计 算 机 学 报
CHINESE JOURNAL OF COMPUTERS

分布式存储系统中纠删码容错技术性能优化研究综述

李梓航

华中科技大学 计算机科学与技术学院,武汉市 430074

摘 要 随着大数据时代的到来，海量数据的存储面临着巨大挑战。大规模分布式存储系统以其海量存储能力、高吞吐量、高可用性和低成本的突出优势取代了集中式存储系统。由于分布式存储系统中节点数量庞大，经常会产生各种类型的故障。因此，必须采用容错技术来保证在部分存储节点失效的情况下，数据仍然能够被正常读取和下载，而容错能力必然会带来额外成本，因此，具有容错能力且节约资源的分布式存储编码成为大数据时代重点研究的核心技术之一。本文讨论了大数据背景下存储与可靠性的问题，从而引出数据容错对分布式存储的重要性，以及降低容错技术带来的额外成本的重要性。阐述了多副本机制和纠删码技术，纠删码技术能在保持高可靠的同时降低存储成本。重点讨论了以下 5 种基于纠删码的分布式存储系统容错技术性能优化算法：1）CRaft 降低存储和网络成本；2)结合局部信息的宽条带纠删码；3)全节点修复框架 RepairBoost；4)几何分区提升纠删码恢复性能；5)PDL 加速故障恢复

关键词 纠删码；分布式存储系统；大数据

A Review of Performance Optimization Research on Fault-Tolerant Techniques in Distributed Storage Systems with Erasure Code

ZiHang Li

Abstract With the advent of the era of big data, the storage of massive data is facing great challenges. Large-scale distributed storage systems have replaced centralized storage systems with their outstanding advantages of massive storage capacity, high throughput, high availability and low cost. Due to the large number of nodes in distributed storage systems, various types of failures are often generated. Therefore, fault-tolerant technology must be used to ensure that data can still be read and downloaded normally in case of partial storage node failure, and fault-tolerance inevitably brings additional costs, so distributed storage coding with fault-tolerance and resource-saving becomes one of the core technologies focused on research in the era of big data. This paper discusses the issue of storage and reliability in the context of big data, which leads to the importance of data fault tolerance for distributed storage and the importance of reducing the extra cost brought by fault tolerance techniques. The multi-copy mechanism and corrective coding techniques are described, and Erasure coding techniques can reduce storage costs while maintaining high reliability. The following five algorithms are discussed to optimize the performance of distributed storage systems based on censoring: 1) CRaft to reduce storage and network costs; 2) Wide-strip Erasure code with local information; 3) RepairBoost, a full-node repair framework; 4) Geometric partitioning to improve Erasure code recovery performance; 5) PDL to accelerate fault recovery

Key words Erasure Code; Distributed Storage Systems; Big Data

1 引言

在当前云计算时代，全球流量快速增长，互联网、物联网、移动终端、安全监控和金融等领域的数据量呈现出“井喷式”增长态势。存储在云服务器中数据的增长速率甚至超越了摩尔法则，云存储系统成为云计算关键组成部分之一。数据的飞速增长对存储系统的性能和扩展性提出了更苛刻的挑战。传统的存储系统采用集中式的方法存储数据，使得数据的安全性和可靠性均不能保证，不能满足大数据应用的需求。

分布式存储系统以其巨大的存储潜力、高可靠性和易扩展性等优点成为大数据存储的关键系统并被推广应用到降低存储负荷、疏通网络拥塞等业务领域上，且其以高吞吐量和高可用性成为云存储中的主要系统。目前 Hadoop 分布式文件系统(HDFS)作为谷歌文件管理系统(DFS)[1]的开源实现已成为最主流的分布式系统，它被应用于许多大型企业，如脸书，亚马逊等。为了应对海量数据的存储需求，该类存储系统的规模往往非常庞大，一般包含几千到几万个存储节点不等。早在 2014 年，中国互联网百度公司单个集群的节点数量就超过了 10000[2]。近两年，腾讯云的分布式调度系统 VStition 管理和调度单集群的节点数量可达 100000。

然而数量庞大的节点集群经常会产生如电源损坏、系统维修及网络中断等故障致使节点失效频发。据一些大型分布式存储系统的统计数据表明,平均每天都会有 2 %左右的节点发生故障[3]。在过去一年中，迅速发展的云计算市场不断涌出如谷歌云、亚马逊 AWS、微软 Azure 及阿里云等主流云服务器大型宕机事件。由此可见，全球的云服务器发生故障导致数据丢失的情况时有发生。显然，能够可靠存储数据并有效修复失效数据的容错技术对分布式存储系统的重要性不言而喻。因此如何及时有效地修复失效节点，确保数据能被正常地读取和下载就显得非常重要。据统计，在一个有 3000 个节点的脸书集群中，每天通常至少会触发 20 次节点修复机制。具有节点修复能力的数据容错技术是通过引入元余的方式来保证分布式存储系统的可靠性，存储系统能够容忍一定数量的失效节点，即提高了系统的数据容错能力。

为了保证用户访问数据的可靠性、维持分布式存储系统的容错性，需要及时修复失效节点。良好的容错技术要求存储系统具有低的冗余开销、低的节点修复带宽、低的编译码复杂度等特点。如何降低失效节点的修复带宽、降低磁盘 IO、降低存储系统编译码的复杂度、提高系统的存储效率成为分布式存储中研究的热门方向。

