码的其它研究内容;第 9 节展望未来的研究方向;最后,第 10 节总结全文.

2 纠删码的基本原理和概念

通常,纠删码[15-16]可以用三元组(n,k,k')来表示,其中$n>k'\geq k$.一个(n,k,k')-纠删码将大小为M的数据对象O划分成k个大小均为M/k的数据块$o_1,o_2,\cdots,o_k$,然后通过相应的编码算法对这些数据块进行运算,得到n个编码块$c_1,c_2,\cdots,c_n$,并保证使用这n个编码块中的任意k'个都能解码恢复出原始数据对象O.

2.1 纠删码的基本原理

目前,存储系统采用的纠删码都是线性的,本文也只关注线性纠删码.如式(1)所示,在线性纠删码中,每个编码块c_i都可表示成所有数据块的线性组合.其中$g_{ij}\in F_q$,F_q是包含q个元素的伽罗瓦域(Galois Field).

$$(g_{i1}\ \ g_{i2}\ \ \cdots\ \ g_{ik})\times\begin{pmatrix}o_1\\o_2\\\vdots\\o_k\end{pmatrix}=c_i,\ i=1,2,\cdots,n \quad (1)$$

与编码过程相同,在线性纠删码中,失效块的修复也是通过对可用块进行线性组合来完成.可用块的选择以及相应组合系数的计算由具体的纠删码设计决定.例如,对于经典的 RS 码[17],修复任何一个块都需要下载k个块;对于近来提出的改进型纠删码,如 LRC(Locally Repairable Code)[20]等,在多数失效情况下进行修复需要的块数目都小于k,只在多块失效等少数情况下才会大于k.

2.2 纠删码的相关概念

目前,对存储系统中纠删码容错技术的相关概念尚无一致的定义.为了便于本文的描述与理解,现对本文常用的相关概念明确如下:

(1)数据块(Data Block).用户存储到系统中的原始数据对象被系统划分后产生的块.

(2)编码块(Coded Block).数据块经过编码算法运算后产生的所有块.

(3)条带(Stripe).独立地与一个纠删码算法相关的所有编码块所构成的集合.例如,一个(n,k,k')-纠删码的条带包含n个编码块.

(4)系统码(Systematic Code).如果一个纠删码产生的编码块包含条带内的所有数据块,即满足$\{o_1,o_2,\cdots,o_k\}\subseteq\{c_1,c_2,\cdots,c_n\}$,则称该纠删码为系统码.

对于系统码,在条带内没有编码块失效时,原始数据对象可以直接读取,而无需解码.这个性质对存储系统至关重要.因此,用于存储系统的纠删码几乎全都是系统码.在没有特殊说明的情况下,后文所讨论的纠删码都指系统码.

(5)精确码(Exact Code).如果一个纠删码进行数据修复时所恢复出的编码块与丢失的编码块在内容上完全相同,则称这一纠删码为精确码.

任何线性精确码都可转化为系统码[21].因此,用于存储系统的纠删码也几乎全部都是精确码.在没有特殊说明的情况下,后文所讨论的纠删码都指精确码.

(6)校验块(Parity Block).系统码产生的所有编码块中除数据块以外的部分称为校验块.在不需要区分数据块和校验块时,后文统一用编码块或块来指代.

(7)容错能力(Fault Tolerance).一个条带可以容忍的最大任意块失效数目.假设一个纠删码的容错能力为t,则此纠删码能且只能在任意不多于t个块失效的情况下,恢复出原始数据.一个(n,k,k')-纠删码的容错能力为$t=n-k'$.

(8)MDS码(Maximum Distance Separable Code).如果一个(n,k,k')-纠删码满足$k'=k$,则称此纠删码满足 MDS 性质,也称该纠删码为 MDS 码.MDS 码可以用更简单的二元组(n,k)来表示.MDS 码在相同的容错能力下拥有最小的存储空间开销.

(9)数据修复(Data Repair).当一个条带中的一个或多个编码块因节点或磁盘失效等原因而丢失时,利用条带内剩余的编码块恢复出丢失的编码块的过程称为数据修复.

3 主要挑战

采用纠删码进行数据容错不仅可以极大地降低系统的存储空间消耗,甚至同时还可以提供更高的数据可靠性.

早期纠删码主要应用在基于 RAID[3]的存储系统中.由于以前芯片的运算能力有限,早期关于纠删

码的研究主要关注其计算效率问题.传统纠删码的复杂性主要来源于编解码过程中大量伽罗瓦域上的运算.为此,涌现出了众多只需进行异或运算的纠删码,极大降低了计算的复杂度.该类纠删码包括由RS码[17]优化而来的柯西RS码[22],用于RAID的EVENODD码[23]、RDP(Row-Diagonal Parity)码[24]、X-Code[25]、Liberation码[26]、Liber8Tion码[27]、STAR码[28]、F-Code[29]、P-Code[30]、T-Code[31]和一般化的X-Code[32]等.

纠删码容错技术在大规模分布式存储中面临的主要挑战已不再是其较高的运算复杂度,而是其较高的网络资源消耗,以及实现上的复杂性.一方面,随着CPU运算能力的飞速发展,现在普通商用服务器的运算能力对于大部分存储系统已经严重过剩.例如,百度云盘的云端存储系统Atlas甚至由于普通处理器利用率过低而采用了ARM处理器[33].另一方面,与RAID的集中式操作不同,分布式系统的分布特性使得纠删码容错的相关操作需要多个节点相互协作,不可避免地带来大量的数据传输,占用较多的网络资源.一直以来,网络都是分布式系统中的稀缺资源,往往是整个系统性能的瓶颈所在.此外,分布式系统的特性也增加了实现的难度.

具体来说,大规模分布式存储系统中纠删码容错技术面临的挑战主要表现在编码实现、数据修复和数据更新等3个方面.

3.1　编码实现

编码实现是指根据给定的纠删码对数据块进行运算、得到校验块并将数据块和相应的校验块分散到不同存储节点上的过程.其主要挑战在于如何在编码实现完全完成之前保证数据的可靠性.一方面,由于存储系统一般以大小固定的块为基本管理单位,数据产生只有累积到一定量才能启动编码实现;另一方面,由于编码实现的复杂性,其编码过程需要花费一定的时间.因此,在将数据块和校验块分散到相应的存储节点之前都有可能发生数据失效,造成数据丢失.此外,编码过程中数据块的读取和校验块的分发也会产生大量的数据传输,占用网络资源.总之,编码实现不仅要考虑数据可靠性还要尽量降低数据传输量.

3.2　数据修复

数据修复的挑战主要在于如何降低数据读取量和数据传输量.与多副本容错技术只需重新拷贝一个副本不同,采用纠删码容错的分布式存储系统进行数据修复时,需要从多个存储节点下载数据并对这些数据进行编解码运算.对于大部分纠删码来说,这些运算是如式(2)所示的线性组合.

$$c_x = \beta_1 c_1 + \beta_2 c_2 + \cdots + \beta_r c_r \tag{2}$$

一个编码块失效,需要使用多个其它编码块进行修复,过程中需要读取并传输大量的数据.这不仅会影响数据修复的速度而且会对数据访问效率产生明显的影响.

3.3　数据更新

数据更新的挑战主要在于如何降低数据读写量和数据传输量.在基于多副本容错的数据更新中,仅需用新的数据依次覆盖各个副本中的原数据即可.在基于纠删码容错的数据更新中,需要先将数据块中校验块中更新部分的原数据读取出来,重新计算相应的校验数据,然后再写入.在纠删码容错中一个数据块关联的校验块较多,需要更新的编码块也更多,这必将产生较大的数据读写量和数据传输量,直接影响到数据更新即覆盖写操作的性能.此外,更新过程中节点也可能失效,造成部分数据已更新而其余部分未更新,从而导致数据不一致.因此,保证数据的一致性也是纠删码容错技术中数据更新的主要挑战之一.

4　编码实现技术研究

在采用纠删码容错的分布存储系统中,需要对新写入的数据进行编码,为了保证数据可用性,通常先利用复制技术保存新数据,等到编码完成后再将多余的副本删除.当前,采用这种方式的分布存储系统包括Facebook公司的HDFS-RAID[34]、微软的云存储系统WAS(Windows Azure Storage)[35]、百度云盘的存储系统Atlas[33]和阿里的TFS(Taobao File System)①等.具体的编码实现方法包括集中式编码实现方法和分布式编码实现方法.

4.1　集中式编码实现方法

目前,实际系统中大多采用集中式编码实现方法.如图1所示,在集中式编码实现方法中,一个条带的所有编码工作都由一个编码节点来单独完成.编码节点从存储节点下载条带中的所有数据块,编码计算校验块,然后再把校验块发送到其它存储节点上.在具体实现中,可以从存储数据块的节点中选择一个作为编码节点,以便减少传输数据量,降低网络资源开销.

① http://tfs.taobao.org/

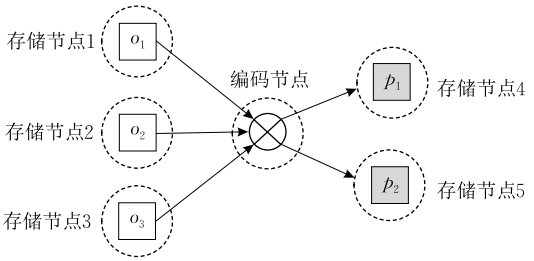


图 1 集中式编码实现方法

HDFS-RAID[34]、Atlas[33]、WAS[35]和 TFS 等分布式存储系统均采用集中式编码实现方法.其中,HDFS-RAID 和 TFS 在编码实现时,编码节点从多个数据节点下载数据块进行编码;Atlas 和 WAS 在编码实现时,将存储在一个数据节点上的大数据块划分成较小的块进行编码,然后把划分的数据块和编码产生的校验块分发到不同的数据节点上.Atlas 和 WAS 在编码之前不需要下载数据,但是其在编码完成后需要发送大量的数据.总体而言,Atlas 和 WAS 的数据读取量和传输量与 HDFS-RAID 和 TFS 相等.但是,Atlas 和 WAS 的数据写入量远高于 HDFS-RAID 和 TFS.以常见的(14,10)-RS 码为例,Atlas 和 WAS 的数据写入量是 HDFS-RAID 和 TFS 的 3.25 倍.

集中式编码实现方法的优点是简单易于实现,其缺点是存在较为严重的性能瓶颈.编码节点负责编码计算和分发编码块,易产生计算瓶颈和网络传输瓶颈,从而影响编码实现效率.

4.2 分布式编码实现方法

为了解决集中式编码实现方法的性能瓶颈问题,Pamies-Juarez 等人[36]针对 RapidRaid 码的特点,提出了基于流水线的分布式编码实现方法.RapidRaid 码是 Pamies-Juarez 等人设计的一种非精确码.该方法将 RapidRaid 码的编码计算分解为可以在不同节点上并行执行的子任务,将编码实现的网络传输负载和计算负载均衡分布到多个节点上,采用流水线方式进行编码计算.如图 2 所示,该编码方法将参与编码实现的节点构成一条流水线,节点 i 接收前一个节点 $i-1$ 发送的数据 $x_{i-1,i}$,将 $x_{i-1,i}$ 和自己所存储的数据块 o_i 进行组合,分别生成自己最终需要存储的编码块 c_i 和供节点 $i+1$ 编码使用的数据 $x_{i,i+1}$,并将 $x_{i,i+1}$ 发送出去.

