分类号 _____ 学号 M201773132

学校代码 10487 密级 _____

华中科技大学

硕士学位论文

基于 FUSE 的 MBR 编码的并行化研究

学位申请人：徐熙豪

学 科 专 业：计算机技术

指 导 教 师：胡燏翀　副教授

答 辩 日 期：2019 年 5 月 27 日

A Thesis Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Engineering

Study on Parallelization of MBR Codes based on FUSE

Candidate : Xu Xihao

Major : Computer Technology

Supervisor : Assoc. Prof. Hu Yuchong

Huazhong University of Science and Technology

Wuhan, Hubei 430074, P. R. China

May 2019

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□，在____年解密后适用本授权书。

　　不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

摘 要

为了解决存储负载过高的问题，存储系统中引进了纠删码机制。纠删码能显著降低系统的存储负载，但是纠删码在修复失效节点时，会造成系统的网络带宽严重拥塞。因此，如何快速地读写数据、快速修复失效数据的同时尽量减小系统网络带宽消耗是一个亟待解决的问题。

针对上述问题，基于 FUSE 文件系统，设计并实现了支持分布式存储的 P-MBR 系统，实现了 MBR 编码的并行化。具体来说包括以下几个方面：（1）研究了 MBR 编码的并行化读性能，主要包括正常读和降级读两个场景，通过对每一个数据块采用分片技术，重新对 FUSE 的读文件函数进行封装，使得读文件时可以同时从两个结点以分片的方式读取数据块，从而来提升系统的读性能；（2）研究了 MBR 编码的并行化写性能，利用 MBR 编码的每一个块分布在两个不同节点上的特性，通过对 FUSE 的写文件函数进行封装，使得可以同时向两个节点写入数据块，以提升系统写入过程的并行度；（3）为了提升系统的修复性能，利用 MBR 编码布局，通过 Cache 层加速了文件块的读取和写入过程，此外还采用多线程技术提升了系统修复的并行度，从而来加速修复过程。

实验结果表明，MBR 编码在读写性能方面相较于 RAID 5、RAID 6 编码方式有小幅度提高；降级读、修复性能方面有极大的改善，降级读性能提升大约 70%，修复性能分别提升大约 40%和 60%，修复速度快且带宽开销较低。

关键词：FUSE　分布式存储　最小带宽再生码　并行化

Abstract

In order to solving the problem of high storage overload, erasure code is introduced in the storage system. Erasure code can significantly reduce the storage overload of the system, but the network bandwidth of the system will be severely congested when we use it to repair some failed node. Therefore, it is an urgent problem how to read and write data quickly, and repair failed node quickly while reducing the network traffic of the system.

To solve above problems, this thesis designs and implements a P-MBR system supporting distributed storage based on FUSE, and realizes the parallelization of MBR codes. Specifically, it includes the following aspects: (1) we study the parallel reading performance of MBR codes, mainly include normal read and degraded read; by using slice technology in every block, we rewrite the read function of FUSE，make it possible to read data block from two node by slice when reading file every time, thus to improve the system performance of read. (2) We study the parallel write performance of MBR codes. Taking advantage of the feature that each block of MBR codes is distributed on two different nodes, we rewrite the write function of FUSE, and make it possible to write data block to both nodes at the same time, so as to improve the parallelization of the writing process of the system. (3) In order to improve the system's repair performance, we speed up the process of reading data and writing data by cache layer and MBR codes layout; in addition, multithreading technology is also used to speed up the parallelization of system's repairing, thus improving repair process.

Experimental result show that the MBR codes has a small improvement in reading and writing performance compared with RAID 5 and RAID 6; a great improvement in degraded reading and repair performance, the degraded read performance is improved by about 70% and the repair performance is improved by about 40% and 60% respectively, a fast repair speed and a low network traffic.

Keywords: FUSE Distributed storage Minimum bandwidth regenerating codes Parallelization

目 录

1 绪论

本章主要是对全文做一个总体性的介绍，首先说明了本课题研究的背景和意义，继而对当前国内外的研究现状做了一个简要的说明，主要包括分布式存储系统的发展、分布式存储系统的容错机制、分布式容错恢复机制研究现状等。然后，概括了本文的主要工作，最后给出了全文的组织结构。

1.1 研究背景和意义

随着“云”时代的到来，各种电子设备都支持接入互联网，如电脑笔记本、手机、平板电脑、各种智能家居设备、智能穿戴设备等。当前，5G 的发展如火如荼，万物互联的时代已然即将来临。据研究机构 IHS Markit 估计[1]，2017 年全球大约有 203 亿个互联设备，而到了 2025 年末将会有超过 750 亿个互联设备,平均每个人有 9 个智能互联设备。相应地，为了支持各种电子设备的使用，数以亿计的应用程序以及 APP 被开发出来，服务于人们日常工作、学习、生活的方方面面。

海量的应用必将产生海量的数据 ，尤其是诸如微博、微信、QQ、Facebook、Twitter、Instagram 等社交媒体更是生成数据的主体。Domo’s 在报告《Data Never Sleeps 5.0 Report》中指出了 2017 年各大主流应用每分钟所产生的数据量，Twitter 上每分钟会发送 45.6 万条推特，Instagram 用户每分钟会分享大约 4.67 万张图片，Facebook 上每分钟大约会有 51 万条评论以及 29.3 万条状态更新[1]。Jim Gray 曾指出，在网络环境中，每 18 个月产生的数据量就会等于有史以来产生的数据量总和[2]。如图 1-1 所示，据 statist 网站统计[2]，2005 年全球所产生的数据总量为 0.1ZB，2015 年这一数字为 12ZB，而到了 2025 年据估计全球将会有 163ZB 数据[3]。

海量数据的背后离不开一个强大的存储系统的支持，为了适应这一发展，分布

[1] Bernard Marr. How Much Data Do We Create Every Day. https://www.forbes.com/sites/bernardmarr/

[2] Statista. https://www.statista.com/statistics/871513/worldwide-data-created/, 2018

式存储系统得到了快速应用。为了存储海量数据，分布式存储系统通常将数据分散存储在不同的存储节点上，很多节点甚至在地理位置上也是不同的。同时，为了节省存储成本，诸多企业均是在廉价的商用磁盘上对数据进行存储。比如，Google 的分布式结构化存储系统 BigTable，旨在成千上万台商用服务器上管理 PB 级的结构化的数据。Google 的诸多项目，如网页索引、谷歌地图、谷歌财经等均是使用 BigTable 来进行存储[4]。BigTable 的底层是 GFS 分布式文件系统[4-5]。两者主要区别在于，GFS 分布式文件系统主要是直接面向大型分布式的密集型的数据应用，BigTable 主要是处理结构化的数据，BigTable 相当于传统的数据库所扮演的角色。

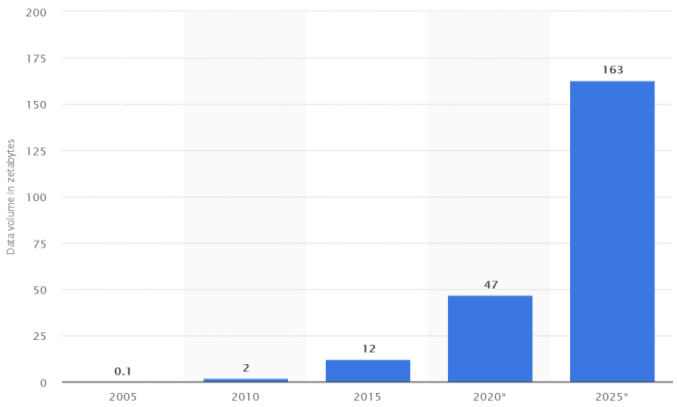


图 1-1 全球数据量变化趋势

文献[5]中提到，GFS 所部属的最大的一个集群中有数千台服务器，包含数千块磁盘，可以提供数百 TB 的存储空间，可同时服务于数百个客户机。由于 GFS 中包含几百甚至几千台普通的商用廉价磁盘，并且几乎同时为同等数量的客户机提供服务，势必导致在任何给定时间内，都有某些磁盘可能无法工作。文献[6-7]中也提及，Facebook 于 2010 年所部属的某个 Hadoop 集群中，有 2000 台存储容量为 12TB 的机器，总共大约有 21PB 的总存储容量，如此大的存储容量也导致集群中每天都会有存储设备失效。

因此，集群中存储设备失效是常态。那么，如何快速而有效地恢复出失效数据将变得非常有意义，同时还要尽量在恢复数据时消耗较小的带宽,这也是当前的研究热点[8-11]。因此，本文试图从编码的并行化角度对分布式存储系统读写性能、修复性

能进行研究，以提升系统的读写性能、修复性能。

1.2 国内外研究现状

1.2.1 容错机制

分布式存储系统中，由于节点数量庞大，多使用廉价的商用磁盘作为存储介质以降低成本。然而，正因为磁盘质量不可靠，再加上软件或者程序故障等，数据失效是常态。因此，在数据失效时如何快速恢复失效数据是一个热点问题。常见的容错机制有副本机制和纠删码机制。

（1） 副本

副本机制也称为复制机制，是容错机制中最容易实现的，使用也很广泛。诸如HDFS[12][13]、Ceph[14-15]、Swift[16]文件系统中都有使用。GFS、HDFS 、Swift 都采用了三副本的机制，即每一份数据会存储三份。以HDFS 为例，将原始文件划分为64MB或者 128MB 大小的文件块存储在不同的三个数据节点上。这样一来，任何一个数据块失效时，都可以通过另外的两个数据节点上的副本进行修复，除非三个数据节点均失效，否则我们总能修复失效的数据。此时，系统在任意时刻可以容忍 2 个数据节点失效。

副本机制具有实现简单、计算资源占用小、文件块存取方便等优点。然而，对于任何一个采用副本机制的系统，存储负载的开销庞大。我们假定一个存储系统采用 N 副本机制，那么存储系统中实际存储的数据只有 1/N, 其余的均为冗余数据。在存储系统中数据量较小的情况下，比如几百至几千 TB 级，很多企业依然能负担得起这个存储代价。但是当存储的数据量变得庞大时，比如 PB 甚至 ZB 级时，存储成本就格外的庞大了。

（2） 纠删码

针对副本机制存储代价大的问题，人们开始在存储系统中广泛使用纠删码机制来保证系统的可靠性，因为纠删码机制在保证与副本机制同样地容错能力的情形下，具有最低的存储开销。纠删码的主要思想是将原始文件通过纠删码算法来编码形成校验数据，然后将原始文件和校验数据一起存储起来，以达到容错的效果。具体来

说，纠删码机制下，将一个大小为 M 的原文件平均分为 k 个数据块，然后通过编码算法，将这 k 个数据块进行编码形成 n-k 个校验块，最终会得到 n 个编码块，存储至 n 个不同的节点上形成一个条带[17-18]。对于 n 个编码块中的任意 n-k 个块失效，均可以通过同一条带中余下的任意 k 个块解码进行恢复。