比较传统的维护数据可靠性的方案是多副本机制,许多实际的存储系统(如 Ceph,HDFS,Swift)存放原始数据及它们的两个副本这会大幅增加存储成本。多副本机制是产生几余最基本的方式。为了弥补数据失效带来的损失，确保存储系统中数据的完整性，将数据的副本存储在多个磁盘中，只要有一个副本可用，就可以容忍数据丢失。如果数据以最常见的 3 副本方式存储，那么原始数据会被复制到 3 个磁盘上，因此任何 1 个磁盘故障都可以通过剩余 2 个磁盘中任意 1 个来修复。显然，该 3 副本机制有效的存储空间理论上不超过总存储空间的 1/3,多副本机制显著降低了存储效率。

近年来，纠删码(Erasure Code)以其低存储开销、高可靠性的特点被当作是数据复制的替代方案。其中,Reed-Solomon 码是使用最广泛的纠删码因为它在给定的存储开销下提供了最大的可靠性，并且可以灵活选择确定可实现可靠性的编码参数。

Reed-Solomon 码(以下简称 RS 码)的编码参数一般为(k，m)，意思是通过 k 个原始的数据块编码得到 m 个元余块(编码参数也可以是(n，k)，其中 n=k+m)，这(k+m)个块的集合视为一个条带(Stripe)，当(k+m)个块中的任何 k 个块都足以重建原始数据时，编码完成。例如，在 RS(4，2)码中，将总大小为 256MB 大小的数据分为四个块，每个块大小为 64MB，然后，使用这四个大小为 64MB 的块进行计算，得到两个大小同样为 64MB 的几余块。同样地，若要在三重复制系统下得到冗余，则这四个 64MB 的块均要被复制一遍。综上所述，RS(4，2)编码系统花费 1.5 倍初始存储空间来保证存储系统的可靠性:而三重复制系统花费 3 倍的初始存储空间才能容忍相同数量的同时发生的故障。

然而，虽然纠码系统不需要太多的存储开销去维持数据可靠性，但是纠删码系统在面临数据块失效时，需要多个幸存块来恢复丢失块。以 RS(4，2)系统为例，若有个大小为 64MB 的数据块丢失，则需要 4 个幸存块去恢复丢失块。这意味着，相比于

二重复制系统，RS(4，2)纠删码系统将修复时的网络 I/0 放大了 4 倍(块大小相同时)。因此，为了缓解纠删码中的网络 I/0 放大问题，研究者们提出了各种修复算法，用以减少纠删码系统修复过程中的网络传输时间，甚至有的修复算法理论上能将网络传输时间减少到 0(指单块修复时间)。显然，当纠删码的修复时间在可控范围之内时，纠删码存储系统的可靠性会大大增加。

2 算法背景与优势

2.1 CRaft 降低存储和网络成本

一致性协议能提供高度可靠和可用的分布式服务。在这些协议中，日志条目被完全复制到所有的服务器中。这种完全的条目复制会导致高的存储和网络成本，从而损害性能。纠删码是一种常见的用于减少存储和网络成本的技术，同时还可以保持相同的容错能力（即可靠性）。如果在一致性协议中的这种完全条目复制能被替换为一个纠删码的复制，存储和网络成本能够被大大降低。RS-Paxos是第一个支持纠删编码数据的一致性协议，但其可用性比其他常用的一致性协议差得多，如Paxos和Raft。Zizhong Wang等人，指出了RS-Paxos的有效性问题并试图解决它[6]。基于Raft，他们提出了一个新协议，CRaft。通过两种不同的复制方法，CRaft可以像RS-Paxos一样使用纠删码来降低存储和网络成本，且同时像Raft一样保持其有效性。与原始Raft对比，CRaft可以节省66%的存储空间，达到250%的写吞吐量，并减少60.8%的写延迟。

2.2 结合局部信息的宽条带纠删码

CRaft通过存储数据条带和奇偶校验块达到容错目的。宽条带最近被提出来抑制条带中奇偶校验块的比例，以实现极端的存储节省。然而，宽条带加重了修复成本，而现有能有效修复数据的纠删码方法并不能很好地解决宽条纹问题。Yuchong Hu等人，提出了组合局部性，这是第一个通过结合奇偶校验局部性和拓扑局部性来系统地解决宽条带修复问题的机制[7]。用有效的编码和更新方案进一步增强了组合局部性。在 AmazonEC2 上的实验表明与基于局部的技术相比，组合局部减少了 90.5%的单块修复时间，冗余低至 1.063x。结合奇偶校验局部性和拓扑局部性的特点，以获得更好的权衡从而使宽条纹实际适用。

2.3 全节点修复框架 RepairBoost

纠删码的修复性能还受到大量修复流量的严重阻碍。近年来的一些工作通过设计新型纠删码或提出单条带修复优化算法来提升修复性能，但是这些方法直接应用在单节点修复上将会受到若干限制。Shiyao Lin等人，提出一种通用的调度框架RepairBoost，可以协助现有的线性纠删码和单条带修复优化算法加速全节点修复[4]。RepairBoost建立在三个设计原语的基础之上：（i）修复抽象，使用特定的有向无环图来描述单个块的修复过程；（ii）修复流量平衡，平衡系统中各节点上传和下载的修复流量负载；（iii）传输调度，仔细地调度请求的数据块的传输顺序，以尽量饱和地占用各个时刻系统中节点的上传和下载带宽。实验结果表明，对于多种编码和修复算法，RepairBoost可以将修复速度提高35.0-97.1％。