针对 RapidRaid 码的分布式编码实现方法解决了集中式编码的负载不均衡问题,但是,该方法也存在许多不足之处:(1)该方法只适用于 Pamies-Juarez 等人提出的 RapidRAID 码[36],通用性较差;

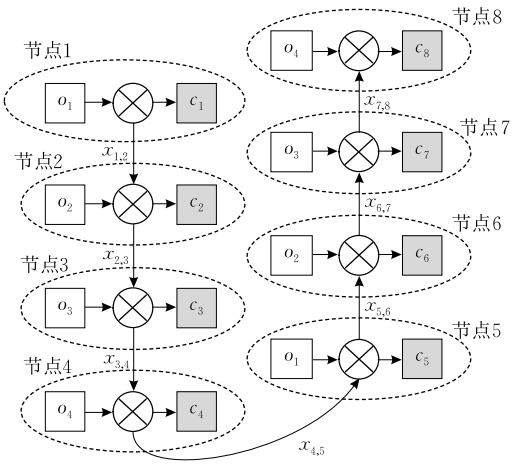


图 2 RapidRAID 码的分布式编码实现方法[36]

(2)RapidRAID 既不是系统码,在大多数情况下也不是 MDS 码,数据读取性能较差且空间利用率不高;(3)与集中式编码实现方法相比,该方法没有减少编码实现过程中的计算量或传输的数据量,并且将读取的数据量提高了 1 倍.

为了减少编码实现过程中的数据传输量,Pamies-Juarez 等人[37]进一步提出了一种新的分布式编码实现方法.该方法采用流水线思想,只有最终存储校验块的节点参与编码实现,且流水线可能需要循环多次;需要特殊的数据块放置策略将多余的数据块均匀放置到最终存储校验块的节点上,以使参与编码实现的节点上有多个数据块.

文献[37]中的方法虽然在一定程度上降低了编码实现过程中传输的数据量,但是它需要根据要产生的校验块数目和数据块的分布情况来构造相应的纠删码.此外,该方法需要参与编码实现的节点上有较多的数据块.否则,流水线需要循环多次才能达到较高的可靠性,从而增加了编码实现过程中的计算量.例如,为了获得(10,6)-MDS 码,当参与编码实现的 4 个节点上有 12 个数据块时,流水线只需循环 1 次;而当参与编码实现的 4 个节点上只有 6 个数据块时,流水线则需要循环 2 次.循环 2 次时,该方法虽然可将网络传输量减少约 22%,但是会增加约 79%的乘法运算和 60%的加法运算.

4.3 总 结

总体来说,相对于集中式编码实现方法,现有分布式编码实现方法可以提升编码实现的性能,但提升效果较为有限.一方面,集中式编码虽然具有一定的性能瓶颈,但是实际上其性能瓶颈并没有那么严重.因为这里的集中仅指一个条带的编码工作,不同条带的编码仍然可以由不同的节点并行地完成,在

并行度较高时，各个节点负载相差得并不多.另一方面，现有分布式编码实现方法一般对纠删码或数据放置等有严格要求，并且数据传输方面的提升往往以更多的数据读取量或计算量为代价.

5　低修复开销纠删码研究

修复开销是由纠删码的特性决定的.因此，设计新型纠删码是从根本上降低修复开销的重要途径.根据不同的编码结构，低修复开销纠删码可以分为分组码和再生码两大类.

5.1　分组码

传统 MDS 码(如 RS 码[17])数据修复开销大的根本原因是其条带内的每个校验块都与所有数据块相关，导致任何一个块失效都需要下载其它 k 个块才能修复.因此，如果将一个条带内的数据块分组，利用组内的数据块产生局部校验块，就可以降低数据修复时需要下载的数据量.以该思想为基础构造的纠删码称为分组码.根据不同的分组方法，分组码可进一步分为层次分组码和交叉分组码两类.

5.1.1　层次分组码

层次分组码的分组呈现明显的层次.首先将一个条带内的数据块分成少数几个较大且互不重叠的组，然后每个组再进一步分成几个若干互不重叠的子组.以此类推，可以根据需要划分多个层次.最后，每组各产生一个只与组内数据块相关的局部校验块，整个条带再产生若干与条带内所有数据块相关的全局校验块.这样大部分数据失效就可以在组内进行修复，从而降低成本.最基本的层次分组码是两层分组码.图 3 显示了微软云存储系统 WAS[35] 中采用的 LRC 码[20] 的一个实例 LRC(8,2,2).LRC(8,2,2)的一个条带包含 8 个数据块，平均分成两个组，每组各产生 1 个局部校验块，整个条带再产生 2 个全局校验块.可见，当组内只有 1 个块失效时，只需使用组内的其它 4 个块即可进行修复.只有在一个组内有多个块失效时，才需要利用组外的块来修复.考虑到超过 98% 的失效情况下，条带内都只有一个块失效[38]，LRC 可以有效降低数据修复的成本.

在组内有多个块失效时，两层分组码仍然需要在全局范围内进行修复.为此，可以对数据块进行更多层次的分组，高层的组包含若干低层较小的组.Pyramid 码[39] 就是根据该思想构造的多层次分组码.随着分组层次的增多，多层分组码的平均修复开销会降低，但其存储空间开销会增大.因此，多层分组码可以在修复开销与存储空间开销之间灵活地进行权衡.

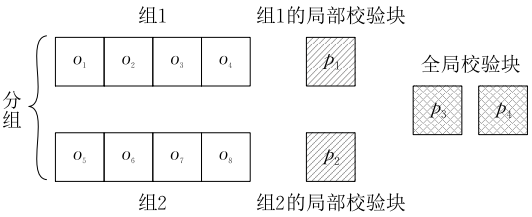


图 3　LRC(8,2,2)的组织结构

除了 LRC 码[20] 和 Pyramid 码[39]，层次分组码还有 Facebook 采用的 LRCs 码[11] 和 EXPyramid 码[40].与 LRC 码相比，LRCs 码的主要不同之处是其对所有全局校验块再产生一个局部校验块，降低了全局校验块的修复开销.EXPyramid 码将两层 Pyramid 码的数据块改为阵列结构，在横向和纵向都产生局部校验块，进一步降低了数据修复的成本.

5.1.2　交叉分组码

交叉分组码与层次分组码的不同之处有两点：(1)交叉分组码中各组包含的数据块数目基本相同，且各组包含的数据块有一部分相同，而层次分组码不同层级的分组大小差别很大，每个高层的分组会包含若干较低层次的组；(2)交叉分组码一般不存在全局校验块.

图 4 是 Miyamae 等人[41] 提出的交叉分组码 SHEC(Shingled Erasure Code)的一个实例 SHEC(10,6,5).SHEC 的基本思想是使各个分组像屋顶的瓦片一样相互重叠，从而在多个块失效时可以利用较少的组进行局部修复.图 4 中的 SHEC(10,6,5)将 10 个数据块分为相互重叠的 6 个组，每组包含 5 个数据块，除了第 3 组和第 4 组，其它任意两组之间重叠 3 个数据块，每组的校验块均通过组内的 5 个数据块运算产生.具体的编码过程如式(3)所示：

$$\begin{cases} p_1=\alpha_{1,1}o_1+\alpha_{1,2}o_2+\alpha_{1,3}o_3+\alpha_{1,4}o_4+\alpha_{1,5}o_5 \\ p_2=\alpha_{2,1}o_3+\alpha_{2,2}o_4+\alpha_{2,3}o_5+\alpha_{2,4}o_6+\alpha_{2,5}o_7 \\ p_3=\alpha_{3,1}o_5+\alpha_{3,2}o_6+\alpha_{3,3}o_7+\alpha_{3,4}o_8+\alpha_{3,5}o_9 \\ p_4=\alpha_{4,1}o_6+\alpha_{4,2}o_7+\alpha_{4,3}o_8+\alpha_{4,4}o_9+\alpha_{4,5}o_{10} \\ p_5=\alpha_{5,1}o_8+\alpha_{5,2}o_9+\alpha_{5,3}o_{10}+\alpha_{5,4}o_1+\alpha_{5,5}o_2 \\ p_6=\alpha_{6,1}o_{10}+\alpha_{6,2}o_1+\alpha_{6,3}o_2+\alpha_{6,4}o_3+\alpha_{6,5}o_4 \end{cases} \tag{3}$$

在此实例中，单个块的失效修复只需下载 5 个块；两个块的失效修复只需下载 6 或 7 个块.例如，当图 2 中的 o_6 和 o_9 失效时，根据式(3)中的第 3 和第 4 个等式，使用 o_5，o_7，o_8，o_{10}，p_3 和 p_4 这 6 个块即可修复.可见，SHEC 避免了下载数据量随着失效块数增加而急剧增长的情况.

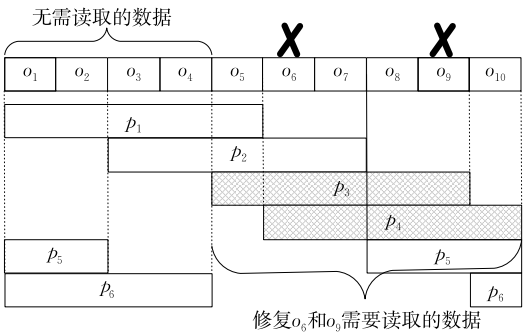


图 4 SHEC(10,6,5)的构造和修复示意图[41]

交叉分组码还有 WEAVER 码[42]、GRID 码[43]和 Hover 码[44]等. WEAVER 码先限定所有数据块的出度(一个数据块关联的校验块个数)和校验块的入度(一个校验块关联的数据块个数),然后通过搜索的方法找出最优的组合和放置方法,使容错性达到最高. WEAVER 码在固定数据修复开销前提下可构造出容错能力高达 12 的实例. 但是 WEAVER 码的冗余度较高,空间利用率始终不超高 50%. 如果想要同时维持较低的修复开销和较高的容错能力,甚至需要 3 倍或 4 倍的存储空间开销. GRID 码和 Hover 码都是纵横相结合的阵列码,可以达到较高的容错能力,同时也具有较高的存储空间利用率.

5.2 再生码

再生码(Regenerating Codes)[45-46]是一种基于网络编码思想[47]设计的纠删码,由 Dimakis 等人[48]在 2007 年首次提出. 其基本思想是通过适当增加冗余并且使新生节点从尽量多的节点下载数据来降低修复需要下载的总数据量. 再生码具有两个明显的特点:(1)再生码的数据块和校验块都包含相同数量的子块,编码与修复时以子块为基本单位,子块之间的关系也更为复杂;(2)再生码在进行数据修复时,新生节点需要从尽量多的节点来下载数据.

再生码一般用三元组$[n,k,d]$来表示. $[n,k,d]$-再生码的一个条带包含 n 个编码块,可以容忍任意 $n-k$ 个块失效,进行数据修复时新生节点可以连接 d 个存活节点下载数据,其中 $k\leq d\leq n-1$. 另外,再生码还有 3 个常用的辅助参数 α,β 和 B,分别表示单个编码块包含的子块个数,连接到 d 个节点进行数据修复时从单个节点下载的子块个数和一个条带包含的所有数据子块个数.