实际的存储系统中所用的纠删码通常会尽量满足 MDS（Maximum Distance Separable）性质。因为 MDS 码能保证同样的容错级别下，具有最小的存储负载。常用的纠删码是 RS[19] 码，比如 HDFS-RAID[20] 中使用的即是 RS(14, 10)的编码形式，RS 编码的构造方式主要有范德蒙矩阵以及柯西矩阵。图 1-2 中给出了 RS 码编解码的一个简单示例。

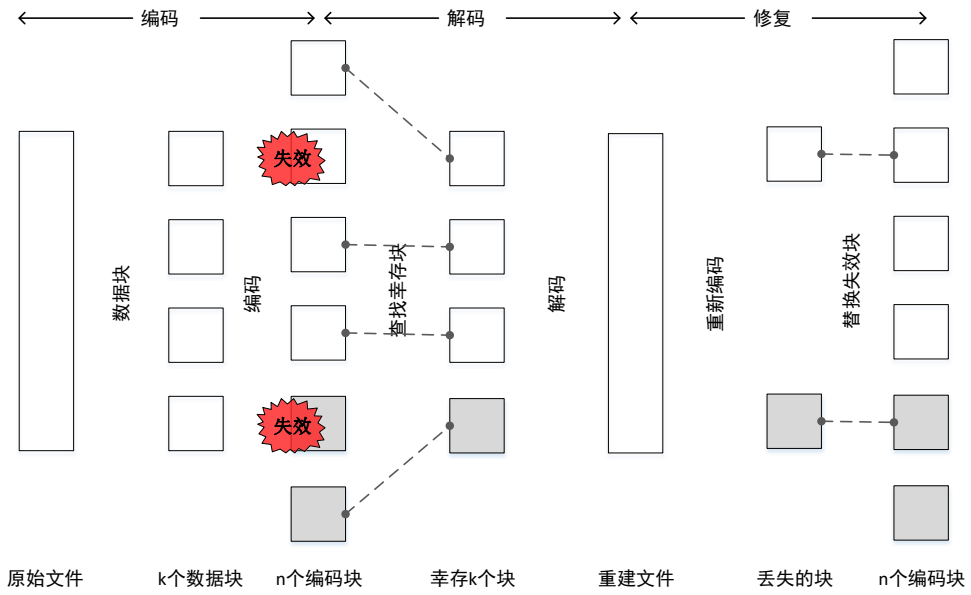


图 1-2 RS (n, k) 码的编解码过程

1.2.3 容错修复机制研究现状

针对三副本机制存储负载过高，基于 MDS 编码方案的纠删码虽然存储负载低，但是修复成本高的问题，Cheng Huang 等人在文献[21]中提出了 Pyramid 编码方案，以谋求存储空间和访问效率的平衡。文中设计了两种弹性纠删码方案，一种是基本金字塔编码（BPC），另一种是广义金字塔编码（GPC），相比基于 MDS 的弹性纠删码，这两种方案虽然稍微多一些存储开销，但是可以明显提高重构性能。此外，文

中还建立了一个必要的匹配条件来表征故障恢复的极限，所有满足匹配条件的失效情形均可以进行恢复。此外，文献[8]中还介绍了一种并行局部修复码，可以减少修复时间。

Katina Kralevska 等人在文献[22]中提出了一种名为 HashTag 编码（HTECs）的再生码，该编码方案达到了最优存储容量-修复带宽平衡曲线的 MSR 点，即 HTECs 编码的存储负载是最优的。与 RS 编码、Piggyback 编码相比，HTECs 编码在单节点失效情形下，可以分别节省 60%、30%的修复带宽。此外，文中还详细阐述了修复带宽、I/O 以及修复时间之间的关系，深入分析了 HTECs 编码的修复过程，证明了单节点修复过程是线性的并且高度并行化。文献[23]中也提出了一种 MSR 编码，名为 Butterfly，旨在降低修复带宽，提高系统修复性能。

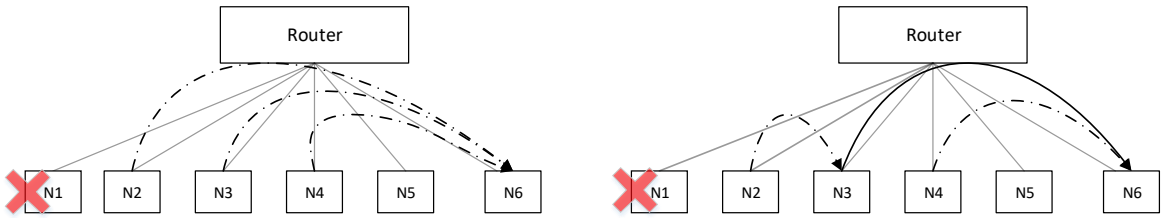
可以看到，分布式存储系统中的容错修复机制，正从早期的副本机制、传统的纠删码机制，开始转向基于网络编码的纠删码方案。许多研究者，谋求在存储开销和系统修复带宽之间找到平衡。

1.2.3 编码的并行化研究现状

分布式存储系统中经常要面对的一个问题就是系统的容错性，因此人们想到用数据冗余的方法来提高系统的可靠性。从最初的副本机制和传统的纠删码机制到如今的再生码，均在分布式存储系统中得到了广泛应用。许多学者发现将编码与分布式存储系统相结合，确实能提高系统的可靠性。但是随着可靠性问题的解决，学者们又开始谋求在修复速度、修复带宽、计算复杂度等方面谋求最优的结果。

副本最初是为了提高系统的可靠性，而被分布式存储系统广泛采用，但是副本的问题在于极大的增加了系统的存储负载。为了减小系统中的存储负载，传统的纠删码方式开始在分布式存储系统中应用，但传统的纠删码也带来了单节点修复带宽过高的问题。由于单节点失效在分布式存储系统中是很常见的，因而人们又开始希望能减小单节点修复带宽、提高修复速度。基于上述原因，各种新的纠删码以及基于网络编码的再生码层出不穷，有的为谋求最小化存储负载，有的为谋求减小计算复杂度，有的为谋求减小网络带宽消耗。

最近的一些研究中，有不少学者从并行化的角度设计了编码方案或者并行化机制来提升系统的修复性能。文献[8]中，Subrata Mitra 等人提出了一种面向纠删码存储系统的分布式重建技术，名为并行局部修复（PPR,Partial-Parallel-Repair）。一个简单的示意图如图 1-3 所示，图 1-3(a)表示传统的 RS(5,2)重建方法，图 1-3(b)表示采用 PPR 重建失效节点的方法。假设有 3 个数据块通过 RS 编码方式形成 5 个编



(a) 传统的RS(5,2)重建　　(b) PPR重建

图 1-3 传统的 RS 重建和 PPR 重建

码块，由 RS（5,2）编码的性质可知，当有一个块失效时，剩下的 4 个编码块中的任意 3 个均可以恢复出失效块。

当 N1 节点失效时，本例（图 1-3）中传统的纠删码的做法是，节点 N2、N3、N4 通过路由将各自的编码块传送至节点 N6 上，从而恢复出失效块。这种做法会导致节点 N6 所连接的链路成为瓶颈，因为节点 N2、N3、N4 会同时向节点 N6 传输数据块，从而会导致修复时间变慢，并且节点 N6 所连接的链路带宽会成为整个系统的最大带宽。但是其他节点间的链路却很空闲，带宽资源没有得到充分利用。

PPR 的做法是采用局部并行修复的方法，将整个修复过程分为两个时隙，第一个时隙如图 1-3（b）中虚线箭头所示，第二个时隙如图 1-3（b）中的实线箭头所示。通过将修复过程分为两个时隙，缓解了节点 N6 所在链路的拥塞。通过实验验证，PPR 技术用于 RS 编码、LRC 编码、Rotated RS 编码上时，均能显著降低系统的修复时间以及系统的最大网络带宽要求。

文献[11]中，Li Runhui 等人通过设计了一种修复流水线技术（Repair Pipelining），加速了单节点修复过程以及降级读过程。Pipelining 技术的主要思想在于将每一个编码块分成粒度更小的片，每个块中相同偏移的片形成一个条带，相当于有两个维度的条带，一个是块级的，另一个是分片级的。最终通过不同节点中的同一块级条带

上的不同分片并行地传输编码数据从而加速修复过程以及降级读过程。将 Pipelining 技术用于采用 RS 编码的 HDFS 以及 QFS 系统中时，节点修复时间也显著降低了。

文献[24]中 Jun Li 等人提出了一种名为 Carousel codes 的编码方式，能提高数据读取的并行度，从而节省 hadoop 任务时间。具体来说，Carousel codes 是在 RS 码的基础上进行优化的，传统的 RS 编码方式下，同一条带上数据块与校验块分别存放在不同的节点上，因而在读取数据块时只会从存储了数据块的节点上读取，存储了校验块的节点并没有参与这一过程。Carousel codes 通过矩阵变换的方式，使得同一条带上的校验数据均匀的分布在每一个节点上，这样一来每个节点上既有原数据块上的数据，又有了校验数据，从而使得所有节点都可以参与读过程。此外，作者还基于 MSR codes 扩展了 Carousel codes，也在一定程度上减小了系统的修复带宽。

另一方面，Carousel codes 还存在着问题：Carousel codes 未考虑系统的异构性，所有的节点上源文件分布是一致的，但实际的分布式存储系统中不同的服务器性能也是不同的。为此，Jun Li 等人又在文献[25]中提出了 Galloper codes 的编码方案，Galloper codes 是基于局部修复码（LRC, Locally repairable code）设计的。首先，Galloper codes 考虑了系统中服务器的异构性，根据各服务器性能的不同，源文件在各节点上的分布也不同。其次，基于局部修复码的设计，可以有效降低磁盘 I/O，加速修复时间。最后，Galloper codes 也扩展了局部修复码的并行度，因为局部修复码中编码块和数据块也是分开存放于不同的节点上的，从而也提升了系统的读性能。

1.3 研究内容和目标

在分布式存储系统中，更快地读写速度和修复速度是人们一直所追求的。因此，本文试图从并行化地角度对这几个方面进行讨论。

为了实现 MBR 编码的并行化，本文以 NCFS[26]文件系统为基础，设计并实现了 P-MBR 系统。该系统是基于 FUSE 进行开发的，可以支持不同的编码方案、 不同的存储设备，主要开发语言是 C++。P-MBR 系统分为四层——文件系统层、编码层、缓存层、存储层。文件系统层主要处理文件读写、删除、创建目录、获取文件属性、初始化文件系统等基本操作。编码层主要是实现各种编码方式，比如本文实现的 MBR

编码、RAID 5、RAID 6 编码，主要包括编码、解码等操作。缓存层主要是用来加速文件读写、修复过程。存储层主要是为文件系统访问不同的存储节点提供一个统一的接口。此外，还有一个专门的修复工具用来处理文件的修复请求。