2.4 几何分区提升纠删码恢复性能

再生码是一类特殊的纠删码，它的提出是为了尽量减少纠删码修复所需的数据量。Yingdi Shan等人，评估了最佳修复如何有助于改善对象存储系统，发现再生码带来了独特的挑战：再生码以块而不是字节为粒度进行修复，而块大小的选择导致了流式降级阅读时间和修复吞吐量之间的矛盾。为了解决这个难题，Yingdi Shan等人，提出了"几何分区"，它将每个对象分成一系列大小为几何序列的块，以获得大块和小块的好处[8]。Geometric Partitioning帮助再生代码达到1.85倍于RS代码的恢复性能，同时保持较低的降级读取时间。

2.5 PDL 加速故障恢复

传统的随机数据放置会导致大量的跨架流量和故障恢复期间的严重不平衡负载，使纠删码恢复性能大大降低。此外，在分布式存储系统中共存的各种纠删码策略也加剧了上述问题。Liangliang Xu等人，提出了PDL，一种基于PBD的数据布局，以优化DSS的故障恢复性能[5]。PDL是基于配对平衡设计构建的，这是一种具有统一数学特性的组合设计方案，因此呈现出一种统一的数据布局。然后，Liangliang Xu等人，提出了rPDL，一种基于PDL的故障恢复方案。rPDL通过统一选择替换节点和检索确定的可用块来恢复丢失的块，有效地减少了跨架流量，并提供近乎平衡的跨架流量分布。他们在Hadoop 3.1.1中实现了PDL和rPDL。与HDFS现有的

数据布局相比，实验结果表明，rPDL平均减少了62.83%的降级读取延迟，提供了6.27倍的数据恢复吞吐量，并为前端应用提供了更好的支持。

3 算法原理

3.1 结合 RS 纠删码的 Raft 版本

Zizhong Wang等人提出了一个支持纠删码码的Raft版本CRaft。在CRaft中，leader有两种方法将日志条目复制给其follower。如果leader能与足够多的follower沟通，它将通过编码碎片复制日志条目以获得更好的性能。否则，它将复制完整的日志条目，以保证有效性。与RS-Paxos一样，CRaft可以处理纠删码的数据，因此它可以节省存储和网络成本。然而，CRaft和RS-Paxos之间的一个主要区别是，CRaft具有与Paxos和Raft相同水平的活性，而RS-Paxos则没有。CRaft算法有以下两种复制方法：

1.编码片段复制

在Raft中，每个日志条目都应包含其来自客户端的原始内容，以及其在协议中的term和index。当一个Leader试图编码一个条目时，可以通过协议选择的(k，m)-RS代码将内容编码成N= (k+m)个片段。term和index不应该被编码，因为它们在协议中起着重要的作用。编码过程如图3.1。

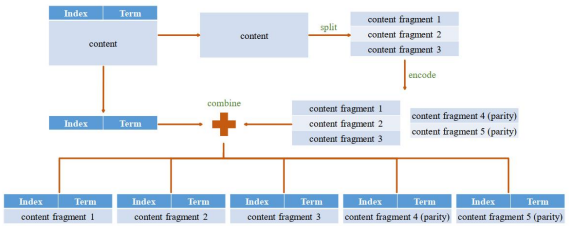

图3.1 CRaft的encoding过程

编码后，leader 将有 N 个条目的编码片段。然后，它将向每个 follower 发送相应的编码片段。在收到其相应的编码片段后，每个 follower 将回复 leader。当 leader 确认至少有(F+k)个服务器存储了一个编码片段时，这个条目和它之前的条目就可以安全地被应用。

leader 将承诺并应用这些条目，然后通知 follower 应用它们。编码碎片复制的承诺条件比 Raft 的更严格。这个承诺条件也意味着，当没有(F +k)健康的服务器时，leader 不能使用编码-碎片复制来复制一个条目用来提交它。

根据编码程序，编码碎片的大小约为完整条目大小的 1/k。因此，使用编码碎片复制时，存储和网络成本可以大大降低。

2.全条目复制

为了减少存储和网络成本，鼓励 leader 使用编码碎片复制。使用编码碎片复制。然而，当没有(F+k)个健康的片段时，编码的片段当没有(F+k)健康的服务器时，编码分片复制将无法工作。编码复制将不起作用。当健康服务器的数量大于 F 并且小于（F+k）时，leader 应该使用完整条目复制方法来复制条目。

在完整条目复制中，leader 必须将完整的条目复制到至少(F+1)个服务器上。在提交条目之前，leader 必须将完整的条目复制到至少(F+1)个服务器上。就像 Raft 一样。由于提交规则与 Raft 相同由于提交规则与 Raft 相同，安全性和有效性都不是问题。然而，由于 CRaft 支持编码片段，leader 可以通过编码片段复制一个条目，而不是将完整的条目复制给其余的 follower。

使用编码碎片复制而不是完整的条目复制可以获得更好的性能，如果这两种方法都能成功复制。贪婪的策略是，leader 总是试图通过编码-碎片复制来复制条目。

如果它发现没有(F+k)健康的服务器，它就转向以完整条目复制的方式进行复制。然而。如果 leader 已经知道健康服务器的数量小于(F+k)，那么通过编码碎片的第一次复制尝试是没有意义的。

准确地选择复制方法可以达到最佳性能。然而，leader 不能确定其他服务器的状态。因此，它只能预测有多少个当它试图复制一个条目时，它只能预测能与多少个健康的服务器通信。复制一个条目。leader 可以使用最近的心跳回答来估计健康服务器的数量。图 3.2 展示了预测过程。