图 5 显示了一个简单的$[4,2,3]$-再生码及其数据修复过程. 在该再生码中,原数据对象被分割为 4 个大小相等的数据子块 $o_{1,1}$,$o_{1,2}$,$o_{2,1}$ 和 $o_{2,2}$,每两个数据子块合成一个数据块,一个校验块也包含两个校验子块. 校验子块完全通过异或运算产生,校验子块和数据子块之间的关系如图 5 所示. 当第 2 个校验块失效时,先将子块 $o_{2,1}$ 和 $o_{2,2}$ 在其节点内相异或得到 $o_{2,1}+o_{2,2}$,然后 $o_{2,1}+o_{2,2}$ 与 $o_{1,2}+o_{2,2}$ 相异或恢复出失效的 $o_{1,2}+o_{2,1}$,$o_{1,1}$ 与 $o_{1,2}+o_{2,2}$ 相异或恢复出失效的 $o_{1,1}+o_{1,2}+o_{2,2}$. 在一般的(4,2)-纠删码中,新生节点需要从 2 个节点下载相当于 4 个子块的数据. 而在图 3 的再生码中,新生节点虽然连接了 3 个节点,却只需下载 $o_{1,1}$,$o_{2,1}+o_{2,2}$ 和 $o_{1,2}+o_{2,2}$ 这 3 个子块,将下载量减少了 25%. 实际上,Dimakis 等人[48]证明,如果新生节点能够从 $n-1$ 个节点下载数据,再生码可以将总体数据下载量降低高达 84%.

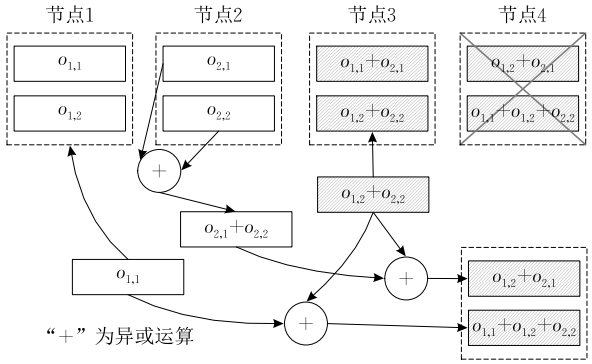


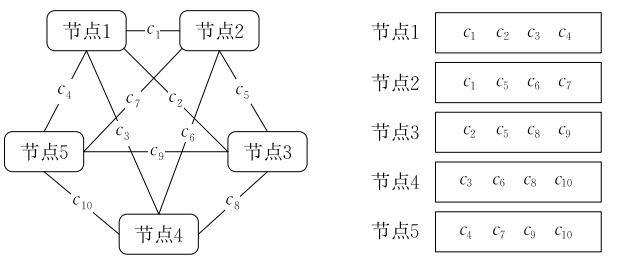
图 5 一个[4,2,3]-再生码及其数据修复过程

再生码的研究主要关注 MBR 码(Minimum Bandwidth Regenerating Codes)和 MSR 码(Minimum Storage Regenerating Codes). MBR 码具有最低的数据修复带宽,MSR 码具有最低的存储开销. 再生码同样分为精确码和非精确码,本文只关注精确再生码. Dimakis 等人[48]提出了再生码的概念并证明了再生码修复带宽的下界,但是没有证明达到这个下界的再生码是否存在,也没有给出构造这种再生码的具体方法. 2009 年,$[n,2,n-1]$的精确 MSR 码被证明是存在的[49]. 同年,$[n,k,n-1]$的精确 MBR 码被构造出来[50]. 2010 年,$[n=d+1,k,d\geq 2k-1]$的精确 MSR 码被构造出来. 2011 年,Rashmi 等人[21]利用矩阵乘的方法构造出了$[n,k,d]$的精确 MBR 码和$[n,k,d\geq 2k-2]$的精确 MSR 码,并证明不存在 $d<2k-2$ 的精确 MSR 码. 至此,所有存在的 MBR 码和 MSR 码都可以用统一的方法被构造出来.

再生码可以极大地减少修复时的传输数据量,但是需要读取的数据量却更大. 在数据修复过程中,参与修复的节点需要把自己存储的所有数据都读取出来进行组合. 由于 MBR 码需要存储的数据量更

大，所以修复时需要读取的数据量比传统纠删码多，这不仅增加了系统的磁盘负载，也限制了修复的速率.虽然 MSR 码存储的数据量和传统 MDS 码相等，但是修复时需要从多于 k 个节点下载数据，所以其读取的数据量也比较多.

针对上述问题，研究者设计了 RBT(Repair-by-Transfer)MBR 码[51-52]. RBT MBR 码在数据修复时只传输数据而不进行任何数学运算，使需要读取的数据量和需要传输的数据量相同.该类再生码最先由 Shah 等人[51]提出.其构造方法是：先用任何一种$(n(n-1)/2,B)$-MDS 码对条带中的 B 个数据子块进行编码，产生 $n(n-1)/2$ 个编码子块.然后，将产生的 $n(n-1)/2$ 个编码子块分别对应于一个包含 n 个顶点的全连接图的 $n(n-1)/2$ 条边，该图的每个顶点分别对应一个存储节点，如图 6(a)所示，每个节点存储与其对应的顶点相连的所有边上的编码子块，如图 6(b)所示.从图 6(b)可以看出，当任何一个节点失效时，只需从其余节点分别读取并传输一个编码子块就可以恢复出失效数据，不需要进行任何数学计算.

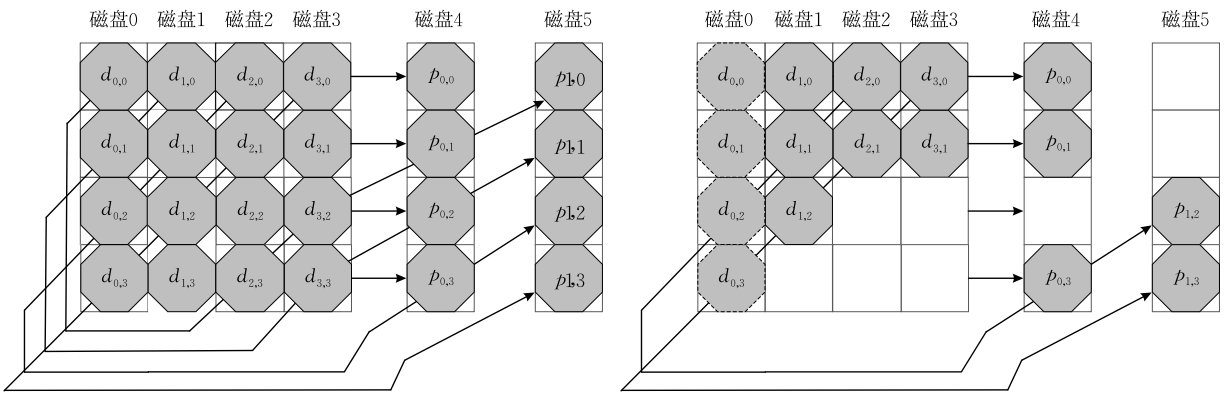


(a) 构造关系　　(b) 数据分布情况

图 6　[5,3,4]-RBT MBR 码的构造图[51]

针对上述构造方法存在计算复杂度较高的缺点，Lin 等人[52]提出了一种新的构造方法.此构造方法也采用 Rashmi 等人[21]提出的矩阵乘框架，可将编码运算需要的伽罗瓦域大小从 $n(n-1)/2$ 减小为 n，将编码运算的复杂度从 $O(n^4)$ 降低到 $O(n^3)$.近来，Rashmi 等人将 RBT MBR 码的思想引入到 MSR 码中，提出了一种近似 Repair-by-Transfer 的 MSR 码，称为 PM-RBT(Product-Matrix RBT)码[53].由于 MSR 码的冗余度较低，PM-MSR 码只能减少部分数据修复时需要读取的数据量.

此外，大部分阵列码在结构上也具有再生码的特点，可以通过从尽量多的节点下载数据来降低数据修复开销.此发现最早由 Xiang 等人[54]提出，用于优化 RDP 码[24]修复时下载的数据量.图 7(a)显示了包含 4 个数据磁盘的 RDP 码[24]的编码实例，其中磁盘 0、磁盘 1、磁盘 2 和磁盘 3 为数据磁盘，磁盘 4 和磁盘 5 为校验磁盘.现假设磁盘 0 失效.传统的修复方法是使用磁盘 1、磁盘 2、磁盘 3 和磁盘 4 上的数据进行修复，即分别使用 $d_{1,i}$，$d_{2,i}$，$d_{3,i}$和 $p_{0,i}$ 修复 $d_{0,i}$，$i=0,1,\cdots,3$.这需要访问 4 个磁盘并下载 16 个子块.Xiang 等人指出，访问 5 个磁盘可以减少总下载数据量.如图 7(b)所示，可以先用 $d_{1,0}$，$d_{2,0}$，$d_{3,0}$，$p_{0,0}$ 和 $d_{1,1}$，$d_{2,1}$，$d_{3,1}$，$p_{0,1}$ 分别修复出 $d_{0,0}$ 和 $d_{0,1}$，然后使用 $d_{2,0}$，$d_{1,1}$，$p_{0,3}$，$p_{1,2}$ 和 $d_{3,0}$，$d_{2,1}$，$d_{1,2}$，$p_{1,3}$ 分别修复出 $d_{0,2}$ 和 $d_{0,3}$.虽然访问了 5 个磁盘，但是只需下载 $d_{1,0}$，$d_{2,0}$，$d_{3,0}$，$p_{0,0}$，$d_{1,1}$，$d_{2,1}$，$d_{3,1}$，$p_{0,1}$，$p_{0,3}$，$d_{1,2}$ 和 $p_{1,3}$ 等共计 12 个块即可完成修复，将总数据下载量减少了 25%.



(a) 编码结构　　(b) 数据修复

图 7　包含 4 个数据磁盘的 RDP 码的编码结构与数据修复示意图

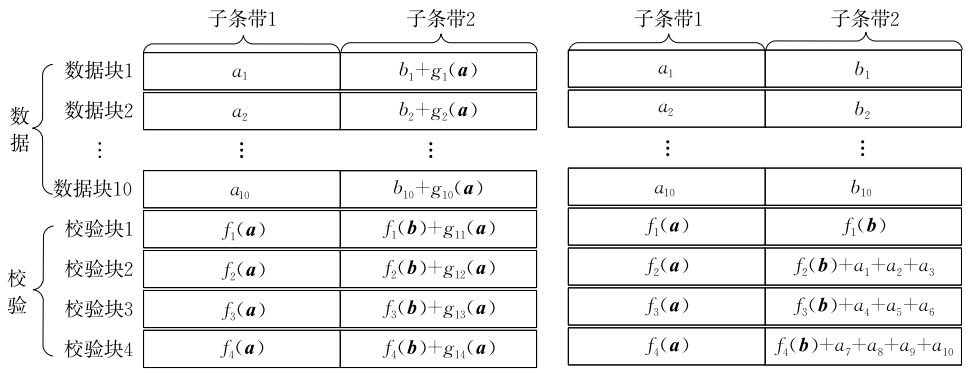
Wang 等人[55]和 Xu 等人[56]分别将 Xiang 等人[54]的思想引入到 EVENODD 码[23]和 X-Code 码[25]的修复中，通过同时使用多个校验磁盘来降低这些纠删码的数据修复开销.Khan 等人[38]进一步将此问题一般化，使其适用于任何基于异或运算的纠删码.基于异或运算纠删码的每个块均有多个大

小相等的子块构成，校验子块通过某些数据子块相异或而产生. 由于数据子块与校验子块之间的关系较为复杂，在一个块失效时，其中的每个子块都可能有多种修复方法. Khan 等人[38]提出了一种通用的搜方法，来查找出使总下载数据量最小的修复方法. 基于此思想，Khan 等人[38]还提出了容错能力分别为 2 和 3 的 Rotated RS 码. Khan 等人[38]用他们提出的搜索算法分析了多种基于异或运算纠删码的性能. 结果表明，对于常见的基于异或运算纠删码，上述方法可将总数据下载量减少 20%左右.