具体来说，主要包括以下三个方面的工作：

（1）研究了 MBR 编码的并行化读性能，主要包括正常读和降级读两个场景，通过对每一个数据块采用分片技术，重新对 FUSE 的读文件函数进行封装，使得读文件时可以同时从两个结点以分片的方式读取数据块，从而来提升系统的读性能；

（2）研究了 MBR 编码的并行化写性能，利用 MBR 编码的每一个块分布在两个不同节点上的特性，通过对 FUSE 的写文件函数进行封装，使得可以同时向两个节点写入数据块；

（3）为了提升系统的修复性能（包括更快的修复速度以及更小的修复带宽两方面），利用了 MBR 编码布局，通过 Cache 层加速文件块的读取，此外还采用多线程技术来加速修复过程；

1.4 本文组织结构

全文共分为六章，主要内容组织如下：

第一章主要介绍了 MBR 编码并行化研究的背景和意义，分析了国内外在容错机制以及编码并行化研究方面的研究现状，阐述了本文的主要研究内容和目标；

第二章主要对 MBR 编码的相关理论进行了分析，包括 MBR 编码主要解决的问题，最小修复带宽理论分析等；

第三章详细阐明了 P-MBR 系统以及 MBR 编码并行化设计，从系统总体设计、文件系统层设计、编码层设计、缓存层设计、存储层设计等 5 个维度进行了分析，阐述了 MBR 编码的并行化设计，从文件读写、文件修复方面给出了并行化的方案；

第四章从实际应用的角度给出了 MBR 编码的并行化实现，包括并行化读写，并行化修复；

第五章对系统进行了测试，并对实验结果进行评估；

第六章主要总结了本文的主要内容和成果，展望了进一步的工作。

2 MBR 编码的理论分析

本章首先阐明了 MBR 编码要解决的问题。MBR 编码发源于网络编码，因此紧接着对网络编码进行了简要分析。然后，对 MBR 编码的修复带宽进行了理论上的分析，最终引出了对 MBR 编码的构造方法，详细分析了本系统中 MBR 编码是如何构造的，包括编码策略、数据重建、结点修复等三方面。

2.1 MBR 编码主要解决的问题

提到 MBR 编码，始终绕不开纠删码。图 2-1 中给出了一些经典的编码的发展脉络[27],最初是传统的纠删码，比如 RS 码等。在第一章的容错机制中介绍过，存储系统中最初是通过副本机制来维持系统的冗余性，从而保证系统的可靠性。

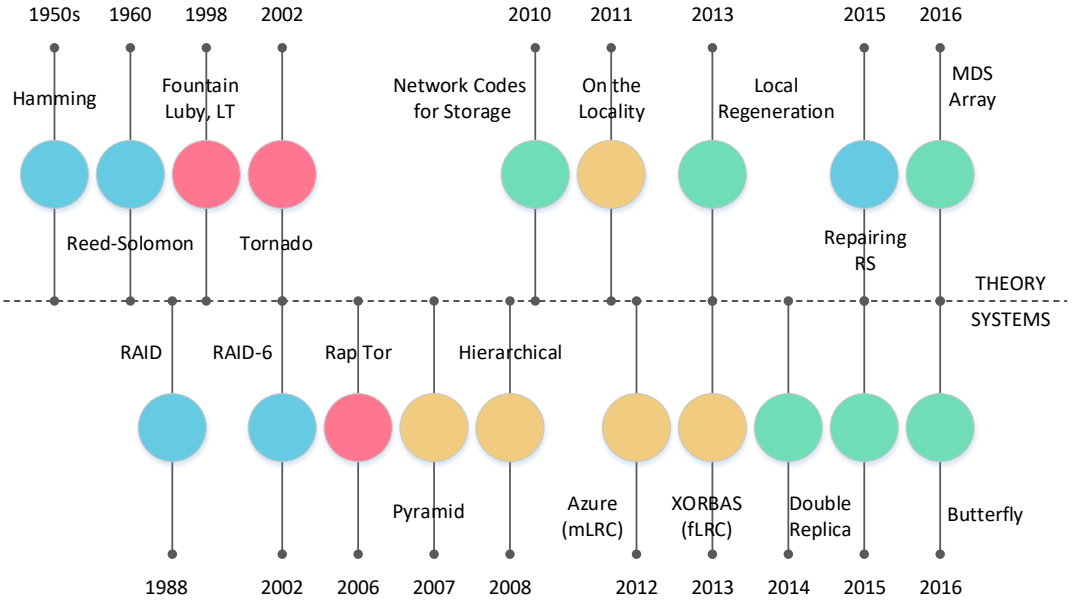


图 2-1 分布式存储系统中纠删码的发展脉络

副本机制的弊端在于：当分布式存储系统中数据量很大时，系统的存储成本会特别高。因此，为了降低存储成本，人们开始在分布式存储系统中引入了纠删码机制。纠删码机制与副本机制本质上是存储负载与系统可靠性的权衡。以 HDFS 与 HDFS-RAID 为例，HDFS 采用三副本机制，可以同时容忍 2 个节点出现故障，由于

原文件中每一个数据块存储了三份，因而存储负载是 3（假定存储系统中刚好只存储原文件时存储负载为 1）。至于 HDFS-RAID，是经典的 RS(14，10)编码，即对原文件以条带的形式进行存储，每 10 个数据块就会编码形成 4 个校验块，总共有 14 个编码块，因而存储成本是 14/10=1.4，同时该情形下系统可以容忍任意 4 个节点出现故障。因此，分布式存储系统中引入纠删码机制可以大大降低存储系统的负载。

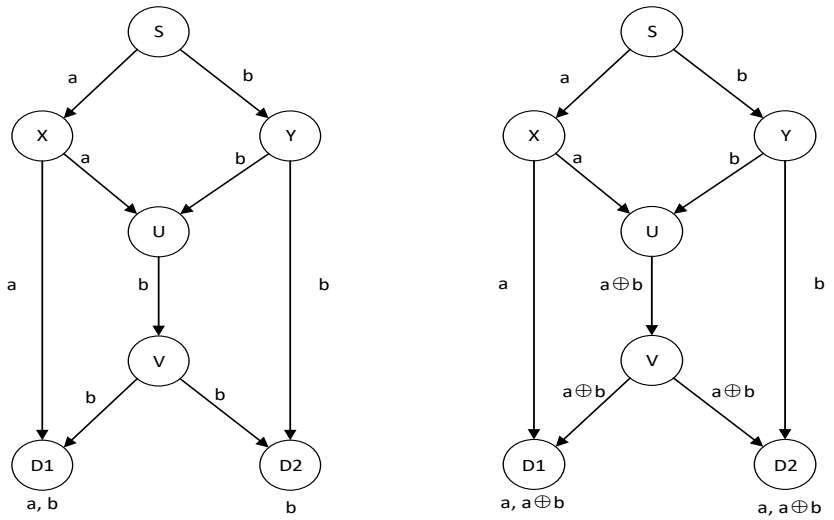
然而，传统的纠删码机制虽然降低了系统的存储负载，却带来了节点修复方面的问题。比如 RS（14, 10）编码，当仅有一个节点失效时，我们需要传输其他的 10 个块到新的节点上来解码生成失效的那一个块，也就是说相对于副本机制而言，所需要的网络带宽变为原来的 10 倍。 为了减小修复过程中的带宽，再生码开始在分布式存储系统中广泛应用，图 2-1 中所示的 Double Replica、Butterfly 等编码均属于再生码。再生码是在网络编码的基础上得以发展并形成的，而 MBR 编码正是一类特殊的再生码，MBR 编码在修复失效数据时，所需要的修复带宽最小。因此，MBR 编码主要解决的问题就是如何减小系统的修复带宽。

2.2 网络编码

信息论的创立者香龙最早在文献[28]中指出了网络有向图模型的最小割决定了通信网络中端对端的最大信息流。传统的多播网络中，中间节点对于信源节点所发送的数据仅仅是进行存储转发，并没有做额外的处理。因此，香龙所提出的理论吞吐量上限在实际的网络模型中一直没有实现。直到 2000 年左右，Rudolf Ahlswede 等人在文献[29]中提出了网络编码的概念，首次创造性地提出了中间节点对源数据进行编码并转发的思想。它的主要贡献在于，突破了人们关于中间节点对于接收到的数据进行处理没有意义的认知，从而使得多播网络中的网络容量达到最大流最小割理论（Max-flow Min-cut Theorem）中的阈值成为可能。

根据网络编码理论，中间节点相当于先对接收到的数据进行编码，接着将处理后的数据转发给后续节点，也就是说中间节点具备了编码和存储转发的双重特性[30]。根据最大流最小割定理，数据传输链路的最大通信量不能超过发送方与接收方之间的最大流值，而通过网络编码可以达到这一最大流值。

下面，以图 2-2 为例来说明使用网络编码的多播网络环境中，多播路由网络传输的最大流值是如何取得的，其中图 2-2（a）是传统的采用路由的多播网络，图 2-2（b）是采用了网络编码的多播网络。节点 S 是源节点，节点 D1、D2 是信宿节点（汇聚节点），余下的节点是中间节点。假设图 2-2 中的每条传输链路的容量为单位 1，源数据 a 和 b 的数据量大小也为 1，则每条链路刚好可以传输一个数据块，理论上该网络的最大传输容量为 2，即信宿节点 D1、D2 可以同时接收到数据块 a、b。



（a）传统的多播网络　　（b）具备网络编码的多播网络

图 2-2 单源二信宿网络

在图 2-2（a）所示的传统的多播路由中，源节点 S 分别向中间节点 X、Y 转发数据块 a 和 b。经过节点 X、Y 的存储转发，中间节点 U 接收到数据块 a 和 b 后，只能向后继节点传输数据块 a 或 b 中的一个，因为传统的多播路由网络中的中间节点不具备编码能力。图 2-2（a）演示的是中间节点 U 向中间节点 V 转发数据块 b 的情形，那么最终信宿节点 D1 同时接收到了数据块 a, b，节点 D2 仅接收到了数据块 b。同理可知，当中间节点 U 转发数据块 a 时，则信宿节点 D1 只能接收到数据块 a，信宿节点 D2 能同时接收到数据块 a, b。也就是，在图 2-2（a）所示的传统多播路由中，每次只有一个信宿节点能同时接收到数据块 a、b，而另外的一个信宿节点只能接收到其中的一个数据块 a 或 b，此时信宿节点所在链路的平均吞吐量为（1+2）/2=1.5。