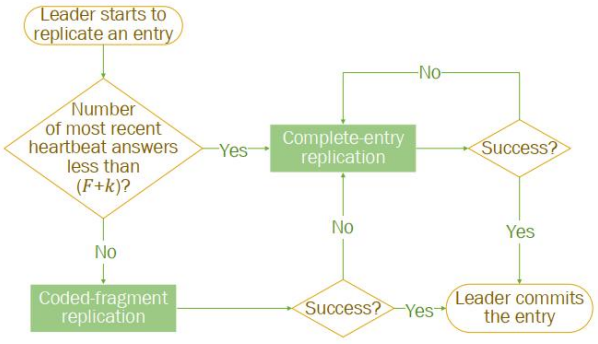

图3.2 CRaft的预测过程

两种复制方法都能在 leader 拥有所有完整条目的情况下保证安全性和活性。然而，当新当选领导人时，新当选领导人的日志很可能没有完整的副

本，而只有一些条目的节点片段。当只有(F+1)个健康服务器时，不能保证这些不完整的条目可以恢复。如果新当选的 leader 没有应用这些不可恢复的项目，该 leader 没有办法处理这些项目。根据 Raft 的规则，leader 从客户端收到的新条目也不能复制到 followers。因此协议无法充分发挥作用，协议的活性也无法得到保证。因此，需要一些额外的操作来保证活性.

因此，作者引入 LeaderPre 操作:当新 leader 当选时它首先检查自己的日志，找出未应用的代码片段。然后,它向 follower 请求他们自己的日志，重点关注未应用的代码片段的索引。至少收集(F+1)个答案(包括新 leader 本身),或者新 leader 继续等待。新的 leader 应该尝试按顺序恢复其未应用的代码片段。对于每个条目，如果在(F+1)应答中至少有 k 个节点片段或 1 个完整副本，则该条目可以被恢复，但不允许立即提交或应用。否则，新的 leader 应该在其日志中删除此条目和所有以下条目(包括完整条目)。在恢复或删除所有未使用的表项后，可以进行整个 LeaderPre 操作。在 LeaderPre 期间，新当选的领导人应该不断向其他服务器发送心跳，防止它们超时并开始新的选举。

通过心跳信息来预测健康服务器的数量，从而选取全条目复制或编码片段复制两种复制方式。LeaderPre 操作确保新 leader 当选后的 liveness 问题。将纠删码应用到 Raft 里，既可以使数据的一致性得到了保证，又减少了网络和存储的开销。

3.2 结合局部信息的宽条带纠删码

纠删码是一种低成本的分布式存储冗余机制，通过存储数据块和奇偶校验块来实现。宽列通过增加数据块的数量且保持校验块的数量不变来降低存储冗余度，节省了大量存储空间。然而宽列提高了恢复成本，对于现有的较高效率的纠删码恢复方法来说依然不能有效地解决该问题。

宽列主要带来了以下的三个性能挑战：

1.昂贵的恢复成本：（n，k）RS 代码通过检索 k 个块来恢复一个块，宽列中的 k 很大，因此需要更多带宽和 I/O，与传统的方法相比成本将增加数倍。

2.昂贵的编码成本：随着 k 的增加，计算开销增大，编码过程中 CPU 缓存有限,导致编码性能下降。

3.昂贵的更新成本：任何更新的数据块都会导致所有 n-k 奇偶校验块更新。

面对上面三个挑战，提出了三个主要的技术点：

恢复方法 对于单块恢复来说，ECWide 实现了组合局部性（组合了 LRC 和 TL）的编码方法。考虑一种存储系统，给定 k、f（容错数量）和 γ（冗余度）。在步骤 1 中，ECWide 通过文中给出的约束确定参数 n 和 r。然后将 k 个数据块编码成 n−k 个局部/全局奇偶校验块。在步骤 2 中，ECWide 为每个局部组选择(r +1)/f 个机架。上述两个步骤确保了单块恢复的跨机架恢复带宽最小化。对于整个节点恢复来说，一个全节点恢复可以被看作是多个单块恢复过程，可以并行化恢复。然而多个单块恢复可能选择相同的节点作为请求者或局部恢复者，从而导致所选节点过载，降低了整体节点恢复性能。为此，ECWide 设计了一个最近最少选择的方法来选择节点作为请求者或本地恢复器，并通过一个双链表和 hashmap 来实现该结构。

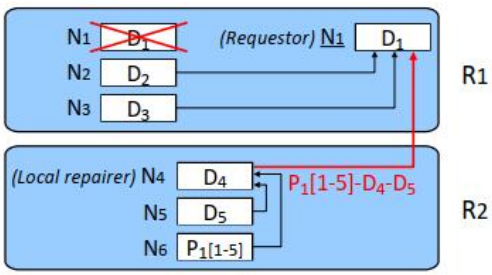

图 3.3 ECWide 节点恢复

编码方法 为了克服单节点编码的局限性，作者考虑了一种多节点编码方案，其目标是实现宽列的编码高吞吐量。其思想是将具有大 k 的单节点编码操作划分为跨不同节点的小 k 的多个编码子操作。原因如下：(1)带有小 k(例如，k = 16)的条带的编码性能很快;(2)奇偶校验块是数据块的线性组合;(3)同一机架内节点之间的带宽往往是充足的。

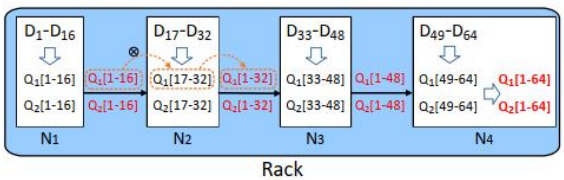

图3.4 ECWide多节点编码

更新方法 对于全局校验块更新来说，ECWide 首先在全局块 Q1[1-20]的机架上传输增量块 D’1->D1。它通过修改 α（D’1-D1）来更新 Q1[1-20]，其中 α 是 Q1[1-20]中 D1 的编码系数。ECWide 仅通过机架内数据传输增量块来更新另一个全局奇偶