然而，Khan 等人[38]提出的搜索算法复杂度极高，为了选出最好的组合不得不穷尽所有的选择. 为了解决这一问题，Khan 等人[57]和 Zhu 等人[58]分别提出了近似最优的搜索方法.

基于异或运算的纠删码和 Rotated RS 码的容错能力都比较有限，上述思想对这些纠删码的优化效果也较为有限. 近来，Rashmi 等人[12,59-60]提出了一种称为 Hitchhiker 的 MDS 码. Hitchhiker 码可基于任何 MDS 码进行构造. Hitchhiker 码的一个条带由两个普通 MDS 码条带构成. 其基本编码思想是：每个子条带分别先用普通的 MDS 码进行编码，然后对其中一个子条带中的数据计算一些检验子块，并把这些校验子块与另一个子条带中的子块相合并，如图 8(a)所示. 图 8(a)中的 f_1 和 g_1 等表示计算校验的函数，其中 f_1，f_2，f_3 和 f_4 表示采用的普通 MDS 计算校验的函数. 例如，$f_1(\boldsymbol{a})$ 表示子条带 1 的第一个校验子块.

容易证明，不论 $g_1, g_2, \cdots, g_{14}$ 是什么样的函数，Hitchhiker 码都具有 MDS 性质. 但若要降低数据修复开销，则需要对这些校验函数进行特殊的设计. Rashmi 等人[59]给出了 3 种设计. 图 8(b)显示了其中一种，称为 Hitchhiker-XOR 码. Hitchhiker-XOR 码的额外校验数据全部通过异或运算产生. 现假设数据块 1 失效，即子块 a_1 和 b_1 失效，数据修复分为 3 步. 第 1 步，使用 $b_2, b_3, \cdots, b_{10}$ 和 $f_1(\boldsymbol{b})$ 等 10 个子块修复出子块 b_1；第 2 步，使用 b_1 和 $b_2, \cdots, b_{10}$ 等 10 个块计算出 $f_2(\boldsymbol{b})$；第 3 步，使用 $f_2(\boldsymbol{b})$，$f_2(\boldsymbol{b})+a_1+a_2+a_3$，$a_2$ 和 a_3 等 10 个子块修复出子块 a_1. 整个过程中总共需要读取 $b_2, b_3, \cdots, b_{10}, f_2(\boldsymbol{b})+a_1+a_2+a_3, a_2$ 和 a_3 等 13 个子块. 普通的 MDS 码则需要下载相当于 20 个子块的数据. Hitchhiker-XOR 码将数据下载量降低了 35%.



(a) 基本编码框架 (b) 一个编码实例

图 8 Hitchhiker 码的基本编码框架和编码实例[59]($n=14, k=10$)

相比于基于异或运算的纠删码和 Rotated RS 码，Hitchhiker 码能够提供任意的容错能力，并且其可将数据修复所需下载的数据量降低高达 45%.

5.3 总 结

5.3.1 数据编码性能对比

影响纠删码数据编码性能的因素有编码算法的时间复杂度以及编码过程中需要读取、传输和写入的数据量.

纠删码的编码运算主要是有限域上的加法和乘法运算，其中较为费时的是乘法运算. 所以，乘法运算的数量可以用来表征编码算法的复杂度. 此外，编码使用的有限域的大小也对运算时间有重大影响. 随着有限域的增大，乘法运算的复杂度成指数级增长. 此外，对于较小的有限域，如 8 位 256 个元素的有限域，可以将所有可能的乘法运算结果保存在内存中，用查表的方法加快乘法运算速度. 目前对于常见的参数，上述各类纠删码中较优秀者的编码运算基本可以在 8 位有限域上完成.

在现代分布式存储系统中，真正决定纠删码编码性能的是需要读取、传输和写入的数据量. 随着计算机运算能力的飞速增长，编码运算的速度已远远超过数据的读取、传输和写入速度. 影响数据读取、

传输和写入量的主要因素是编码前数据的分布情况和采用的编码实现方法.此外,纠删码的数据冗余度也对运算量、数据传输量和写入量有重大影响.冗余度越高,意味着有更多的校验数据需要产生、发送出去并写入到磁盘中.

表 1 从存储空间利用率以及编码单位数据平均需要的乘法运算量、平均数据读取量、平均数据传输量和平均数据写入量等方面,对比了几种典型纠删码在集中式编码实现方法下的编码性能.除了其中的(9,5,8)-MSR 码和(14,10,13)-MBR 码两个再生码以外,其它纠删码均已被应用在了实际的大型分布式存储系统中.表 1 中,SHEC(10,6,5)的容错能力为 3,其它纠删码的容错能力均为 4.

表 1　几种典型纠删码的数据编码性能对比

纠删码	空间利用率/%	平均乘法运算量	平均数据读取量	平均数据传输量	平均数据写入量	类别
RS(14,10)	71.40	4.0	1	1.30	0.40	传统 MDS 码
LRCs(10,2,4)	62.50	4.0	1	1.50	0.60	分组码
SHEC(10,6,5)	62.50	3.0	1	1.50	0.60	
(9,5,8)-MSR	55.60	6.4	1	1.60	0.80	再生码
(14,10,13)-MBR	46.70	14.0	1	2.04	1.14	
(14,10)-Hitchhiker-XOR	71.41	4.0	1	1.30	0.40	

从表 1 可以看出,在运算复杂性方面,RS 码与 LRCs 码和 Hitchhiker 码这些以 RS 码为基础的纠删码基本相同.交叉分组码 SHEC(10,6,5)完全的运算复杂度较低,但其容错能力也较低.(9,5,8)-MSR 码和(14,10,13)-MBR 码具有最高的运算复杂度.除了较为复杂外,再生码较高的运算复杂度也与其更高的数据冗余度有重大关系.在集中式编码实现方法下,除了数据读取量,数据传输量和写入量完全取决于纠删码的冗余度.因此,冗余度最低的 RS(14,10)码和(14,10)-Hitchhiker-XOR 码表现最好,冗余度最高的(14,10,13)-MBR 码表现最差.

5.3.2　数据修复性能对比

通常情况下,采用修复一个失效块平均所需下载的数据量来衡量纠删码的数据修复开销.

将修复一个块平均所需下载的数据量与块大小之比称为单块修复代价,以 R_s 表示.单块修复代价在一定程度上反映了纠删码数据修复的开销,但无法反映整个系统数据修复的总体开销.不同冗余度的纠删码具有不同的存储开销.在原始数据量相同的情况下,系统采用不同开销的纠删码,其实际存储的数据量是不同的,从而导致数据修复的总体开销不同.

假设系统存储的原始数据量为 V,数据失效速率为 λ,纠删码的存储空间利用率为 η,则可计算出系统单位时间内的总体修复带宽 F 为

$$F=\frac{V}{\eta}\cdot\lambda\cdot R_s=\frac{R_s}{\eta}\cdot V\cdot\lambda \tag{4}$$

从式(4)可以看出,在原始数据量和数据失效速率相同的情况下,纠删码的总体修复带宽正比于其单块修复代价 R_s,反比于其存储空间利用率 η.因此,可以用 R_s 与 η 之比来表示纠删码的总体修复代价.以 R_c 表示纠删码的总体修复代价,则有

$$R_c=\frac{R_s}{\eta} \tag{5}$$

表 2 从存储空间利用率、单块修复代价和总体修复代价等方面对比了表 1 中几种典型纠删码的数据修复性能.作为对比,表 2 中加入了常见的三副本技术.

表 2　几种典型纠删码与多副本技术的数据修复性能对比

纠删码	空间利用率/%	单块修复代价	总体修复代价	类别
RS(14,10)	71.40	10.00	14.00	传统 MDS 码
LRCs(10,2,4)	62.50	3.75	6.00	分组码
SHEC(10,6,5)	62.50	5.00	8.00	
(9,5,8)-MSR	55.60	2.00	3.60	再生码
(14,10,13)-MBR	46.70	1.00	2.14	
(14,10)-Hitchhiker-XOR	71.41	7.64	10.70	
三副本	33.30	1.00	3.00	多副本

从表 2 可以看出,传统 MDS 码的存储空间利用率最高,但是其数据修复开销也最大,甚至高于其它种类纠删码数倍.相比于传统 MDS 码,分组码能够以较少的额外存储空间开销为代价,显著降低数据修复的成本.分组码也较容易实现,这也是其在大型存储系统中得到应用的重要原因之一.再生码可以极为有效地降低数据修复开销,但是再生码的存储空间利用率明显低于其它类别纠删码,其存储空间利用率最高也只能达到 50%左右.所以,再生码不适合于对存储成本要求较高的大规模存储系统,而适合于对带宽成本极其敏感的系统.例如,可以将再生码用在数据中心级的数据容错中,因为数据中心之间的网络带宽极其昂贵.

5.3.3 总 结

分布式存储系统对纠删码的要求主要在容错能力、存储空间开销和数据修复开销 3 个方面.但这 3 个方面是相互排斥的,提升其中一个方面必然会牺牲其它方面.分布式存储系统中,纠删码的设计本质上都是在容错能力、存储空间开销和数据修复开销之间进行权衡.由于容错能力是分布式容错系统的基本要求,现有研究基本都是在保持容错能力的前提下,在存储空间开销和数据修复开销之间进行权衡.分组码就是此思路的典型代表.相比于分组码,再生码在理论上取得了重大突破,证明并达到了一定存储空间开销下修复时数据下载量的下界.不过早期再生码在数据下载量方面的减少是以增加数据读取量为代价的.RBT MBR 码的提出,使 MBR 码在数据下载量和读取量都达到了最优.PM-RBT 码的提出,缓解了 MSR 码在数据下载量方面的问题.

6 数据修复技术研究

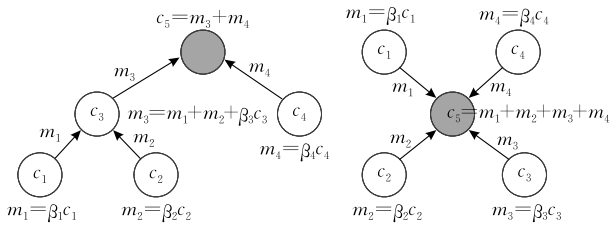
除了从纠删码本身入手降低数据修复的代价之外,从数据修复的具体过程入手,通过优化修复时的数据读取和数据传输过程也可以进一步提高数据修复的效率.

传统的数据修复方法通常采用星型的数据传输方式,所有提供节点直接将数据发送给新生节点,所有参与修复的节点构成一个以新生节点为中心的星型结构.星型数据修复方法简单直观,但是其存在严重的性能瓶颈.