在图 2-2（b）所示的网络编码的多播网络中，同样是从源节点 S 发送数据块 a

和 b。但是，由于中间节点具备了编码能力，节点 U 会对接收到的数据块 a、b 进行异或形成加工处理后的数据块 a⊕b，然后继续向后继节点转发。最终，信宿节点 D1 接收到了数据块 a 和 a⊕b，信宿节点 D2 接收到了数据块 b 和 a⊕b。在节点 D1 上通过 a⊕(a⊕b)可以得到数据块 b，同理在节点 D2 上通过 b⊕(a⊕b)可以得到数据块 a。这样一来，信宿节点都同时接收到了源节点 S 发送的数据块 a 和 b，此时信宿节点的平均吞吐量为（2+2）/2=2。因此，通过网络编码的形式，多播网络中可以达到网络容量的理论阈值。

2.3 MBR 编码的修复带宽分析

2.3.1 再生码

Alexandros G. Dimakis 等人于 2010 年在文献[31]中基于网络编码的理论提出了再生码的概念。Dimakis 等人首先提出了一个修复性问题：在经典的（4, 2）MDS 纠删码情形下，原文件通过编码形成了 4 个段，存储在 4 个节点上（x1, … x4 ），如果有一个节点失效了（假定为 x4），那么通过连接 x1, x2, x3 三个节点在新节点 x5 上生成新的编码段（注意：新生成的编码段加上原有的 3 个编码段仍需满足 MDS 性质），最小的通信量是多少？

我们知道，满足（4, 2）MDS 的纠删码，可以容忍 2 个段（块）失效，并且当任何一个段失效时，通过连接其他的任意 2 个节点上的段，可以恢复出失效的段。在再生码概念产生之前，人们普遍认为采用（n, k）MDS 编码，相比于副本会导致 k 倍的带宽消耗，并且这一缺点是无法避免的。以（4, 2）MDS 编码为例，假设原文件大小为 M=2MB，则每一个段的大小为 2/2=1MB。因此，按照传统的纠删码概念来看，节点 x4 失效后，需要连接其他的 2 个段来进行恢复，也就是说整个网络的通信带宽是 2MB。那么我们能否减少这一带宽消耗呢？答案是肯定的。

下面我们通过图 2-3 中的（4, 2）编码的信息流图来进行说明。图 2-3 中的信息流图是一个有向无环图，主要由三种节点构成，源数据节点 S，汇聚节点 DC，以及存储节点 i（存储节点 i 在图中由存储输入节点 X_{in}^{i} 和存储输出节点 X_{out}^{i} 表示）。其中，

源节点到存储节点和存储节点到汇聚节点间的有向边的通信容量均为无穷大，存储节点 X_{in}^{i} 到 X_{out}^{i} 之间的通信容量等于每个存储节点所存储的数据量 α 的大小，在本例中为 1，即有 α =1。在再生码方案下，当节点 x4 失效时，可以通过 x1，x2，x3 节点分别传送 β 大小的数据到新的节点 x5 上来生成新的编码段，只要我们能保证通过 x1，x2，x3，x5 中的任意两个节点可以恢复出原始数据即可。

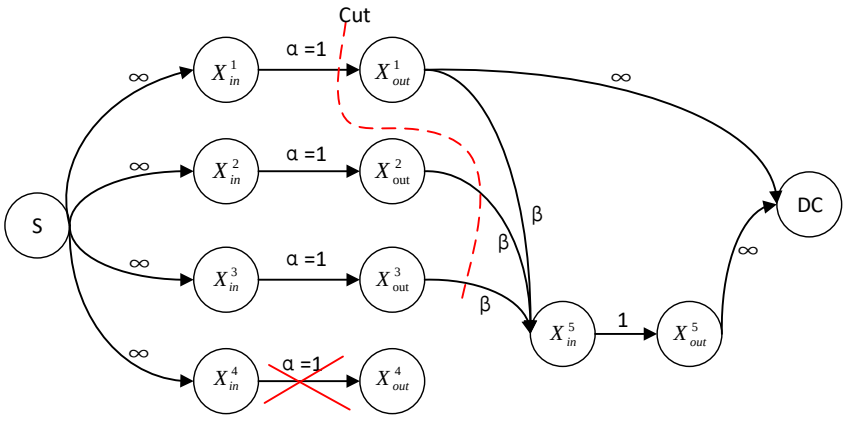


图 2-3 采用（4, 2）编码的信息流图

根据最大流最小割定理，为了能最终在 DC 节点上重建出原文件，分离源节点 S 和汇聚节点 DC 的最小割必须大于原文件的大小 M，即有 1+2 β ≥2，则 β ≥0.5。也就是说我们只需要传输 3 β =1.5MB 的数据到节点 x5 上即可重建出原文件，Dimakis 等人在文献[31]中已经给出了该示例的详细说明。

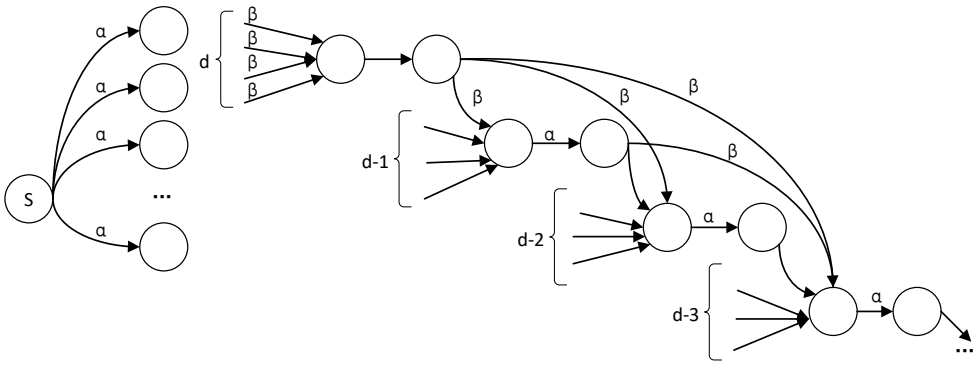


图 2-4 再生码修复信息流图的一般情形

图 2-4 中将上述信息流图推广到了一般情形。假设原文件大小为 M，对原文件

进行编码形成 n 个大小为 α 的块，分布在 n 个节点上。再生码满足 n 个节点中的任意 k 个可以恢复出原文件这一性质，故 α =M/k。当系统中有节点失效时，通过连接幸存的任意 d（d≤n-1）个节点在新节点上生成新的编码块后，原文件仍然可以被恢复。此时，每个节点需要传输的数据量为 β，总修复带宽 γ =d β 。当有新的节点失效时，重复这一过程即可，最终使得系统总是满足纠删码的（n, k）性质。

2.3.2 MBR 编码带宽消耗理论分析

通过最大流最小割定理来分析信息流图，Dimakis 等人给出了如下定理[31]：

定理 2-1：对于 $\forall \alpha \geq \alpha^*(n,k,d,\gamma)$，点 (n,k,d,α,γ) 是可行的并且可以在线性网络编码中实现。在信息理论上 $\alpha < \alpha^*(n,k,d,\gamma)$ 的点是不可实现的。$\alpha^*(n,k,d,\gamma)$ 的阈值函数如下:

$$\alpha^*(n,k,d,\gamma)=\begin{cases}\dfrac{M}{k}, & \gamma \in [f(0),+\infty) \\ \dfrac{M-g(i)\gamma}{k-i}, & \gamma \in [f(i),f(i-1))\end{cases} \tag{2-1}$$

其中，

$$\mathrm{f}(i) \triangleq \frac{2Md}{(2k-i-1)i+2k(d-k+1)} \tag{2-2}$$

$$\mathrm{g}(i) \triangleq \frac{(2d-2k+i+1)i}{2d} \tag{2-3}$$

其中 d≤n-1。对于任意给定的 d，n，k，由式（2-2）可以进一步得到：

$$\gamma_{\min} = f(k-1) = \frac{2Md}{2kd-k^2+k} \tag{2-4}$$

通过（2-4）式可以知道，修复带宽 γ =d β 是随着参与修复过程的节点数量 d 的增大而减小的。也就是说，当在新的节点上生成新的编码块时，参与修复的节点数量 d 的增加幅度慢，每个修复节点所传输的数据量 β 的减小幅度要快，最终使得 γ

=d β 是减小的。因此，当 d=n-1 时，系统的修复带宽 γ 最小。

对于满足（n, k）性质的再生码，根据最大流最小割定理，原文件的大小 M 必须小于等于网络信息流图中的最小割才能恢复出原文件。推广至一般情形，对于给定参数(n, k, d, α, β)，则有：

$$M \leq \sum_{i=0}^{k-1} \min\{\alpha, (d-i)\beta\} \tag{2-5}$$

当（2-5）式中等号成立时，则该编码是最优的。

根据定理 2-1 以及 2-5 式，对于确定的 n、k、d、M，可以得到关于 α 和 β 不同取值时的最优存储容量-修复带宽平衡曲线，如图 2-5 所示。Dimakis 等人将满足该存储容量-修复带宽平衡曲线的编码称之为再生码。

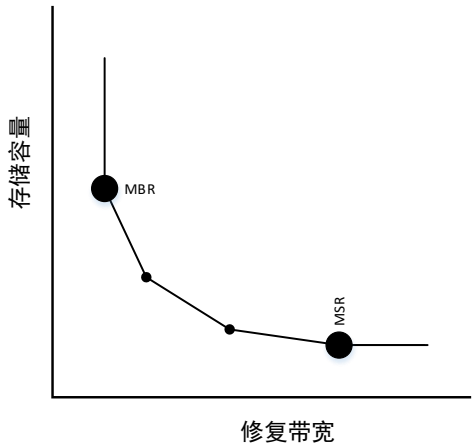


图 2-5 存储-修复带宽平衡曲线

最优存储容量-修复带宽平衡曲线的两端各有一个极值点，分别对应着最优的存储效率以及最小的修复带宽，这两个点对应着两种特殊的再生码——最小存储再生码（MSR, Minimum-Storage Regenerating Codes）以及最小带宽再生码（MBR, Minimum-Bandwidth Regenerating Codes）。

根据定理 2-1，在 MSR 点有最小的存储消耗，此时对应的 α 、γ 取值为：

$$(\alpha_{MSR}, \gamma_{MSR}) = \left(\frac{M}{k}, \frac{Md}{k(d-k+1)}\right) \tag{2-6}$$

在 MBR 点有最最小的修复带宽，此时对应的 α 、γ 取值为：

$$(\alpha_{MBR}, \gamma_{MBR}) = \left(\frac{2Md}{2kd - k^2 + k}, \frac{2Md}{2kd - k^2 + k} \right) \quad (2\text{-}7)$$