校验块 Q2[1-20]。对于局部校验块更新来说，在每个条带中，ECWide 会首先记录每个机架数据块的更新频率，并为每个局部组查找更新最密集的机架。通过这种方式，P1[1-5]被移动到更新最密集的机架上，这样局部奇偶校验更新可以在机架内执行，而不会引起机架间的数据传输。

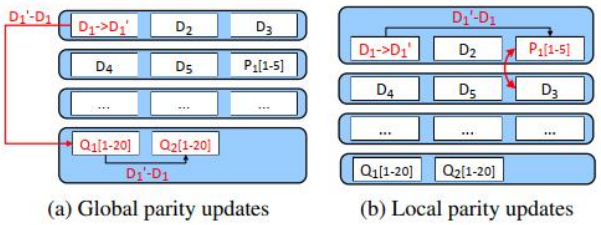

图 3.5 ECWide 更新

作者提出了组合局部性的宽列方法，这是第一个通过结合奇偶校验局部性和拓扑局部性来系统地解决宽列恢复问题的机制。作者进一步利用有效的编码方法和更新方案增强了组合局部性。通过具体实现和实验验证了方法有效。

3.3 RepairBoost 加速全节点修复

纠删码存储中的修复通常分为：（i）全节点修复，恢复故障节点中所有丢失的块（例如，由磁盘故障引起），以及（ii）对暂时不可用的块（例如，由系统升级和网络断开引起）或全节点修复中尚未修复的块的降级读取。在本文中，作者主要关注单一的全节点修复，这被认为是导致服务停机的主要原因之一。

全节点修复必须跨多个条带进行修复操作。虽然存储效率高，但纠删码容易产生高的修复惩罚。例如，RS(k;m)需要检索同一条带的 k 个幸存的块来恢复一个丢失的块，从而使修复中的存储和网络 I/O 放大了 k 倍。具体来说，假设一个块 C^* 发生故障，$\{C_1,C_2,...,C_K\}$是 C^* 的同一条带中的 k 个块的集合。可以用 k 个块的线性组合来修复 C^*，即:

$$C^* = \sum_{i=1}^{k} \beta_i C_i \qquad (1)$$

其中β_i是 C_i用于修复 C^* 的解码系数。

RepairBoost 调度框架可以协助各种擦除码和修复算法来提升全节点修复性能。

假设：RepairBoost 是基于以下假设而设计的。首先，RepairBoost 主要关注单一的全节点修复，据报道这是在实践中最主要的故障事件（例如，占总故障事件的 98%）。然而，RepairBoost 可以扩展到处理多节点故障（即一个条带中超过一个节点发生故障）。其次，为了简化作者的讨论，作者在同质环境（即具有相同的链路带宽）中使用 RS 码来阐述 RepairBoost，但作者也表明 RepairBoost 适用于其他擦除码，可以部署在异质环境（即具有不同的链路带宽）。

概述。它首先通过一个一般的修复有向无环图（RDAG）抽象出一个单块修复方案。通过支持多个 RDAG 的调度，RepairBoost 可以实现灵活性（即允许不同修复算法的协作）和通用性（即可用于各种擦除码和修复算法）。

然后，RepairBoost 将多个 RDAG 分解为具有专门修复任务的顶点。然后，它仔细地将修复任务分配给各节点，以便平衡各存活节点的整体上传和下载修复流量。

RepairBoost 最后根据幸存的节点和相应的修复任务构建一个有向网络。

然后，它通过解决最大流量问题来确定要传输的块，以便在每个时隙完全饱和未被占用的上传和下载带宽。

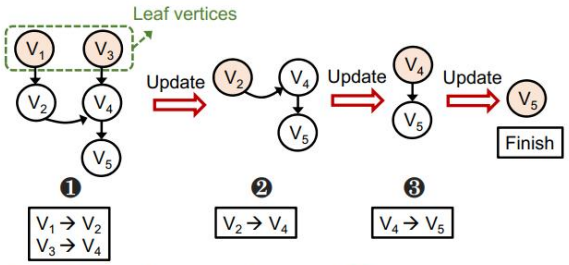

Figure 5: Example of an RDAG of PPR [40] when $k = 4$. The vertex with pink color indicates the leaf vertex. We repeatedly update an RDAG as the repair proceeds.

图 3.6 RDAG 示例

作者以图 3.6 中 PPR 的 RDAG 为例，其中 k=4。假设$\{v_1,v_2,...,v_4\}$中的四个请求存活的块分别用$\{C_1,C_2,...,C_4\}$来表示。在第一阶段，v_1向其父 v_2发送一个块β_1C_1(其中β_1是 C_1的解码系数；见公式(1))，同时 v_3向其父 v_4发送另一个块β_3C_3。作者还通过删除 $e_{1,2}$，$e_{3,4}$, v_1和 v_3来更新 RDAG。在第二阶段，叶子顶点 v_2将β_1C_1和其存储的块 C_2结合起来，并将结果$\beta_1C_1+\beta_2C_2$发送给其父级 v_4。在第三阶段，v_4从其子代收集了两个块后，将$\beta_1C_1+\beta_2C_2$、β_3C_3和其本地存储的块 C_4结合起来，恢复丢失的块 $C^* = \sum_{i=1}^{4} \beta_i C_i$，最后将 C^*发送给 v_5。当 v_5成为叶顶点时，C^*的修复完成。