现有的数据修复技术大部分都基于树型数据修复方法.系统会先构建覆盖所有参与修复的节点且以新生节点为根的修复树.在修复过程中,叶节点先将自己的数据乘以相应的系数后将其向上传输给自己的父节点,内部节点收取其所有子节点发送的数据并将这些数据和自己的数据进行一定的组合,然后再将组合结果传输给自己的父节点,以此类推,直至最终到达修复树的根节点.根节点将收到的所有数据进行组合后就可恢复出失效数据.图 9(a)显示了树型数据修复方法的数据传输结构.其中,失效编码块 c_5 可用编码块 c_1,c_2,c_3 和 c_4 来修复,其关系如式(6)所示.作为对比,图 9(b)显示了传统星型数据修复方法中的数据传输结构.

$$c_5=\beta_1 c_1+\beta_2 c_2+\cdots+\beta_4 c_4 \tag{6}$$

根据修复树构造方法的不同,现有数据修复技术可以分为两大类:一是带宽感知的数据修复技术,依据网络带宽来构建修复树;二是拓扑感知的数据修复技术,依据网络拓扑来构建修复树.



(a) 树型数据修复方法 (b) 星型数据修复方法

图 9 树型修复方法和星型修复方法的数据传输结构

6.1 带宽感知的数据修复技术

带宽感知的数据修复技术最早由 Li 等人[61]提出.其主要考虑到大规模分布式系统往往是异构的,即节点的性能不尽相同,节点之间的网络带宽也不同.此类方法试图根据网络带宽来构造修复树,通过尽量利用网络中的高可用带宽达到提高数据传输速度、缩短修复时间的目的.Li 等人证明,采用以节点间可用网络带宽作为边权重的最大生成树作为修复树,可使数据修复时间达到最小.

图 10 显示了传统星型数据修复方法和文献[61]中带宽感知的树型数据修复方法的对比情况.其中,原数据对象大小为 M,分为 3 个数据块,通过编码产生个 1 校验块,3 个数据块和 1 个校验块分别存储在节点 2、节点 3、节点 4 和节点 5 等 4 个节点上,各块大小均为 $M/3$.假设节点 4 上的块失效,用节点 2、节点 3 和节点 5 上的 3 个可用块进行修复,节点 1 为新生节点.

图 10(a)显示了各节点之间的可用网络带宽.图 10(b)显示了采用星型数据修复方法时的数据传输情况.在此方法中,所有 3 个提供节点(节点 2、节点 3 和节点 5)都直接向节点 1 传输数据.星型修复方法的修复时间取决于可用节点与新生节点之间的瓶颈带宽,即节点 1 和节点 3 之间的带宽.每个可用节点与新生节点之间传输的数据量均为 $M/3$.可以计算出,传统星型数据修复方法的修复时间为 $(M/3)/20=M/60$.

图 10(c)显示了带宽感知的树型数据修复方法的数据传输过程,图 10(d)显示了对应的修复树.根据 Li 等人的证明,该方法的修复时间取决于修复树中的最小带宽.此实例中的最小带宽为 40.该方法中,节点之间传输的数据量也为 $M/3$,可以算出其修复时间为 $(M/3)/40=M/120$.可见,相比于星型数据修复方法,带宽感知的数据修复方法将修复时间缩短了一半.

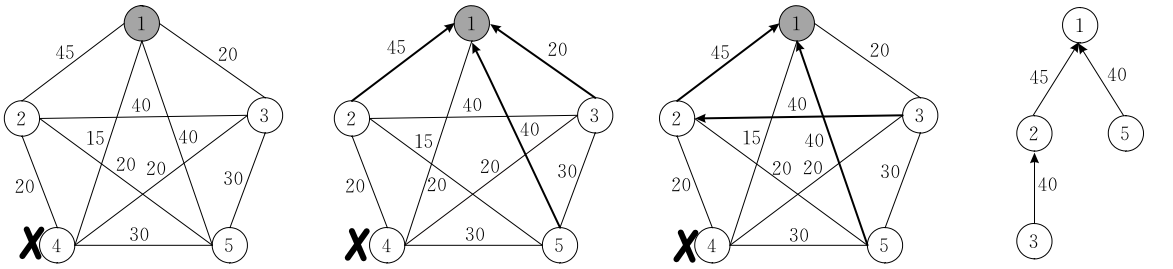


图 10　星型修复方法和带宽感知的树型数据修复方法的对比

Li 等人[62]又将充分利用可用带宽的思想引入到再生码中，提出了针对再生码的树型数据修复方法 RCTREE. 为了更加有效地利用系统中的可用带宽，加速多节点同时失效情况下的数据修复，Sun 等人[63]扩展了文献[61]中的数据修复方法，提出了一种带宽感知的并行数据修复方法 TPR(Tree-structured Parallel Regeneration). 当多个节点同时失效时，TPR 方法会以各新生节点为根，分别构建多个修复树，并行地对失效节点进行修复.

6.2　拓扑感知的数据修复技术

拓扑感知的数据修复技术的基本思想是通过构造与物理拓扑相符的修复树，以减少数据修复时在网络拓扑的高层链路上传输的数据量.

目前，最常见的网络拓扑仍然为多层的树型结构[64]，由下到上依次为由机架交换机(Top-of-Rack，TOR)组成的边界层(Edge Layer)，由聚合交换机组成的数据聚合层(Aggregation Layer)和由核心交换机和路由器组成的核心层(Core Layer). 树形网络的突出问题是高层的带宽往往非常紧张，目前部署的网络中边界层的总带宽仍然为核心层的 4～10 倍[64-65]. 近来有关数据中心网络负载的研究[8,64,66]均表明，核心层链路的利用率是最高的. 因此，如果能够有效减少核心层的带宽消耗，将极大提高系统整体的性能.

针对此问题，Zeng 等人[67]和 Zhang 等人[68]提出了拓扑感知的数据修复技术，以降低数据修复占用的核心网络带宽. 这种数据修复技术的基本思想是，将距离较近的编码块(如处于同一个机柜中的编码块)先就近组合然后再发送到更远的节点进行进一步的组合，直至最终汇入新生节点，这样就可以逐步减少在网络拓扑高层中传输的数据量，降低核心带宽消耗，从而提高数据修复效率，并降低数据修复对整个系统性能造成的不良影响.

图 11 显示了该修复方法的一个实例，其中存储校验块 $o_1+o_2+o_3+o_4$ 的节点失效. 修复过程中，交换机 S1 先将数据块 o_1，o_2 和 o_4 进行组合，产生 $o_1+o_2+o_4$ 并将其发送给交换机 S2. 交换机 S2 再将 $o_1+o_2+o_4$ 和 o_3 组合从而恢复出失效的校验块 $o_1+o_2+o_3+o_4$. 整个过程中，交换机 S1 和 S2 之间只传输了 1 个块. 在传统的星型修复方法中，交换机 S1 和 S2 之间则需要传输 3 个块. 可见，拓扑感知的树型数据修复方法能够有效降低网络拓扑中高层的数据传输量.

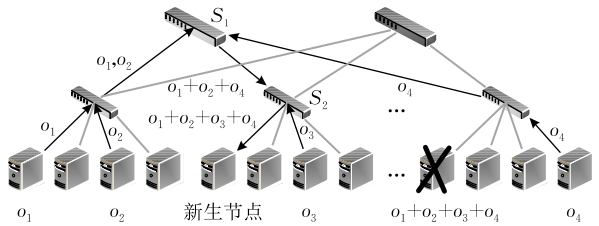


图 11　拓扑感知的数据修复方法实例[67]

6.3　总　结

带宽感知的数据修复技术虽然在理论上非常吸引人，但是存在难以克服的缺点：(1) 分布式系统中节点间的带宽是实时动态变化的，测试成本高且难以获得精确的结果；(2) 该类技术只是将数据传输导向到较快的链路，并没有降低数据修复的负载，所以不能有效提升总体的数据修复效率. 此外，很多研究工作涉及的网络模型也与实际网络不符.

相对而言，拓扑感知的数据修复技术更加具有可操作性. 但是，目前这类方法也存在比较明显的缺点. 需要由交换机完成修复过程中的数据合并，交换机需要支持数据运算，也需要设计专门的底层通讯协议. 这些问题限制了拓扑感知的数据修复技术在实际系统中的应用.

7　数据更新技术研究

纠删码中一个数据块关联着较多的校验块，导

致数据更新需要同时更新较多的块，不仅需要大量的数据传输和写入，也使保持数据的一致性面临挑战.

依据更新方式，可将现有纠删码容错技术中的数据更新方法分为 3 种：替换式更新方法、追加式更新方法和混合式更新方法.

7.1 替换式更新方法

替换式更新方法[69-70]直接用新的数据覆盖原有数据，完成对数据块和校验块的更新后才返回更新成功. 图 12 显示了一个替换式更新方法的实例. 其中，节点 1 和节点 2 存储数据块，节点 3 存储校验块，a，b，c 和 d 分别表示 4 个原始数据块，$a+c$ 和 $b+d$ 分别表示 2 个校验块，a'，b'和 c'分别表示对原数据块 a，b 和 c 中的一部分进行更新的新数据子块. 可以看出，替换式更新方法直接更新了所有的数据块和校验块.

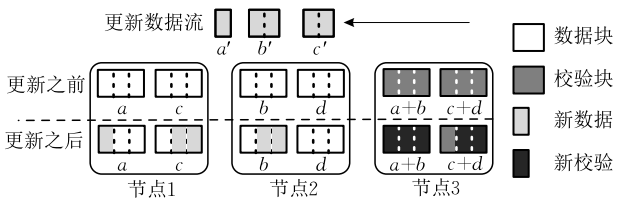


图 12 替换式更新方法的数据更新实例

替换式更新方法能够保证数据块与校验块实时更新，对提高系统的性能有重要作用. 数据块的实时更新能够保证应用访问到的数据始终是最新的，有利于提高应用运行的效率和准确性. 校验块的实时更新能够保证访问的校验数据始终是最新的，有利于提高数据修复的性能. 但是，替换式更新方法存在数据读写量大和数据传输量大等不足，对数据更新效率产生显著影响.

7.2 追加式更新方法

追加式更新方法先将新数据以日志的形式追加在原始数据块之后，将新数据与原数据的差值以日志的形式追加在校验块之后；一段时间之后或需要访问新的数据块或校验块时再将追加的数据与原数据合并. 图 13 显示了一个追加式更新方法的实例. 更新过程中，先读取被更新的数据，并计算新数据 a'，b'和 c'与被更新部分数据的差值 Δa，Δb 和 Δc，然后将新数据 a'和 c'追加到节点 1 上，将新数据 b'追加到节点 2 上，将新数据 a'，b'和 c'与被更新部分数据的差值 Δa，Δb 和 Δc 追加到校验节点上. GFS[5]和 Azure[35]采用的是追加式更新方法.

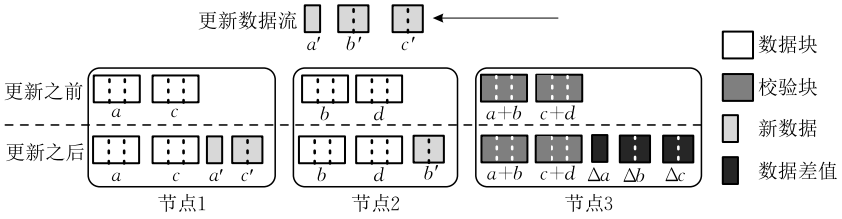


图 13 追加式更新方法的数据更新实例

相比于替换式更新方法，追加式更新方法避免了数据更新过程中对原校验数据的读取，追加式的写入也将随机写转化为了顺序写，这些都有利于提高数据更新的效率. 但是，追加式更新方法存在存储空间消耗较多和数据访问性能较差等不足：(1) 追加式更新方法需要同时保存原数据和新数据与原数据的差值，这必然会消耗更多的存储空间. 当数据更新频繁发生时，该问题会更加严重；(2) 经过更新后，原来数据块中连续的数据将随机地分布在日志中，导致无法顺序读取数据，明显影响了数据访问的效率. 此外，访问校验块需要先合并数据，这影响了数据修复的性能[71-73].