对于最小存储再生码（MSR），我们知道它仍然满足（n，k）MDS 性质，如果我们令 d=k，此时刚好是传统的纠删码的修复方法，修复带宽 γ =Md/(k(d-k+1))=M，即修复带宽为原文件大小。通过（2-6）式可知，修复带宽 γ_{MSR} 是关于 d 的递减函数。因此，当 d 取得最大值时，即 d=n-1 时修复带宽 γ_{MSR} 可以取得最小值。将 d=n-1 代入（2-6）式则有：

$$f(i) \triangleq \frac{2Md}{(2k - i - 1)i + 2k(d - k + 1)} \quad (2\text{-}8)$$

很显然，最小修复带宽 γ_{MSR}^{min} 刚好是每个存储节点所存储数据量的（n-1）/(n-k) 倍，这是对传统的 MDS 码修复方法的扩展，此时最小存储再生码（MSR）仍然满足 MDS 性质，修复带宽和存储容量取得了最优平衡。

对于最小带宽再生码（MBR），由（2-7）式可知，节点上存储的数据量 α 等于修复带宽 γ 。同样地，γ_{MBR} 是关于参与修复的结点数量 d 的递减函数，因此 d 取最大值 n-1 时，γ_{MBR} 有最小值，此时 α_{MBR} 也是最小的。将 d=n-1 代入（2-7）式则有：

$$(\alpha_{MBR}^{min}, \gamma_{MBR}^{min}) = \left(\frac{M}{k}\bullet\frac{2n-2}{2n-k-1}, \frac{M}{k}\bullet\frac{2n-2}{2n-k-1} \right) \quad (2\text{-}9)$$

显而易见，最小存储再生码（MBR）的最小修复带宽 γ_{MBR}^{min} 始终等于节点存储的数据量 α_{MBR}^{min}，因此采用 MBR 编码修复时，不会导致额外的修复带宽。当 n、k、d、M 相同时，结合式（2-8）和（2-9），通过不等式法可以很容易地证明 $\gamma_{MBR}^{min} < \gamma_{MSR}^{min}$ 也就是说，相比较于 MSR 编码，MBR 编码有着更小的修复带宽。

当然，相较于 MSR 编码在存储效率方面的优势，MBR 编码也存在着劣势。MBR 编码的缺陷在于，每个节点需要存储的数据量是 MSR 编码情况下的（2n-2）/（2n-k-1）倍。就给定冗余情况下的可靠性而言，MBR 编码不是最优的，但存储效率方面不是本课题的首要目的。本课题主要想探讨 MBR 编码在降低修复带宽方面的效果，在当

下网络严重拥塞的情况下，如何有效降低带宽消耗是很有意义的。而且，相较于副本的冗余机制，MBR 编码还是极大的降低了存储负载。

2.4 MBR 编码的构造方法

Dimakis 等人提出了最小存储再生码（MBR）的概念，但是并未给出具体的 MBR 编码的策略。比较经典的 MBR 编码的构造方法主要有两种[32-33]，第一种是基于完全图的 MBR 编码构造[32]，第二种是基于乘积矩阵的 MBR 编码构造方法[33]，主要提出者都是 K. V. Rashimi 等人。

第一种经典的 MBR 构造方法是基于完全图中点与边的关系进行编码[32]，在修复时无需解码操作，称之为完全图 MBR 编码（Polygonal MBR Codes）。本课题的 MBR 编码正是基于该方法实现的，因此下面将以图 2-6 中的示例来详细介绍完全图 MBR 编码构造方法。

图 2-6 的示例中各参数值如下（各参数定义参考 2.3.1 小节）：n=5，k=3，d=4，β=1，原文件大小 M=kdβ-（k/2）β=9，本例中假设每个数据块大小为 1。下面将从编码策略、数据重建、节点修复等 3 个方面来展开。

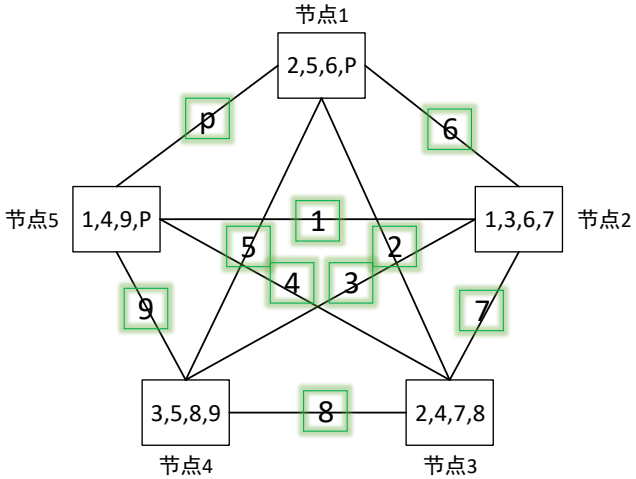


图 2-6 完全图 MBR 编码构造图例

（a）编码策略：首先，我们构造一个包含 5 个结点 10 条边的完全图。9 个数据块（1…9）使用（10,9）MDS 编码形成 10 个编码块，其中包含 9 个源数据块和一个校验块 P。每个编码块对应完全图中唯一的一条边，每个节点存储与其相连的边对应

的编码块，本例中各节点所存储的编码块如图 2-6 所示。

（b）数据重建：从图例来看，由于任意 3 个存储节点中均包含不同的 9 个编码块，根据（10,9）MDS 性质，理论上任意 3 个节点均可以恢复出原文件。不过需要注意的是，当只有 3 节点时，必须通过传统的（10,9）MDS 解码来进行重建原文件，此时不满足 d=n-1=4

（c）节点修复：当任意一个节点失效时，新的节点可以从幸存的 4 个节点中下载编码块，这些编码块对应于失效节点所连接的边。比如示例中节点 5 失效时，假设要在新的节点 6 上恢复节点 5，则节点 1 传输编码块 P，节点 2 传输编码块 1，节点 3 传输编码块 4，节点 4 传输编码块 9，最终在节点 6 上就可以恢复出节点 5。在一个节点失效的情况下，系统无需解码即可修复失效节点，显著降低了系统中网络带宽的消耗。

2.5 本章小结

本章主要是从理论上分析 MBR 编码带宽消耗以及构造方法，为后文的工作打下基础。本章首先分析了 MBR 编码主要解决的问题。然后，在 2.2 小节简要介绍了网络编码的概念，接着详细分析了节点失效时 MBR 编码的修复带宽，通过对网络编码以及再生码的阐述，指出了 MBR 编码是一种特殊的再生码——在修复失效节点时带宽消耗小，最后给出了本文中 MBR 编码的构造方法。

3 P-MBR 系统及 MBR 编码并行化设计

为了对 MBR 编码的并行化进行研究，需要有一个较好的系统平台作为支撑。本章主要是对基于 FUSE[34-35]平台开发的 P-MBR 系统以及 MBR 编码的并行化从设计层面做出了说明。首先概述了系统的设计动机以及总体架构，并且给出了系统的实现功能和目标；接着，详细介绍了各子模块的设计；最后对 MBR 编码的并行化从读、写、修复等三个方面给出了设计方案。

3.1 MBR 编码并行化的研究动机

对于 MBR 的研究动机主要来自三个方面的考虑。首先，MBR 编码与 MSR 编码均为特殊的两类再生码，但是近年来针对 MSR 编码的研究比较多，比如 MSR 编码的参数设置研究[36-38]、DRC codes[39]、具有最优修复性质的 MSR 编码研究[23, 40, 41]、跨机架的 MSR 编码修复研究[42]、MSR 编码精确修复研究[43]等。现有的研究中针对 MBR 编码的研究较少，因此对 MBR 编码的读写、修复性能进行研究是很有意义的。其次，从 1.2 小节所介绍的的研究现状来看，采用并行化技术确实能改善系统的读写、修复性能，因而将并行化技术与 MBR 编码结合起来是一个很好的研究方向。

最后，1.2 小节中介绍的 PPR 技术以及 Pipelinging 技术加速了修复过程，减少了修复时间，但是整个网络中的带宽并没有减少，减少的仅仅是整个系统的最大带宽（或者说是某个链路或多条链路上的最大带宽）。因为文献[8,11]主要是考虑了在传统的 RS 码中的使用情况，因而在修复单个失效块时，整个网络的修复带宽依然是单个数据块的 k 倍。Carousel codes 以及 Galloper codes 则是从编码的角度来提升系统的并行性，尤其是系统的读性能有很大改善。然而，Carousel codes 的编码方案是基于 RS 码以及 MSR 码来设计的，基于 RS 码的 Carousel codes 方案在节点修复时系统的总带宽仍然是单个数据块的 k 倍，基于 MSR 码的 Carousel codes 修复带宽有所降低，但仍然不是最优的。类似的，Galloper codes 方案是基于 LRC 码设计的，虽然修复带宽有所降低，但是也不是最优的。

总结起来，我们可以看到并行化能提升系统的修复性能或者读性能，然而上述编码方案在节点失效时修复带宽方面并没有达到最优。MBR 编码是基于完全图构建的，存储的时候每个块存储了两份，而且同一个块的两个副本分别存储在不同的节点上，天然的对并行性有良好的支持。此外，本文中设计的 MBR 编码满足 Dimakis 等人提出的修复带宽下界，即在修复单个失效节点时，不会增加额外的带宽消耗。因此，本文试图从并行化技术以及编码的角度来提升系统的读写、修复、扩展性能。

3.2 P-MBR 系统总体设计

3.2.1 P-MBR 系统的功能和目标

P-MBR 系统是基于 NCFS[26]开发实现的，主要包括两方面的设计，一方面是分布式存储平台层面，另一方面是编码层面。本课题的主要目标在于：在系统中实现 MBR 编码，探索并行化技术对系统的读写、修复性能的影响。针对单节点修复的情形，我们期望取得最小的修复带宽，这在理论上已经被证明（见 2.3.2 小节）。

本系统主要实现的功能如下：从分布式存储平台的角度来说，需要满足文件系统的一些基本操作功能，如读写文件、删除文件、创建文件、查询文件状态等等。此外，为了在系统中实现不同的编码，还需要对不同的编码方案提供统一的接口，使得相应的编解码功能得以实现。为了使得分布式存储平台与具体的存储介质相分离，需要有一个统一的接口，使得文件系统可以透明的访问不同的存储介质，而察觉不到这种区别。为了利用数据局部性原理加速文件读写、修复过程，在文件系统中可以加入缓存的设计。从编码的角度来说，需要设计并实现相应的编码方案，为并行化提供良好的支持。

3.2.2 总体架构

本系统的整体架构如图 3-1 所示。本系统主要分为 4 个层次，由下至上分别为：存储层、缓存层、编码层、文件系统操作层。存储层主要是处理数据的读写，提供对不同的存储设备的透明访问，比如本地存储设备、网络存储设备等。缓存层主要是利用数据的局部性原理，加速对数据块的访问。编码层主要是针对不同的编码方

案，实现对数据的编解码操作，本次实验中使用到的 MBR 编码便是在这个层次实现的。文件系统层主要是对对数据进行各种管理，比如读写、删除、查看文件属性等。文件系统层还提供了到 FUSE 文件系统的接口，因为对文件的操作最终是通过 FUSE 文件系统来实现的。此外，还有一个专门的修复模块来实现对失效节点的修复。