RepairBoost 调度框架促进各种纠删码和修复算法的全节点修复。RepairBoost 采用了一种称为 RDAG 的图抽象来描述单块修复解决方案。然后细致地将 RDAG 的修复任务分配给节点，以平衡上传和下载修复流量。RepairBoost 进一步调度块的传输，以使空闲带宽饱和。在 AmazonEC2 上的大量实验验证了 RepairBoost 的通用性灵活性和有效性。

3.4 几何分区解决再生码挑战

基于对象的存储系统广泛使用纠删码，以较低的存储开销保证系统的可靠性。但纠删码会带来较大的恢复开销，针对该问题而设计的再生码则通过将数据对象划分为多个数据块，并以数据块为粒度进行恢复，提供了理论上的最优恢复成本。

然而作者观察到，在实际生产系统中再生码并不能同时提供高恢复效率和低降级恢复时间。较小的划分粒度会导致碎片化访问磁盘，从而降低恢复效率，较大的划分粒度则会导致读放大并增加恢复时的等待时间。

作者认为解决这一问题的关键在于提供可变长度的块划分方法，从而同时利用大小块的优势。作者提出了一种新的再生码对象划分方法——几何划分法（Geometric Partitioning），将数据对象划分为大小以几何序列增长的块（如图 3.7 所示，数据对象被划分为 4MB, 8MB, 16MB... 依次增长的数据块），并分别放在对应的桶中进行编码。

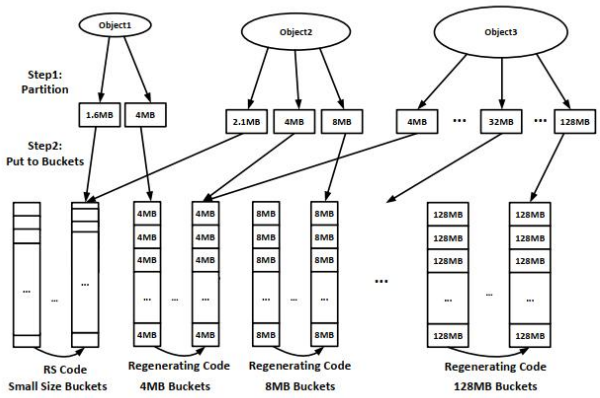

图 3.7 理想的几何划分

作者设计并实现了 RCStor，一个用于存储不可更改的对象的存储系统，以显示 几何分区的有效性。该系统包含 3 个角色，包括目录服务器、存储服务器和 HTTP 服务器。HTTP 服务器负责处理用户 请求。目录服务器存储元信息并监控整个系统。存储服务器存储和管理 对象，并存储索引文件以跟踪对象。每个存储服务器被绑定到一个特定的磁盘。存储服务器的多个进程 存储服务器的多个进程可以运行在同一台机器上的多个 磁盘。恢复任务是由目录服务器派发到 存储服务器。目录服务器的可靠性可以 通过使用复制的状态机来实现。放置组。为了利用更多的磁盘和网卡的带宽进行恢复，作者使用了类似于 Ceph 中的 PG（Placement Group）的概念。作者的系统包含数以千计的 PG，其中每个 PG 是 14 个不同节点上的 k+r=14 个磁盘组，代表要编码的磁盘组。关于 PG 的信息被存储在目录服务器上。每个磁盘可以属于多个 PG，每个 PG 可以独立恢复。当一个磁盘发生故障时，所有与故障磁盘有关的 PG 都将开始恢复。因此，不是只使用 d=13 个磁盘来恢复，而是会有更多的磁盘被利用来加速恢复。Direcotory 服务器将磁盘分配给 PG，这样当一个磁盘发生故障时，就会有最大数量的磁盘与恢复相关联。一个对象首先根据其 ID 的哈希值被映射到一个 PG。然后，选择对该 PG 消耗容量最小的磁盘来放置对象。最后，对象被分割成几块，并放入其相应的桶中。

元数据管理 对于每个 PG，随机选择 r+k 磁盘中的 r+1 来存储该 PG 的复制索引。每个索引文件跟踪该 PG 中所有对象的元数据，并在 Strorage 服务器重新启动时被加载到内存中。索引文件的每条记录包括对象 ID、大小、磁盘 ID、校验和以及对象的分区块的位置。由于一个桶中的所有块都是对齐的，除了小尺寸的桶，一个块的位置可以存储在 2 个字节之内。因此，一个对象的平均元数据大小约为 40 字节，小到可以保存在内存中。

放置和获取 RCStor 通过指定 HTTP 内容类型为 application/octet-stream，向终端用户提供流式 HTTP 接口。

要放置一个对象，我们首先将其放入基于复制的对象存储系统。然后，我们使用后台进程将这些对象从基于复制的存储系统导出到基于擦除码的存储系统中。这种设计类似于 Facebook F4。通过批量导出这些对象，我们可以避免昂贵的奇偶性更新开销。

为了获得一个对象，HTTP 服务器从该 PG 对应的存储服务器中获取索引。如果这是一个降级的读取请求（相应的磁盘离线或在正常读取时发生错误），HTTP 服务器将从存储服务器收集必要的数据，重新生成并传输该对象到客户端。否则，该对象将被直接传输到客户端。