7.3 混合式更新方法

混合式更新方法[74-75]是替换式更新方法和追加式更新方法的结合. 具体来讲，混合式更新方法采用替换式更新方法对数据块进行实时更新，同时采用追加式更新方法对校验块进行延迟更新. 图 14 显示了一个混合式更新方法的实例. 更新过程中，先读取被更新的数据，并分别计算新数据 a'，b'和 c'与被更新部分数据的差值 Δa，Δb 和 Δc，然后用新数据 a'，b'和 c'分别覆盖被更新的部分数据，并将新数据 a'，b'和 c'与被更新部分数据的差值 Δa，Δb 和 Δc 追加到校验节点上.

相比于追加式更新方法，混合式更新方法对数据块的实时更新使得对数据块的读取可以顺序进行，提高了数据读取效率，也一定程度上降低了存储空间的消耗. 但是，混合式更新方法对数据块的写操作是随机的，影响了数据更新的效率. 在不考虑数据合并的情况下，混合式更新方法的数据读写量和传输量与追加式更新方法完全相等. 此外，由于在访问校验块时仍然需要先合并数据，混合式更新方法也同样存在数据修复性能差等不足.

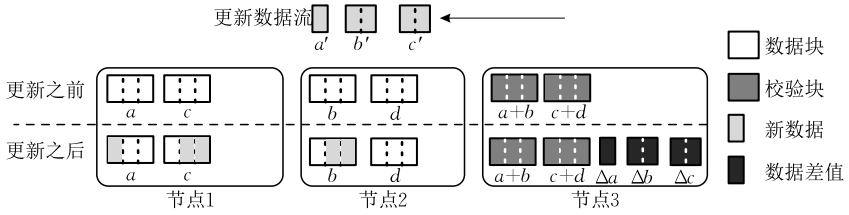


图 14　混合式更新方法的更新过程示意图

传统的混合式更新方法[76-77]由于没有为追加的数据差值预留空间，追加的数据差值实际上是随机分布在磁盘上的，导致合并数据时产生大量的随机读操作. 针对该问题，Chan 等人[78]提出了一种预留空间的混合式更新方法 PLR(Parity Logging with Reserved Space). PLR 在每个校验块后预留一定的空间，使追加的数据可以连续地分布在磁盘上. 这样合并数据时就无需多次访问磁盘，只需在连续的空间读取一次即可，既减轻了磁盘的负载也提高了数据修复的效率. PLR 的不足之处是预留空间导致增加了存储空间的消耗. 为此，Chan 等人[78]设计了一种负载感知的预留空间动态调整方法.

对于以追加的方式处理更新的系统，需要建立有效的索引信息以便快速定位原数据的更新数据. 由于数据更新的位置和大小是随机的，一般把数据划分成大小固定的数据片(如文件系统的 chunk)作为基本单位来管理更新信息. 在磁盘阵列中常用的索引结构是哈希日志列表[75,79]. 如图 15 所示，对于每个被更新的数据片，都会存在一个与之对应的列表来保存其每次更新的信息. 图 15 中，LBA(Logical Block Address)表示每次更新追加的数据的位置，Hash_pre 和 Hash_next 分表表示前一次更新和下一次更新的索引信息的指针. 因此，通过 Hash_pre 和 Hash_next 可以快速遍历一个数据片的所有更新记录. 日志列表适用于数据量较小的系统. 在大规模分布式存储系统中一般使用键/值存储系统来保存追加数据的索引信息. 例如，PLR[78]使用基于 MongoDB① 的键/值存储系统.

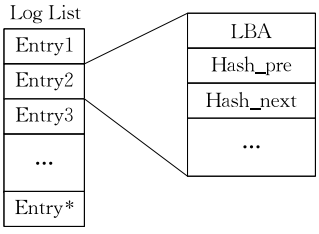


图 15　保存追加数据索引信息的日志列表结构

另外，Pei 等人[80]提出使用树型结构传输更新数据，通过感知机架位置，减少数据更新过程中机架间的数据传输量，提高更新效率.

7.4　总　结

表 3 显示了上述 3 种更新方法的对比情况. 其中，读取量、写入量和传输量分别表示采用(14,10)-RS 码时更新单位数据平均产生的数据读取量、数据写入量和网络传输量. R_L 和 R_H 分别表示追加式更新方法和混合式更新方法因数据合并产生的数据读取量，W_L 和 W_H 则分别表示这两种方法因数据合并产生的数据写入量. 由于混合更新方法不需要合并数据块，在合并频率相同的情况下，混合式更新方法的数据读写量优于追加式更新方法，即有 $R_L>R_H$ 和 $W_L>W_H$. 从表 3 可以看出，替换式更新方法对数据块和校验块进行实时更新的特点，提高了数据访问和修复的效率，但是该方法的数据读取量非常大，明显降低了更新效率. 追加式更新方法通过追加的方式显著降低了数据读取量，提高了数据更新的效率，但增加了存储空间消耗，并降低了数据读取效率. 混合式更新方法对数据块进行实时更新，对校验块进行追加更新，比追加式更新方法具有更高的数据访问性能和更低的存储空间消耗.

表 3　3 种数据更新方法的对比($R_L>R_H$, $W_L>W_H$)

更新方法	读取量	写入量	传输量	读操作	写操作
替换式	5	5	4	连续	随机
追加式	$1+R_L$	$5+W_L$	4	随机	顺序
混合式	$1+R_H$	$5+W_H$	4	连续	随机+顺序

8　其它研究

8.1　带扇区容错的纠删码

带扇区容错的纠删码是针对磁盘扇区出错和 SSD 存储单元易磨损，且这些错误很难发现的问题提出的. 整个磁盘或节点的失效非常容易被探测到，但是存储介质基本单元的失效只有通过全盘扫描才能发现. 庞大的数据量使得无法频繁进行这样的扫描，从而使得系统中存在大量的“隐含”失效. 当系统

① http://www.mongodb.org

探测到磁盘或节点失效时，就有可能因“隐含”失效过多而导致某些数据无法修复. 图 16 显示了一种“隐含”失效导致(6,4)-RS 码无法修复的情况. 其中整个磁盘 0 失效，但是因为磁盘 3 和磁盘 4 的相同位置均存在“隐含”的失效扇区，导致扇区 $d_{0,0}$ 无法修复，同时磁盘 3 和磁盘 4 上的失效扇区 $d_{3,0}$ 和 $p_{0,0}$ 也无法修复，造成数据永久丢失.

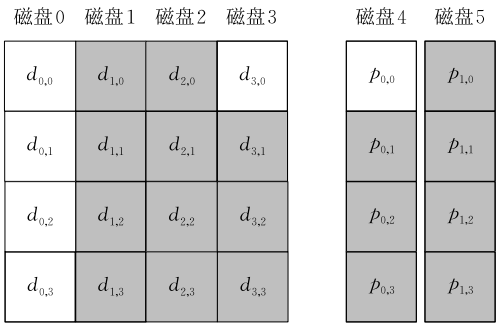


图 16 扇区失效导致(6,4)-RS 码无法修复情况示意图

应对“隐含”失效的简单方法是采用容错能力更强的纠删码. 但是这种方法的存储成本极高，每一个额外扇区的容错能力都要求增加一个额外的冗余磁盘.

针对上述问题，Plank 等人[81-82]提出了 SD (Sector-Disk)码. SD 码能够容忍任意 m 个磁盘和任意 s 个扇区同时失效. 如图 17 所示，除了 m 个校验磁盘外，SD 码任意 s 个扇区的容错能力只需额外增加 s 个扇区的校验数据，存储成本极低.

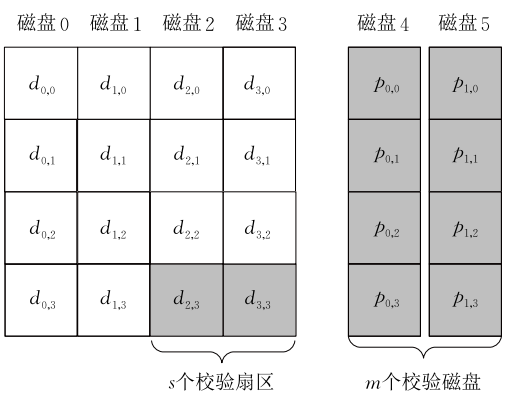


图 17 SD 码的编码结构($m=2, s=2$)

SD 码对失效扇区的分布没有任何要求. 但是 SD 码的 m 和 s 不能任意取值，只有在 m 和 s 的取值满足一定条件的情况下，SD 码才能被构造出来. 并且，SD 码的构造需要进行穷尽搜索. 更严重的问题是，SD 码往往需要在很大的有限域上进行构造，极大地增加了编码和数据修复的复杂度. PMDS (Partial-MDS)码[83]也是带扇区容错的 MDS 码，是 SD 码的子集.

针对 SD 码存在的问题，Li 等人[84]提出了 STAIR 码. STAIR 码同样只需 s 个扇区的额外校验数据即可提供 s 个扇区的容错能力. 与 SD 码能容忍任意 s 个扇区失效不同，STAIR 码限制了包含失效扇区的磁盘个数和每个磁盘包含失效扇区的最大个数. 但是，STAIR 码对任意的 m 和 s 均存在，并且 STAIR 码可以使用统一易行的方法正向构造出来. 此外，STAIR 码的构造也不需要在很大的有限域上进行，编码和修复的复杂度更低.

然而，上述几种带扇区容错纠删码的数据更新成本极高. 这些纠删码中校验磁盘上的每个扇区和每个专门的校验扇区都与所有的数据扇区相关. 这使得更新任意一个数据扇区都要同步更新所有的校验磁盘的所有扇区和所有校验扇区，需要传输大量的数据并进行大量的运算.

8.2 混合使用多种纠删码

不同纠删码具有不同的存储空间开销和数据修复开销. 一般情况下，存储空间开销越高的纠删码其数据修复开销越低；反之亦然. 混合使用多种纠删码可以同时发挥多种纠删码的优势.

Xia 等人[85]提出通过跟踪数据的访问频率来动态调整纠删码的参数，以达到既降低存储空间开销又提高失效数据访问速度的目的. 有些纠删码存储空间利用率较高，但数据修复速度很慢；有些纠删码存储空间利用率较低，但是数据修复速度较快. 我们知道，对系统中数据的访问往往是不均衡的. 有些数据很“热”，经常被访问；有些数据则很“冷”，很少被访问. 数据的“冷”和“热”也是动态变化的. 当数据较“热”的时候使用存储空间开销较大但数据修复较快的纠删码；当数据较“冷”的时候则使用数据修复较慢但存储空间开销较低的纠删码. 这样整体上既能保证较快的失效数据访问速度也能保持较低存储空间开销. 但是，如果使用完全不同种类的纠删码，每次动态转换都相当于完全重新进行了一次编码，成本极高. 较好的方法是在同一种纠删码的不同参数之间转换，这样可以减少转换需要重新产生的校验数据以及数据的读取和传输. 例如，可以采用如 LRC 码[20]这样的分组码，通过调整分组的大小来调整数据修复的速度和存储空间开销. 调整的过程中，全局校验块完全不需要重新产生. 图 18 显示了 LRC(12,6,2)码和 LRC(12,2,2)码的相互转化过程. 从 LRC(12,2,2)码转化为 LRC(12,6,2)码只需将 6 个局部校验块合并为 2 个即可；从 LRC(12,6,2)

码转化为 LRC(12,2,2)码也只需读取 2/3 的数据块和 2 个局部校验块.