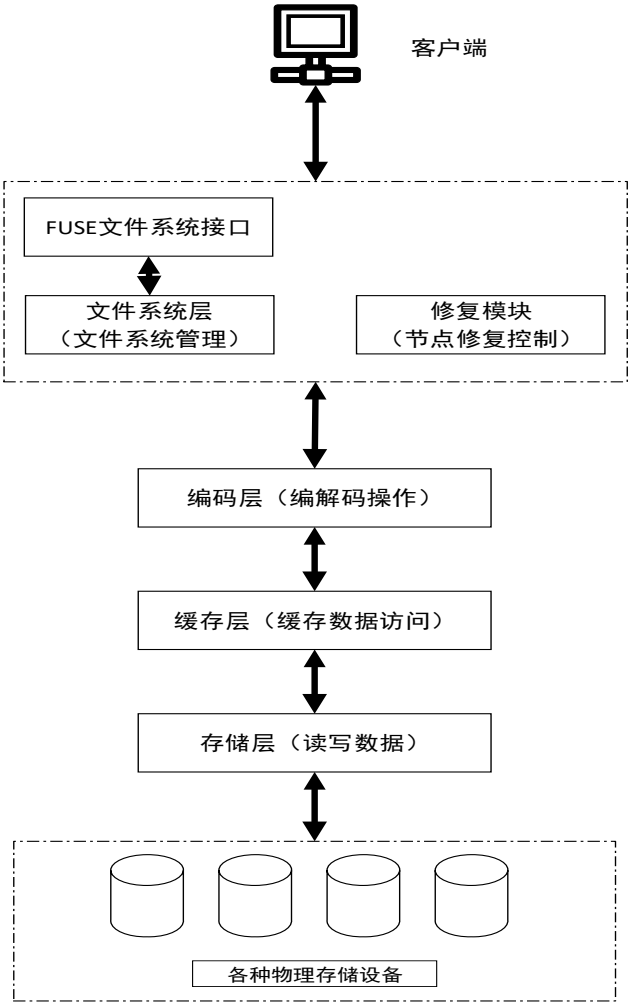


图 3-1 系统层次结构图

3.3 P-MBR 系统各子模块设计

3.3.1 文件系统层设计

文件系统层作为系统的最上层，直接处理用户的各种请求。接收到用户请求后，文件系统层还会把这些操作传递给 FUSE 提供的接口，从而实现相应的功能。具体

来说文件系统主要实现如下功能：文件的读取、写入、删除，文件属性的查询，权限控制，文件夹创建、删除，获取和修改磁盘状态，读取系统配置文件，元数据管理，文件系统初始化等。下面以文件读取、写入，获取、修改磁盘状态，读取配置文件为例进行详细说明，其他的功能此处略去。文件系统层的设计如图 3-2 所示。

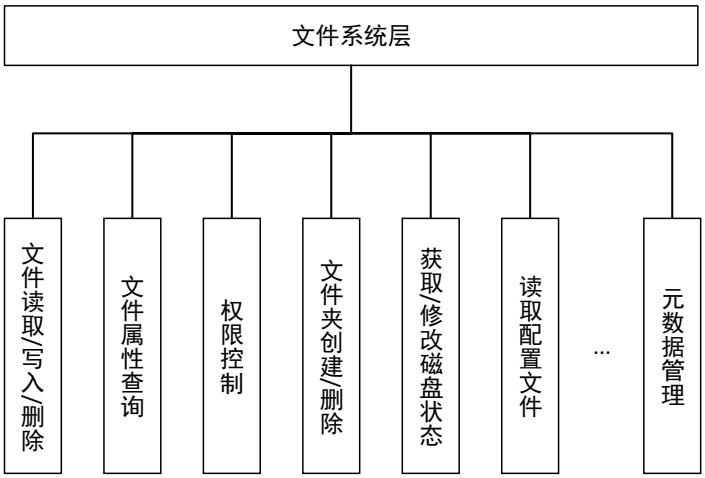


图 3-2 文件系统层

（1）文件读取：指的是从分布式存储系统中将文件读取至客户机，主要处理用户向分布式存储系统发出的读文件请求，文件系统层将读文件请求传递到相应的接口，进行文件的读取；

（2）文件写入：指的是从客户机将文件写至分布式存储系统，主要处理用户向分布式存储系统发出的写文件请求，文件系统层将写文件请求传递到相应的接口，开始进行文件的写入；

（3）获取和修改磁盘状态：磁盘状态有 0、1 两种状态，0 代表磁盘正常，1 代表磁盘故障，通过调用文件系统层的函数可以获取到当前时刻磁盘的状态，或者发现当前磁盘已损坏，可以对磁盘的状态进行修改；

（4）读取配置文件：文件系统中专门用了 config.xml 文件来对各种参数进行控制，可以通过读取配置文件来对各参数进行初始化。

3.3.2 编码层设计

编码层最主要的功能在于实现各种编码方案，为各种编码方案提供统一的接口。

当分布式存储系统中读入数据时，根据配置文件中预先设置的编码方案来进行编码，然后将编码后的数据通过存储层存放到存储设备上。当节点失效时，如果数据无法访问，可以通过调用解码操作来进行降级读以及数据块恢复等操作。编码层的设计如图 3-3 所示，编码解码都有统一的接口，通过这个接口，可以实现 raid5、raid6、RS 以及本文中使用的 MBR 编码方式。总结起来，编码层的主要职责是编码、解码操作。

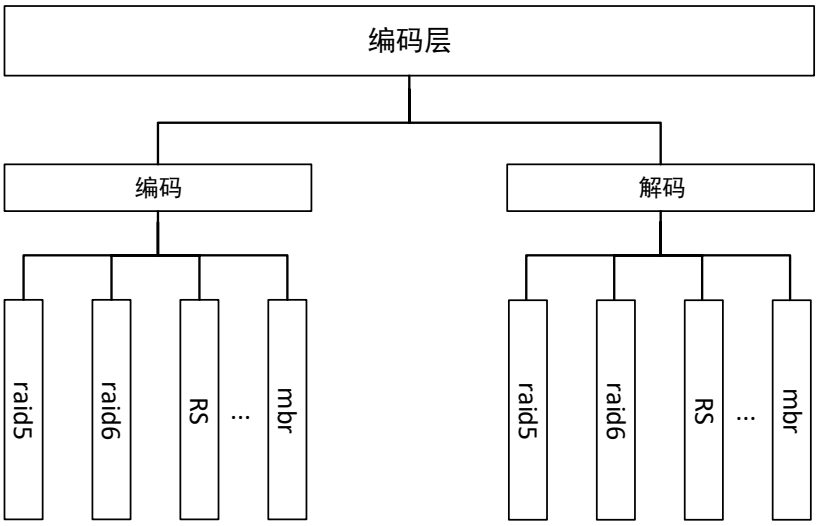


图 3-3 编码层设计

3.3.3 缓存层设计

缓存层主要包括三个部分，文件缓存、文件信息缓存、磁盘缓存。文件缓存和文件信息缓存主要是为文件系统层提供服务，比如文件的起始地址、偏移、读取字节等主要存储在文件信息缓存中，文件信息缓存配合文件缓存就可以执行对文件的读写等操作。磁盘缓存是具体存储数据块的地方，通过磁盘缓存可以快速的读写数据块。此外，上述三个缓存均有读、写、删除操作，而且每个缓存的大小、缓存个数、块大小、页大小、缓冲 buffer 大小都在配置文件中事先设置好了。至于刷入操作，主要是当缓存中接收到的数据到一定程度时，就一次性将缓冲区中的数据刷入到磁盘或者文件中。除此之外，缓存中还加入了线程池，可以利用多线程加速文件读写、修复的过程。缓存层设计如图 3-4 所示。

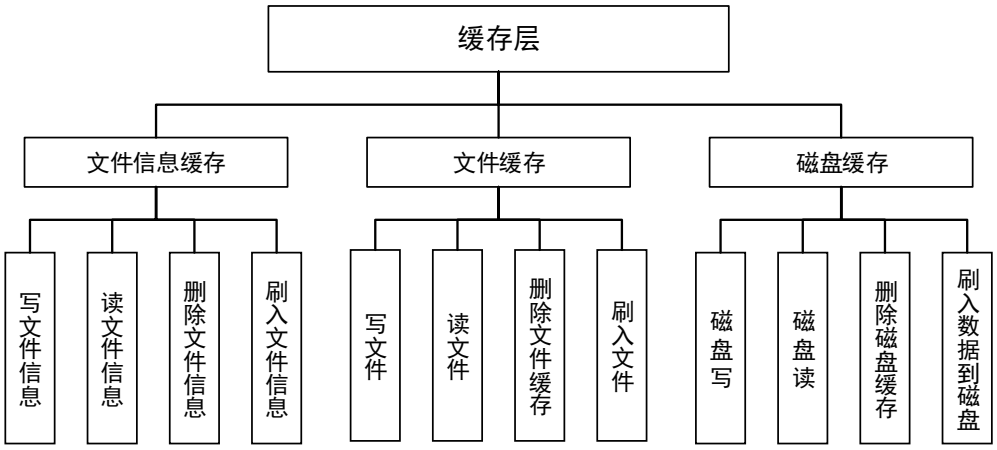


图 3-4 缓存层设计

3.3.4 存储层设计

存储层最主要的功能就是读磁盘和写磁盘，存储层只需要查询文件所在的磁盘 ID，就可以对定位到的磁盘进行读写工作。存储层到实际的存储设备间存在一个映射关系，这种映射关系是通过节点号、节点上的块偏移来进行的。存储层为文件系统访问不同的存储设备提供了统一的接口，对于具体使用何种物理存储介质，文件系统系统并不关心，因而文件系统可以透明的访问不同的存储设备。文件系统对存储设备的访问，可以是本地的 PC 机通过局域网等方式进行，也可以是网络上的各种存储设备，比如网络附连存储设备（NAS）、各种云存储设备（Amazon S3）等，通过诸如 iSCSI、AoE 等网络协议进行。存储层设计如图 3-5 所示。

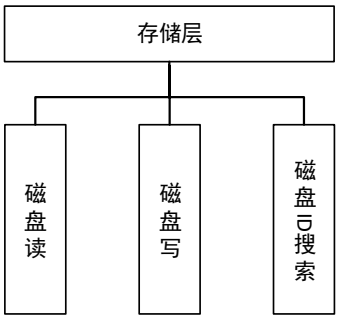


图 3-5 存储层设计

3.4 MBR 编码的并行化设计

3.4.1 MBR 编码的构造实例

基于 2.4 小节中的完全图 MBR 编码构造方法，本文所采用的一种 MBR 编码实例如图 3-6 所示。图 3-6（a）表示的是无校验块的情形，节点个数 n=4,任意 k=3 个节点可以恢复出失效的节点。四个节点上分布有 0、1…5 的数据块，每一个数据块分别存放在不同的两个节点上，也就是说对于每一个块是以双副本的形式存放的。图 3-7（b）表示是的有一个校验块 P 的情形，校验块 P 是通过数据块 0、1…5 异或形成的，0、1…P 相当于 RS（6,5）编码。我们将原文件形成的校验块的个数作为实际的校验块个数 c，原文件直接形成的数据块个数记为 m，这 n(n+1)个块称为一个组，其余的原文件上的数据块以及校验块算作一个副本，则 c=1，m=5，总的块的个数为 2（m+c）=n(n+1)=12 个。在该情形下，4 个节点中的任意 2 个节点可以恢复出原文件。