尽管几何分区需要从多个位置读取数据，而不是从一个连续的区域读取数据，但正常读取的性能损失可能很小，因为块的大小足够大，可以摊销磁盘搜索所消耗的额外时间。

IO 调度 在每个存储服务器上，都有一个与多个线程绑定的先进先出（FIFO）请求队列，这些线程不断地从该队列中抽取。每个正常或降级的读取请求都包含一个对象的所有相关块的偏移量和大小，因此该请求不必为一个对象的不同块排队一次以上。为了处理一个请求，存储服务器将把所有相关的数据读入 256KB（或更小）的数据包，并把它们推送到 HTTP 服务器。对存储服务器的后台请求（恢复、数据导入等）被放入单独的队列并以较低的优先级处理。

平行恢复 为了恢复一个节点，RCStor 不是以对象的粒度来恢复，而是以块的粒度来恢复。单个对象的分块可以同时进行恢复。目录服务器上有一个全局队列，管理所有待恢复的块。每个恢复任务都包含单个块的位置信息。HTTP 服务器从目录服务器上连续拉出这些任务，并同时恢复这些块。恢复任务按其块的大小加权（例如，1 代表 4MB 的块，64 代表 256MB 的块），还有一个全局权重代表并发修复块的最大可能的总块大小（例如，如果全局权重是 512，我们可以并发恢复 8 个 256MB 的块，或者我们可以并发恢复 4 个 256MB 的块和 8 个 128MB 的块）。

实验结果表明，RCStor 相比 RS code 和 LRC 分别提高了 1.85 和 1.30 倍的恢复性能，同时保持了较低的平均降级读取时间。揭示了大规模存储系统中应用再生码的实际差距，提出了几何分割，解决了读取时间降低和恢复效率之间的冲突。用真实实际的 traces 验证了它

3.5 PDL 加速故障恢复

一个(n，m)传统编码方法由以下 3 个步骤组成:(i)选择一个替换节点;(ii)从源节点读取同一条带的 n 个块到替换节点;(iii)重构替换节点中的失效块。为了恢复失效节点需要重构其中所有的数据块/校验块，这导致了巨大的 IO 负载和网络流量。因此，加快故障恢复过程对 DSS 至关重限

除了故障恢复所需要的大量 I/O、流量和 CPU 负载外作者还发现以下因素进一步降低了故障恢复的速度。

故障恢复导致机架间的通信量很大 采用随机数据布局方法，当机架数量足够时，每个条带只需存储一个数据块即可达到机架级最大容错能力。但由于故障恢复，它会导致大量的横线交通。

块的非均匀分布 随机数据布局在概率上达到了条带数量足够多的块的近似均匀分布。但由于 I/O、内存空间和计算资源有限，在每轮故障恢复过程中，恢复的故障条带数通常与节点数处于同一数量级，远远不能得到均匀的块分布。

故障恢复时跨机架流量不均衡 在失效恢复过程中，替换节点和检索块的选择对负载均衡也起着至关重要的作用。如果替换节点聚集在某些机架上(如 HDFS 选择邻近的替换节点，导致少数机架上出现替换节点)，那么将检索到的数据读取到替换节点的内存中，会大大增加 ToR 在这些机架上的跨机架下行负载。如果将检索到的数据块聚集在某些机架中，则会从这些机架中读取大量数据进行故障恢复，大大增加了 ToR 跨机架的上行负载。

1. 基于 PBD 的数据布局 PDL

假设我们要在 DSS 中用 τ 个擦除码的混合物来分配数据/奇偶块，$(n_i; m_i)$码为 $0 \leqslant i \leqslant \tau - 1$,如图 4 中的 RS(4;3),RS(6;3),RS(10;4)和 RS(12;4)。

利用 PDL，我们首先将条带的所有块分成一些组，将块的组分配到机架上，并将同一组的块分配到同一机架的节点上，以达到块的均匀分布。因此，在故障恢复过程中，我们可以在机架内部署部分解码，以减少跨机架的流量。

A. 对条带的块进行分组

对于一个$(n_i; m_i)$码，我们应该分配满足以下要求的块以下的要求。

• 同一条带的最多 m_i 个块被分配到同一个机架，以容忍一个机架故障。

• 同一条带的任何两个区块不被分配到同一个节点，以容忍 m_i 个同时发生的节点故障。

为了最大化部分解码的效果，应该最小化条带的组数但从另一个角度来看，一个机架包含最多 m_i 块的相同条带以容忍一个机架故障的(n_i,m_i)代码。因此 $N_g(n_i+m_i)=[(n_i+m_i)/m_i]$是可以容忍一个机架故障的最小组数，称包含精确 m_i 块的组为满的，否则为非满的。为了使块的分布均匀，试图将所有块分成组，使任何组中的块数量最多相差 1。

B. 使用 PBD 表分发数据

(1)将组映射到机架上 当写一个用$(n_i; m_i)$编码的条带时，我们依次查找大小为 $k_i = N_g(n_i; m_i)$的元组的 PBD 表，然后通过随机映射将条带的 k_i 组分

配到指定元组的机架上，每个组到一个机架。例如，写一个 RS(10,4)编码的条带，我们可以根据图 4(b)中的第一个元组(0,3,4,5)，将条带的四个组 G_0、G_1、G_2 和 G_3 分别分配到机架 R_4、R_0、R_3 和 R_5，其中源机架 R_3 和 R_5 为非满载 G_2 和 G_3。在图 4(a)中，$G_{l,i,j}$ 是第 l 个擦除码中第 i 条的 j 组，其中 l=0,1,2,3 分别指 RS(10; 4)、RS(12; 4)、RS(4; 3)和 RS(6; 3)的使用情况。从图 4(a)中，我们发现所有组都均匀地分布在所有机架上。