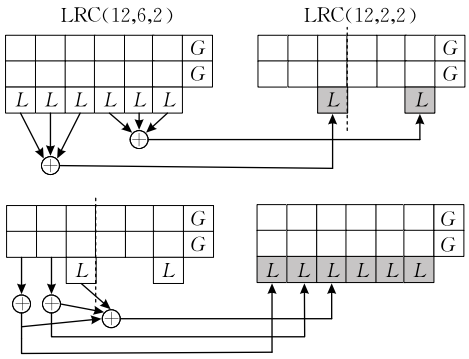


图 18　LRC(12,6,2)码和 LRC(12,2,2)码的相互转化

除了多种纠删码混合使用,多副本也经常和纠删码一起使用.最常见的使用方式是先用多副本保存新产生的数据,过一段时间后再转换为纠删码.这种方式不仅简化了编码实现,也提高了数据的访问速度,因为新产生的数据往往是被频繁访问的“热”数据.HDFS-RAID[34]、微软的云存储系统 WAS[35]、百度云盘的存储系统 Atlas[33]和阿里巴巴的 TFS 等系统都使用了这种方式.另外一种常见的混合使用方式是,用多副本保存那些被频繁访问、对性能有重大影响且规模小的数据,用纠删码保存规模庞大的数据.例如,Zhang 等人[86]在提出在内存键/值存储系统中使用多副本保存规模小且极为分散的元数据和“键”,用纠删码保存规模较大的“值”.TFS 文件系统也使用了这种方式.此外,TFS 文件系统还使用多副本与纠删码相结合的方式来处理数据更新.TFS 文件系统直接将新的数据追加到数据块和校验块之后.更新时不用读取原数据,读取新数据时也不用合并数据.

8.3　磁盘失效预测

近年来,出现的应对数据修复挑战的最新思路是磁盘失效预测.通过预测磁盘的失效,可以提前备份即将失效磁盘上的数据,从而将数据修复转化为复制,不仅可有效降低数据修复开销也可提高数据访问速度.

EMC 公司的 Ma 等人[87]分析了 5 年来约一百万个磁盘的追踪数据,发现磁盘的失效概率与磁盘中再分配扇区的个数有着明显的正相关关系.鉴于此,Ma 等人[87]提出了完全基于磁盘再分配扇区个数的磁盘失效预测方法 RAIDShield,将再分配扇区数目达到一定阈值的磁盘作为即将失效的磁盘.大规模长时间的运行表明,RAIDShield 可以有效地预测磁盘失效,极大减少过多磁盘同时失效导致整个 RAID 失效的概率.

9　研究展望

关于大规模分布式存储中纠删码容错技术,未来值得进一步研究的方向主要包括以下几个方面.

(1)同步编码实现技术

现有的编码实现方法都是异步编码,利用多副本技术来保证编码之前的数据可靠性.在编码之前,数据的多个副本同时提供服务,可以获得更高的访问吞吐率,这对那些需要对数据进行大量分析计算的存储系统具有较大意义.但是,对于无需进行大量运算的系统,比如向用户提供文件存储服务的个人云盘、提供视频流或视频分享的视频存储系统等,由于访问瓶颈存在于互联网,异步编码中的多个副本无法有效提高用户的访问速度,并存在较为严重的资源浪费.因此,有必要研究同步编码实现技术,在数据存储时就立刻进行编码,避免复制数据占用大量磁盘和网络资源.

(2)低冗余再生码设计

再生码能够显著降低数据修复开销,但是其存储开销较大.现有的关于再生码的研究工作都是在一定数据修复开销的前提下,大致估算存储空间消耗,然后再构造具体编码.但是以存储空间利用率为前提,研究再生码数据修复开销的下界,并进一步构造具体的再生码,也是极具挑战的研究工作.从这个思路出发,就有可能设计出兼顾存储开销和数据修复开销的再生码.

(3)数据失效预测技术

较高的数据修复开销一直是纠删码在大数据存储中面临的一个重大挑战.设计新的纠删码和数据修复技术可以在一定程度上解决这些问题,但是无法根本解决问题.Ma 等人[87]对磁盘失效预测的研究为解决这个挑战指出了一个新的有效思路.如果能够预测节点或磁盘的失效,就可以提前复制其中的数据,避免失效后的数据修复.根据我们的初步估算,即使失效预测的准确率只有 50%,也可将数据修复开销降低 40%.

10　总　结

构建于商用廉价服务器之上的分布式存储系统具有服务能力强、扩展性好和成本低廉等优点,被广泛应用于大数据的存储管理.分布式存储系统中节

点失效的常态化使得数据容错不可或缺.近年来,纠删码作为一种容错能力强且存储开销小的数据容错技术,受到了业界的广泛关注.本文首先指出了纠删码容错技术在分布式存储中面临的主要技术挑战;然后分类综述了分布式存储中纠删码容错技术的相关研究进展,重点分析了各种方法的优缺点;最后,本文基于最新研究动态对今后的研究方向进行了展望.

参 考 文 献

[1] Jagadish H V, Gehrke J, Labrinidis A, et al. Big data and its technical challenges. Communications of the ACM, 2014, 57(7): 86-94

[2] Turner V, Gantz J F, Reinsel D, et al. The digital universe of opportunities: Rich data and the increasing value of the internet of things. International Data Corporation, White Paper, IDC_1672. 2014

[3] Patterson D A, Gibson G, Katz R H. A case for redundant arrays of inexpensive disks (RAID)//Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data. Chicago, USA, 1988: 109-116

[4] Milanovic S, Petrovic Z. Building the enterprise-wide storage area network//Proceedings of the 2001 International Conference on Trends in Communications. Bratislava, Slovak Republic, 2001: 136-139

[5] Ghemawat S, Gobioff H, Leung S-T. The Google file system//Proceedings of the 9th ACM Symposium on Operating Systems Principles. Bolton Landing, USA, 2003: 29-43

[6] Shvachko K, Kuang H, Radia S, et al. The Hadoop distributed file system//Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies. Incline Village, USA, 2010: 1-10

[7] Ford D, Labelle F, Popovici F I, et al. Availability in globally distributed storage systems//Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation. Vancouver, Canada, 2010: 1-7

[8] Vishwanath K V, Nagappan N. Characterizing cloud computing hardware reliability//Proceedings of the 1st ACM Symposium on Cloud Computing. Indianapolis, USA, 2010: 193-204

[9] Ouyang J, Lin S, Jiang S, et al. SDF: Software-defined flash for web-scale Internet storage systems//Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems. Salt Lake City, USA, 2014: 471-484

[10] Gill P, Jain N, Nagappan N. Understanding network failures in data centers: Measurement, Analysis, and Implications//Proceedings of the ACM SIGCOMM 2011 Conference. Toronto, Canada, 2011: 350-361

[11] Sathiamoorthy M, Asteris M, Papailiopoulos D, et al. XORing elephants: Novel erasure codes for big data//Proceedings of the 39th International Conference on Very Large Data Bases. Trento, Italy, 2013: 325-336

[12] 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//Proceedings of the 5th USENIX Conference on Hot Topics in Storage and File Systems. San Jose, USA, 2013: 8-8

[13] Wang Y, Li S. Research and performance evaluation of data replication technology in distributed storage systems. International Journal of Computers and Mathematics with Applications, 2006, 51(11): 1625-1632

[14] Plank J S. Erasure codes for storage applications. Tutorial Slides Presented at FAST-2005: The 4th USENIX Conference on File and Storage Technologies. San Francisco, USA, 2005

[15] Lin W K, Chiu D M, Lee Y B. Erasure code replication revisited//Proceedings of the 4th International Conference on Peer-to-Peer Computing. Washington, USA, 2004: 90-97

[16] Weatherspoon H, Kubiatowicz J. Erasure coding vs. replication: A quantitative comparison//Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS). London, UK, 2002: 328-338

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

[18] Luo Xiang-Hong, Shu Ji-Wu. Summary of research for erasure code in storage system. 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. 计算机研究与发展, 2012, 49(1): 1-11)

[19] Plank J S. Erasure codes for storage systems: A brief primer. The USENIX Magazine, 2013, 38(6): 44-50

[20] Huang C, Simitci H, Xu Y, et al. Erasure coding in windows azure storage//Proceedings of the 2012 USENIX Conference on Annual Technical Conference. Cascais, Portugal, 2012: 2-13

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

[22] Roth R M, Lempel A. On MDS codes via Cauchy matrices. IEEE Transactions on Information Theory, 1989, 35(6): 1314-1319

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

[24] Corbett P, English B, Goel A, et al. Row-diagonal parity for double disk failure correction//Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST). San Francisco, USA, 2004: 1-14

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

[26] Plank J S. The RAID-6 Liberation codes//Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST). San Jose, USA, 2008: 1-14

[27] Plank J S. The Raid-6 Liber8Tion code. International Journal on High Performance Computing Applications, 2009, 23(3): 242-251

[28] Huang C, Xu L. STAR: An efficient coding scheme for correcting triple storage node failures//Proceedings of the 4th USENIX Conference on File and Storage Technologies. San Francisco, USA, 2005: 197-210

[29] Fan J, Shou L, Dong J. F-code: An optimized MDS array code//Proceedings of the 3rd International Conference on Advanced Intelligent Computing Theories and Applications. Qingdao, China, 2007: 1325-1336

[30] Jin C, Jiang H, Feng D, et al. P-Code: A new RAID-6 code with optimal properties//Proceedings of the 23rd International Conference on Supercomputing. Yorktown Heights, USA, 2009: 360-369

[31] Lin S, Wang G, Stones D, et al. T-Code: 3-erasure longest lowest-density MDS codes. IEEE Journal on Selected Areas in Communications, 2010, 28(2): 289-296

[32] Luo X, Shu J. Generalized X-code: An efficient RAID-6 code for arbitrary size of disk array. ACM Transactions on Storage, 2012, 8(3): 10:1-10:16

[33] Lai C, Jiang S, Yang L, et al. Atlas: Baidu's key-value storage system for cloud data//Proceedings of the 31st Symposium on Mass Storage Systems and Technologies. Santa Clara, USA, 2015: 1-14

[34] Fan B, Tantisiriroj W, Xiao L, et al. DiskReduce: RAID for data-intensive scalable computing//Proceedings of the 4th Annual Workshop on Petascale Data Storage. New York, USA, 2009: 6-10

[35] Calder B, Wang J, Ogus A, et al. Windows Azure Storage: A highly available cloud storage service with strong consistency//Proceedings of the 23rd ACM Symposium on Operating Systems Principles. Portland, Oregon, 2011: 143-157

[36] Pamies-Juarez L, Datta A, Oggier F. RapidRAID: Pipelined erasure codes for fast data archival in distributed storage systems//Proceedings of the 2013 IEEE INFOCOM. Turin, Italy, 2013: 1294-1302

[37] Pamies-Juarez L, Oggier F, Datta A. Decentralized erasure coding for efficient data archival in distributed storage systems. Distributed Computing and Networking, 2013, 7730: 42-56