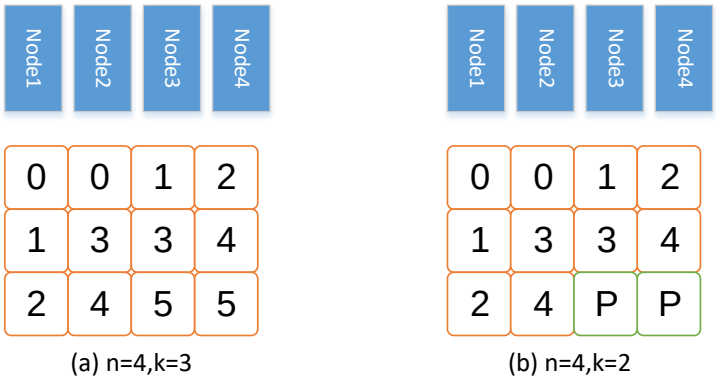


图 3-6 基于完全图的 MBR 编码实例

3.4.2 MBR 编码的并行化读写设计

MBR 编码的并行化读写设计主要包括正常读、正常写、降级读三个方面。正常读指的是节点没有损坏的情况下，客户机向分布式存储系统请求读某一个文件。正常写同样要求节点没有损坏，只不过此时是客户机请求向分布式存储系统请求写入文件。降级读指的是在节点有损坏的情形下，客户机向分布式存储系统请求读取某一文件，此时存储系统能响应客户请求，但是并不会立即修复失效节点，而是等到

读取完毕后才会进行修复操作。

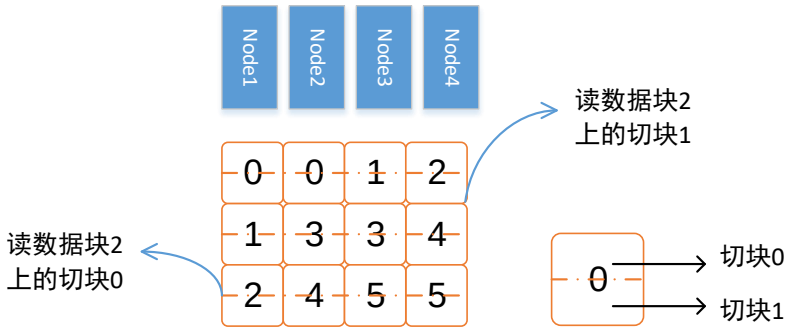


图 3-7 并行读设计

（1）正常读并行化设计

先讨论 n=k-1 的情形，如图 3-7 所示，n=4，k=3。受到文献[11,24]-25]中研究工作的启发，为了能够使得读取数据块时达到并行化的效果，我们首先以更小的粒度将每个块分成两个大小相同的更小的切块，对于每一个数据块分成切块 0、切块 1 两部分。客户发出正常读的请求时，就把对每一个数据块的读取分散到不同的两个节点上，而且对于每一个块只读取其中一个切块，从而达到并行化读的目的，此外系统也并不会读取多余的块。比如，当客户请求读取数据块 2 时，我们可以读取节点 1 上数据块 2 的 0 号切块，节点 4 上数据块 2 的 1 号切块。针对有校验块的情形，使用上述同样的方法读取数据块即可，对于校验块，并不参与正常读。

（2）正常写并行化设计

针对无校验块的情形，即 k=n-1 的情形，如图 3-8 所示，n=4，k=3。当客户机发出写请求时，会将每一个块同时写入两个节点。由于每一个块都是分布在不同的两个节点上，因而使得并行化写成为可能。以数据块 2 为例，当向分布式存储系统中写入数据块 2 时，根据 MBR 编码的方式，其中一个数据块 2 写入节点 1，数据块 2 的副本写到节点 4 上，这两个块的写入是同时进行的。类似地，写入其他块的时候，均是通过这种方式。

针对有校验块的情形，即 k=n-2。相对于 k=n-1 的情形，唯一的不同之处在于增加了校验块，因此在写入校验块之前会先对数据块采用异或编码算法生成校验块 P，然后才会同时将编码块 P 写入到两个不同的节点上。

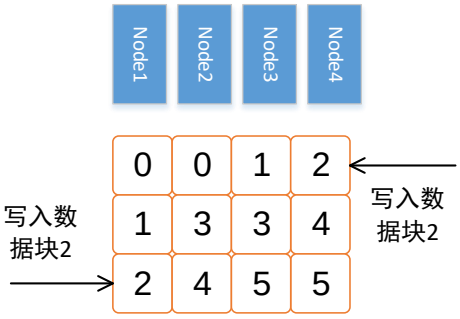


图 3-8 并行化写设计

（3）降级读并行化设计

降级读并行化设计如图 3-9 所示，图中展示的是 k=n-1，单节点失效的情形，n=4，k=3。当有一个节点失效时，分布式存储系统接收到用户发出的读请求后，此时仍能进行响应。基于完全图的 MBR 编码方案，每一个节点上存储的块是不一样的。此外，对于每一个节点上的块的副本而言，也是分别位于彼此不同的节点上，因此本编码方案对于降级读有着极好的性能。依图中示例所示，当节点 1 失效时，节点 1 上的数据块 0、1、2 无法读取，但是这三个数据块的副本分别位于节点 2、3、4 上，因而这三个节点可以同时传输 3 个失效的数据块，提高了系统的并行性。对于有校验块的情形，也是类似的。

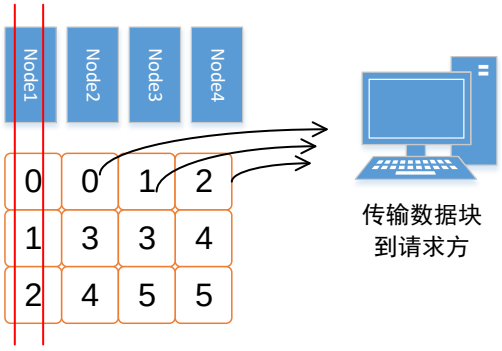


图 3-9 降级读并行化

3.4.3 MBR 编码的并行化修复设计

节点的并行化修复设计如图 3-10 所示，首先讨论 k=n-1 的情形，n=4，k=3。当系统中有单个节点失效时，若客户端发出修复请求，则会在新的节点上重新生成节点 1 上的数据块。同样地，由于每一个节点上的数据块分布在不同的节点上，因而

有良好的并行度。另一方面，在修复失效节点时，并没有传输额外的数据块，所传输的数据块刚好与节点上失效的块一致，此时修复带宽最小，达到了 Dimakis 等人在存储容量-带宽平衡曲线中所提出的 MBR 点的最小修复带宽下界。比如，当节点 1 失效时，结点 2、3、4 分别传输数据块 0、1、2 到新的节点 5 上，这三个数据块的传输可以同时进行，而且分布在三个不同的节点上，彼此互不干涉，从而提高了系统的修复并行度。

下面讨论 k=n-2 的情形，在本例中 n=4，k=2，有一个校验块 P。当有一个节点失效时，与 k=n-1 的情形中单节点失效时的恢复方法一致。当有 2 个节点失效时，理论上是可以恢复的，注意 k=n-1 时是无法恢复 2 个失效节点的。2 个节点失效时，虽然也可以并行化地进行修复，但是在修复过程中不可避免地要通过解码来修复某些数据块，因而会导致额外的网络带宽消耗，此时网络带宽并不是最优的。但是与经典的 RS 码修复方法相比，在一定程度上仍能减小带宽消耗。

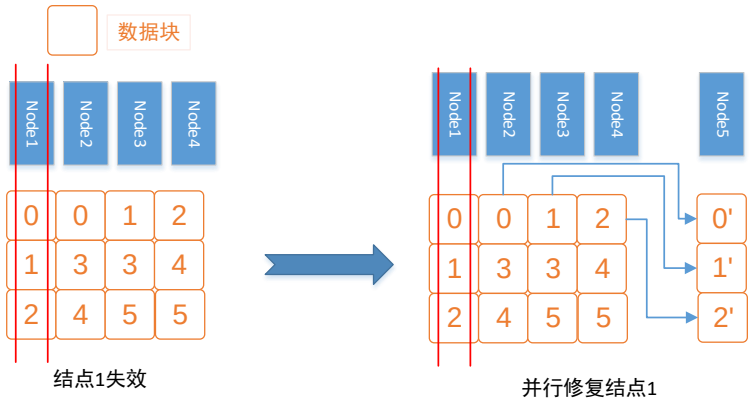


图 3-10 并行化修复设计

3.5 本章小结

本章主要阐述了 P-MBR 系统的设计以及 MBR 编码的并行化设计。首先给出了总体的框架，然后以分层的形式简述了各部分的功能和设计。对于 MBR 编码并行化设计，先是给出了 MBR 编码的构造示例，然后分别从读、写、修复等三个方面给出了设计方案。对于修复而言，文中的设计方案还保证了修复的最小带宽。

4 P-MBR 系统及 MBR 编码的并行化实现

基于第 3 章的设计方案，本部分主要是给出 P-MBR 系统及 MBR 编码的并行化实现，尤其是 MBR 编码的并行化实现是本部分的重要工作。本章首先对文件系统的实现作了了详细阐述，然后从 MBR 编码并行化读、写、修复等方面给出了具体的实现过程。

4.1 文件系统操作实现

4.1.1 主要数据结构

P-MBR 系统是基于 FUSE 实现的，FUSE 为文件系统提供了一套操作接口，在具体实现的时候，开发者的主要工作就是完善接口的功能，从而可以对文件进行读、写等操作。FUSE 中最重要的结构体就是 fuse_operations，本系统中实现的结构体是 ncfs_fuse_operations，它继承了 fuse_operations。可以看到在该结构体中包含了成员函数 ncfs_fuse_operations()，成员函数中等号左侧都是函数指针，右侧是具体的在当前系统中实现的函数。该数据结构如下所示

```
struct ncfs_fuse_operations:fuse_operations {
    ncfs_fuse_operations() {
        getattr = ncfs_getattr;     //获取文件属性
        readlink = ncfs_readlink;
        getdir = NULL;
        mknod = ncfs_mknod;
        mkdir = ncfs_mkdir;
        unlink = ncfs_unlink;     //删除文件
        rmdir = ncfs_rmdir;
        symlink = ncfs_symlink;
        rename = ncfs_rename;
        link = ncfs_link;
        chmod = ncfs_chmod;
        chown = ncfs_chown;
        truncate = ncfs_truncate;       //改变文件大小
        utime = ncfs_utime;             //更改文件的访问和修改时间
```

```
        open = ncfs_open;       //打开文件
        read = ncfs_read;       //从打开的文件中读取数据
        write = ncfs_write;     //向打开的文件写数据
        statfs = ncfs_statfs;   //获取文件系统统计数据

        ...
};
```