(2)将块映射到机架内的节点 在每个机架内，同一组的区块以轮流方式分配到节点上，因此区块的分布是均匀的，也就是说，机架内每个节点的区块数量是相同或相差 1。

通过以上两个步骤，我们实现了基于 PBD 表的统一数据分配--PDL。

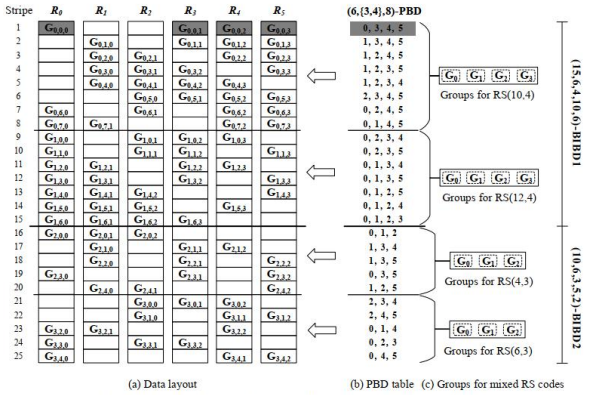

Fig. 4: Data layout of mixed erasure codes based on a $(6, \{3, 4\}, 8)$-PBD.

2. 基于 PDL 的故障恢复方案

假设机架 R_f 中的一个节点 N_f 发生故障。和上一节一样，我们把一个(n,m)代码中的块条分成 $k = N_g(n,m)$组，每组由 m 个或小于 m 个块组成，分别称为全组或非全组。我们将每个组分配到一个单独的机架上，并将拥有完整（非完整）组的机架称为完整（非完整）机架。给定一个(n,m)代码的条带 S，我们把存储 S 块的机架称为源机架。

基于 PDL 的统一数据布局，我们提出了一个单一的故障恢复方案，用 rPDL 表示，它包括三个步骤，即选择替换节点，选择幸存的块来恢复丢失的块和设计解码方案。如果除 R_f 外的所有源机架都是满的，我们称条带 S 为满，如果除 R_f 外的一些源机架是非满的，我们称条带 S 为非满。

1）选择替换节点。为了保证容忍一个机架的故障，我们提出了以下替换节点的选择方案。

• 如果条带 S 是满的，替换节点将从非源机架中随机选择。

• 否则，替换节点将从 R_f 以外的非满源机架中随机选择。

当 R_f 中的节点发生故障时，R_f 是所有条带的源机架，但其他任何一个机架都只是小部分条带的源机架。

由于机架的数量远远大于实际 DSSs 中擦除码的组别大小，所以条带的部分。

如果我们从包括 R_f 在内的随机选择的源机架中选择一个替换节点，那么 R_f 将遭受比其他机架更多的跨机架流量。所以我们不从 R_f 中选择替换节点。此外，我们的替换节点选择方案可以使跨架流量最小化。

2）选择存活的区块来恢复丢失的区块。请注意，我们的恢复算法不会从 R_f 读取幸存的数据来达到负载平衡（与上面的替换节点的选择方案类似）。除 R_f 外的所有源机架都将参与重建失败的块。

PDL 为采用多纠删码的分布式系统故障恢复提供了数据均匀分布和均衡访问的通用支持。更重要的是，与现有的随机数据分布相比，PDL 具有更短的降级读延迟、最小的跨机架重构流量、更好的跨机架重构读/跨机架重构写负载均衡和更高的恢复吞吐量等性能。在故障恢复窗口内，PDL 显著降低了前台用户请求的响应延迟。

4 总结

本文对分布式存储系统中纠删码容错技术的性能优化进行了分析讨论，具有容错能力且节约资源的分布式存储编码是大数据时代重点研究的核心技术之一，研究者们从不同角度对存储编码进行了改进，在吞吐量，延迟，存储容量，网络成本等不同方面进行了相当可观的优化。

参 考 文 献

[1] Ghemawat S,Gobioff H, Leung S T. The Google file system[C]. Proceedings of the l9th ACM Symposium on Operating Systems Principles.2003: 29-43

[2] Shvachko K ， Kuang H ， Radia S , et al. The Hadoop Distributed File System[C]// EEE Symposium on Mass Storage Systems & Technologies. IEEE，2010.

[3] Schroeder B，Gibson G A .Disk Failures in the Real World:WhatDoes an MTTF of 1,000,000 Hours Mean to You? (CMU-PDL-06-111)[J]. 2006.

[4] Shiyao Lin, Guowen Gong, Zhirong Shen, Patrick P. C. Lee, Jiwu Shu:

Boosting Full-Node Repair in Erasure-Coded Storage. USENIX Annual Technical Conference 2021: 641-655

[5] Liangliang Xu, Min Lv, Zhipeng Li, et al. PDL: A DatatowardsFastFailureRecoveryLayoutforErasure-codedDistributedStorageSystems//Proceedings of the the IEEE INFOCOM 2020-1EEEConference on Computer Communications.Toronto, ON.CanadaJuly 06-09 2020: 736-745

[6] Zizhong Wang, Tongliang Li, Haixia Wang, et al.CRaft:An Erasure-coding-supported Version of Raft for Reducing Storage Cost and Network Cost//Proceedingsof the 18th USENIX Conference on File and StorageTechnologies (FAST 20) , SantaClara,CAUSA,February 25-27,2020 :297-308

[7] Yuchong Hu, Liangfeng Cheng, Qiaori Yao,et al,Exploiting Combined Locality for Wide-Stripe Erasure Coding in Distributed Storage. FAST 2021: 233-248

[8] Yingdi Shan, Kang Chen, Tuoyu Gong,et al,Geometric Partitioning: Explore the Boundary of Optimal Erasure Code Repair. SOSP 2021: 457-471