[38] Khan O, Burns R, Plank J, et al. Rethinking erasure codes for cloud File systems: Minimizing I/O for recovery and degraded reads//Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST). San Jose, USA, 2012: 20-33

[39] Huang C, Chen M, Li J. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems//Proceedings of the 6th IEEE International Symposium on Network Computing and Applications. Cambridge, USA, 2007: 79-86

[40] Zhou Song, Wang Yi-Jie. EXPyramid: An array-based flexible coding scheme with high fault-tolerance and low recovery-overhead. Journal of Computer Research and Development, 2011, 48(z1): 30-36(in Chinese)
(周松，王意洁. EXPyramid：一种灵活的基于阵列结构的高容错低修复成本编码方案. 计算机研究与发展，2011，48(z1)：30-36)

[41] Miyamae T, Nakao T, Shiozawa K. Erasure code with shingled local parity groups for efficient recovery from multiple disk failures//Proceedings of the 10th USENIX Conference on Hot Topics in System Dependability. Broomfield, USA, 2014: 5-5

[42] Hafner J L. WEAVER codes: Highly fault tolerant erasure codes for storage systems//Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST). San Francisco, USA, 2005: 16-29

[43] Li M, Shu J, Zheng W. GRID Codes: Strip-based erasure codes with high fault tolerance for storage systems. ACM Transactions on Storage, 2009, 4(4): 15:1-15:22

[44] Hafner J L. HoVer erasure codes for disk arrays//Proceedings of the 2006 International Conference on Dependable Systems and Networks. Philadelphia, USA, 2006: 217-226

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

[46] Jiekak S, Kermarrec A-M, Le S N, et al. Regenerating codes: A system perspective//Proceedings of the 2012 IEEE 31st Symposium on Reliable Distributed Systems. Irvine, USA, 2012: 436-441

[47] Ahlswede R, Cai N, Li S-Y, et al. Network information flow. IEEE Transactions on Information Theory, 2000, 46(4): 1204-1216

[48] Dimakis A, Godfrey P, Wainwright M, et al. Network coding for distributed storage systems//Proceedings of the 26th IEEE INFOCOM. Anchorage, USA, 2007: 2000-2008

[49] Wu Y, Dimakis A G. Reducing repair traffic for erasure coding-based storage via interference alignment//Proceedings of the 2009 IEEE International Symposium on Information Theory. Seoul, Korea, 2009: 2276-2280

[50] Rashmi K, Shah N B, Kumar P V, et al. Explicit construction of optimal exact regenerating codes for distributed storage//Proceedings of the 47th Annual Allerton Conference on Communication, Control, and Computing. Allerton House, Illinois, 2009: 1243-1249

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

[52] Lin S-J, Chung W-H. Novel repair-by-transfer codes and systematic exact-MBR codes with lower complexities and smaller field sizes. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(12): 3232-3241

[53] Rashmi K, Nakkiran P, Wang J, et al. Having your cake and eating it too: Jointly optimal erasure codes for I/O, storage, and network-bandwidth//Proceedings of the 13th USENIX Conference on File and Storage Technologies. Santa Clara, USA, 2015: 81-94

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

[55] Wang Z, Dimakis A, Bruck J. Rebuilding for array codes in distributed storage systems//Proceedings of the 2010 IEEE GLOBECOM Workshops. Miami, USA, 2010: 1905-1909

[56] Xu S, Li R, Lee P, et al. Single disk failure recovery for X-Code-based parallel storage systems. IEEE Transactions on Computers, 2013, PP(99): 1-1

[57] Khan O, Burns R, Park J, et al. In search of I/O-optimal recovery from disk failures//Proceedings of the 3rd USENIX Conference on Hot Topics in Storage and File Systems. Portland, USA, 2011: 6-10

[58] Zhu Y, Lee P, Hu Y, et al. On the speedup of single-disk failure recovery in XOR-coded storage systems: Theory and practice//Proceedings of the 2012 IEEE 28th Symposium on Mass Storage Systems and Technologies. Pacific Grove, USA, 2012: 1-12

[59] Rashmi K, Shah N B, Gu D, et al. A "Hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers//Proceedings of the 2014 ACM Conference on SIGCOMM. Chicago, USA, 2014: 331-342

[60] Rashmi K, Shah N, Ramchandran K. A piggybacking design framework for read-and download-efficient distributed storage codes//Proceedings of the 2013 IEEE International Symposium on Information Theory Proceedings. Istanbul, Turkey, 2013: 331-335

[61] Li J, Yang S, Wang X, et al. Tree-structured data regeneration with network coding in distributed storage systems// Proceeding of the 17th International Workshop on Quality of Service (IWQoS). Charleston, USA, 2009: 1-9

[62] Li J, Yang S, Wang X, et al. Tree-structured data regeneration in distributed storage systems with regenerating codes// Proceedings of the 2010 IEEE INFOCOM. San Diego, USA, 2010: 2892-2900

[63] Sun W, Wang Y, Pei X. Tree-structured parallel regeneration for multiple data losses in distributed storage systems based on erasure codes. China Communications, 2013, 10(4): 113-125

[64] Benson T, Akella A, Maltz D A. Network traffic characteristics of data centers in the wild//Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement. Melbourne, Australia, 2010: 267-280

[65] Hennessy J L, Patterson D A P. Computer architecture: A quantitative approach. 5th ed. San Francisco, USA: Morgan Kaufmann, 2011

[66] Benson T, Anand A, Akella A, et al. Understanding data center traffic characteristics. ACM SIGCOMM Computer Communication Review, 2010, 40(1): 92-99

[67] Zeng T, LiuL, ZhaoJ, et al. Router supported data regeneration protocols in distributed storage systems//Proceedings of the 2011 3rd International Conference on Ubiquitous and Future Networks (ICUFN). Dalian, China, 2011: 315-320

[68] Zhang J, Liao X, Li S, et al. Aggrecode: Constructing route intersection for data reconstruction in erasure coded storage// Proceedings of the 2014 IEEE INFOCOM. Toronto, Canada, 2014: 2139-2147

[69] Aguilera M K, Janakiraman R, Xu L. Using erasure codes efficiently for storage in a distributed system//Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN). Yokohama, Japan, 2005: 336-345

[70] Rawat A S, Vishwanath S, Bhowmick A, et al. Update efficient codes for distributed storage//Proceedings of the 2011 IEEE International Symposium on Information Theory Proceedings (ISIT). Cambridge, USA, 2011: 1457-1461

[71] Aguilera M K, Janakiraman R, Xu L. On the erasure recoverability of MDS codes under concurrent updates// Proceedings of the 2005 International Symposium on Information Theory (ISIT). Adelaide, Australia, 2005: 1358-1362

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

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

[74] Buyya R, Cortes T, Jin H. Parity logging overcoming the small write problem in redundant disk arrays. High Performance Mass Storage and Parallel I/O: Technologies and Applications. Wiley-IEEE Press, San Francisco, USA, 2002: 66-80

[75] Jin C, Feng D, Jiang H, et al. RAID6L: A log-assisted RAID6 storage architecture with improved write performance// Proceedings of the 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies. Denver, USA, 2011: 1-6

[76] Matthews J N, Roselli D, Costello A M, et al. Improving the performance of log-structured file systems with adaptive methods//Proceedings of the 60th ACM Symposium on Operating Systems Principles. Saint Malo, France, 1997: 238-251

[77] Yue Y, He B, Tian L, et al. Rotated logging storage architectures for data centers: Models and optimizations. IEEE Transactions on Computers, 2016, 65(1): 203-215

[78] Chan J C W, Ding Q, Lee P P C, et al. Parity logging with reserved space: Towards efficient updates and recovery in erasure-coded clustered storage//Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST). Santa Clara, USA, 2014: 163-176

[79] Mao B, Jiang H, Wu S, et al. HPDA: A hybrid parity-based disk array for enhanced performance and reliability. ACM Transactions on Storage, 2012, 8(1): 4:1-4:20

[80] Pei X, Wang Y, Ma X, et al. T-Update: A tree-structured update scheme with top-down transmission in erasure-coded systems//Proceedings of the 35th Annual IEEE International Conference on Computer Communications (IEEE INFOCOM 2016). San Francisco, USA, 2016: 1-9

[81] Plank J S, Blaum M, Hafner J L. SD codes: Erasure codes designed for how storage systems really fail//Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST). San Jose, USA, 2013: 95-104

[82] Plank J S, Blaum M. Sector-Disk (SD) erasure codes for mixed failure modes in RAID systems. ACM Transactions on Storage, 2013, 10(1): 4:1-4:17

[83] Blaum M, Hafner J L, Hetzler S. Partial-MDS codes and their application to RAID type of architectures. IEEE Transactions on Information Theory, 2013, 59(7): 4510-4519

[84] Li M, Lee P P C. STAIR codes: A general family of erasure codes for tolerating device and sector failures in practical storage systems//Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST). Santa Clara, USA, 2014: 147-162

[85] Xia M, Saxena M, Blaum M, et al. A table of two erasure codes in HDFS//Proceedings of the 13th USENIX Conference on File and Storage Technologies. Santa Clara, USA, 2015: 213-226

[86] Zhang H, Dong M, Chen H. Efficient and available in-memory KV-Store with hybrid erasure coding and replication//Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST 16). Santa Clara, USA, 2016: 167-180

[87] Ma A, Douglis F, Lu G, et al. RAIDShield: Characterizing, monitoring, and proactively protecting against disk failures//Proceedings of the 13th USENIX Conference on File and Storage Technologies. Berkeley, USA, 2015: 241-256

WANG Yi-Jie, born in 1971, Ph.D., professor, Ph.D. supervisor. Her research interests include distributed storage, big data analysis and cloud computing.

XU Fang-Liang, born in 1989, Ph.D. candidate. His research interests include distributed storage and erasure codes.

PEI Xiao-Qiang, born in 1986, Ph.D. candidate. His research interests include distributed storage and erasure codes.

Background

Protecting data from frequent node failures and ensuring data reliability is essential to large scale distributed storage systems built from inexpensive commodity hardware. As an alternative to ensure data reliability that incurs much less storage overheads and can offer higher data reliability at the same time, erasure coding has been widely studied in the context of distributed storage over the past years and many great progresses have been made in the field. This paper gives a comprehensive overview of the current state of the art. Firstly, we briefly introduce erasure coding and point out the main technical challenges of integrating erasure coding into distributed storage systems. Secondly, we provide a comparison, analysis and peroration of the latest research in the field from the aspects of erasure codes, data encoding, data repair and data update. Finally, based on the above analysis, we point out some future work that can promote the further development of erasure coding in distributed storage systems, including synchronous data encoding, regenerating codes with low redundancy and data failure forecasting.

This work is supported by the National High Technology Research and Development Program (863 Program) of China under Grant No. 2013AA01A213. The project aims to make breakthroughs in cloud computing from the aspects of systems supporting extremely high concurrent rate load, EB scale cloud storage and cloud service management etc., and build an own-control cloud computing platform based on these breakthroughs. The work of our research team is to build a high reliable, cost-effective and high scalable distributed storage infrastructure for other parts of the program. Works related to this have been published in international conferences and journals, such as INFOCOM, CCPE, etc. As an economical and powerful technology to ensure data reliability, erasure coding has found successful applications in many large scale distributed storage systems. This review paper can help us to get a comprehensive understanding of the pros and cons of erasure coding and thus advance our research.