接下来，将对上述结构体中的部分函数作出说明。getattr、readlink、getdir 分别用于获取文件属性、读取符号链接、读取目录内容；mknod、mkdir、symlink 分别表示创建文件节点、创建目录、创建符号链接；unlink、rmdir 分别表示删除文件、删除目录；rename 表示重命名文件夹，link 表示创建到文件的硬链接，chmod 表示修改文件权限，chown 表示修改文件的所有者。其他的一些函数指针也均是对文件或者文件夹的各项操作，这些函数保证了文件系统的正常运行。此外，open、 flush、release、 fsync、 opendir、 releasedir、 fsyncdir、 access、 create、 truncate、 lock、init 、destroy 等是一类特殊的方法，并不是实现一个文件系统所必须的。

为了跟踪系统的状态，定义了一个全局的结构体 ncfs_state。该结构体部分内容如下，限于篇幅未全部列出。首先是关于挂载目录的两个指针，moutdir 是指向挂载目录的指针，挂载目录中存储的文件称之为目标文件，rootdir 主要是存储了目标文件的索引。然后是系统磁盘的状态，比如 disk_total_num、data_disk_num、chunk_size 分别表示磁盘的总数、数据盘的数量、块大小等，系统中设置的块大小是 4KB。disk_raid_type 表示文件系统中采用的编码方案，MBR 编码用 1000 表示。operation_mode 反映了文件系统的状态。接下来，是与 MBR 编码相关的参数，比如 int mbr_n，int mbr_k，int mbr_m，int mbr_c 分别代表了第 3.4 小节中介绍的几个参数 n，k，m，c。

```
struct ncfs_state {
    char *rootdir;
    char *mountdir;
    //start ncfs
    int disk_total_num;
    int data_disk_num;
    int chunk_size;
```

```
        int disk_raid_type;
        int operation_mode;       //0 for normal; 1 for degraded; 2 for incapable
        int *disk_size;         //disk maximum size in number of blocks
        int *free_offset;       //in block number
        int *free_size;         //in number of blocks
        struct space_list *space_list_head;
        int no_cache;           //1 for no cache
        int run_experiment;//1 for running experiment
        int process_state;    //0 for normal, 1 for recovery
        int segment_size;
        int mbr_n;
        int mbr_k;
        int mbr_m;
        int mbr_c;
        int mbr_segment_size;
        // (m*w) * (k*w). m is fail tolerance and k is data disk number
        int *generator_matrix;
        //end ncfs

        …
    };
```

data_block_info 则主要是一些关于数据块的信息。该结构体的成员主要有磁盘 ID 以及块号。磁盘 ID 用于标识一格磁盘，块号用于标识一个块在该磁盘上的位置。

```
    struct data_block_info{
            int disk_id;
            int block_no;
    };
```

recovery_info 主要用于节点修复磁盘信息的记录。failed_disk_id 主要标识失效的磁盘 ID，surviving_disk_id 用于标识幸存的节点 ID。

```
struct recovery_info{
            int failed_disk_id;
            int surviving_disk_id;
    };
```

4.1.2 主要函数功能

文件系统启动的流程如图 4-1 所示，当用户在启动文件系统时，会传入参数。首先，文件系统会通过构造器进行一定的初始化，接着判断用户输入的参数是否规范，

参数的规范性检查主要是通过参数个数来判断的。若参数不符合规范，则用户需要重新输入。参数输入正确后，文件系统的状态会进行初始化，主要是对 rootdir、mountdir 等赋初值。接下来，会读取系统的配置文件，系统的配置文件是用户事先通过 config.xml 进行配置的，其中包含磁盘总数、数据盘个数、编码类型等参数。接着，会获取文件系统的元数据信息。然后，会对 MBR 编码的参数进行处理，此外还会调用范德蒙矩阵生成带有校验块的 MBR 编码。处理完 MBR 编码的参数后，会获取当前磁盘状态，比如磁盘损坏或是正常，同时会获取当前文件系统的工作模式，包括正常读、降级读、系统不可用等状态。最后会将参数、文件系统状态等传递给 fuse_main，底层 FUSE 会完成用户对文件的操作等。

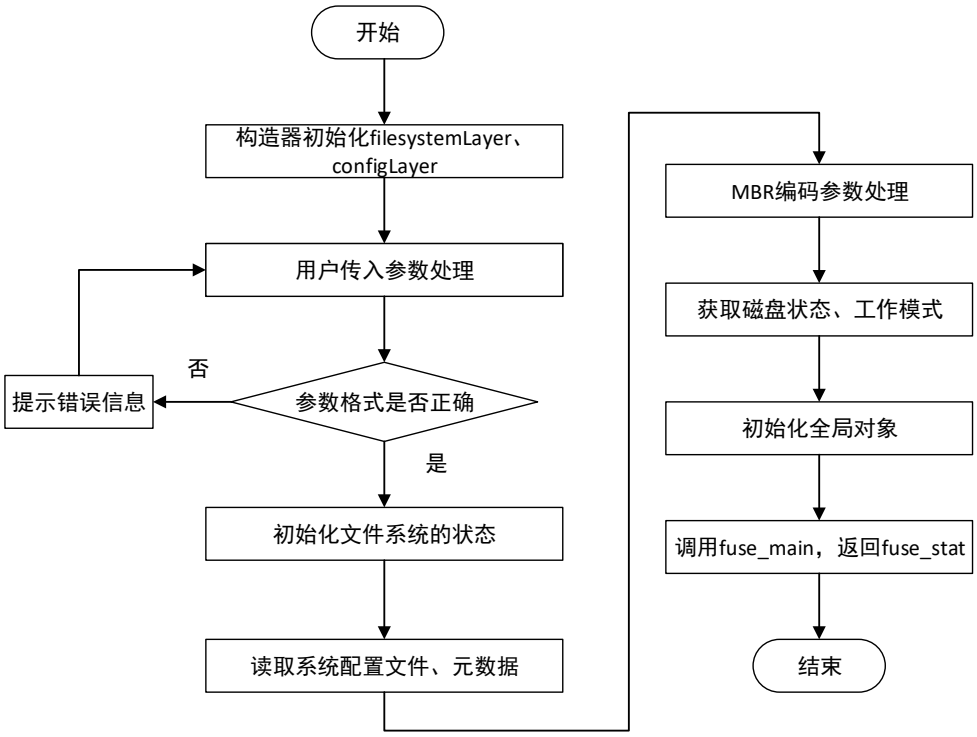


图 4-1 文件系统启动

上述流程中比较重要的函数主要有以下 2 个：

void FileSystemLayer::readSystemConfig(ncfs_data)

功能描述：该函数主要是读取配置文件中预先定义好的文件系统参数；

参数描述：ncfs_data 是一个结构体类型的指针，该指针指向文件系统的状态，包含了文件系统的状态信息。

fuse_main(argc, argv, &ncfs_oper, ncfs_data)

功能描述：调用 FUSE 模块的入口；

参数描述：argc 表示用户在命令行中输入的参数个数，argv 是用户输入的参数，ncfs_oper 是一个 ncfs_fuse_operations 类型的指针，ncfs_data 同上；

返回值：返回值 fuse_stat 表示调用 FUSE 是否成功

4.2 并行化读写实现

4.2.1 并行化读实现

并行化读流程如图 4-2 所示，并行化读包括两种，第一种是正常读，第二种是降级读。

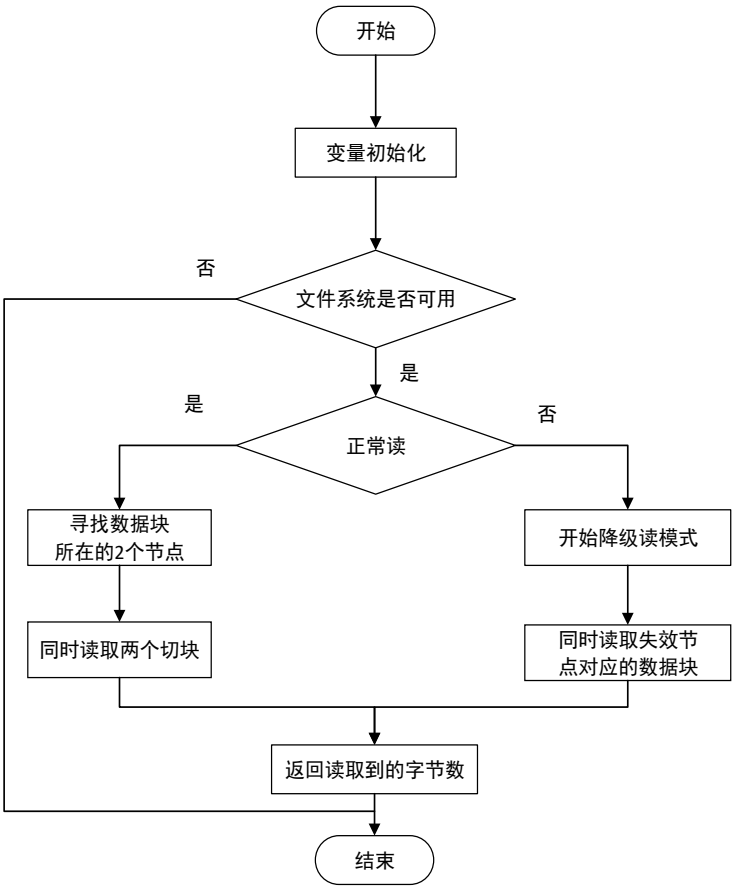


图 4-2 并行读、降级读流程

当用户发出读取文件的命令时，首先会判断文件系统当前处于何种状态，若文

件系统不可用，则输出错误信息，结束读流程。所谓文件系统不可用指的是文件系统中失效节点数过多，因而无法完成用户的读请求。

当文件系统中所有节点正常工作时，此时处于正常读模式。正常读模式下，会按照 3.4 小节中所设计的读方案，按照文件流寻找每个数据块所在的两个不同节点，然后对节点 ID 小的数据块读取切块 0，对另一个节点 ID 大的数据块读取切块 1。读取完毕后，会返回读取到的字节数。

当文件系统中有节点失效时，但是仍然满足降级读条件，则此时处于降级读模式。降级读模式下，首先会读取失效节点上的数据块，此时是直接读取整个数据块，不按照切块读取，当读取完失效节点对应的块时，开始采取与正常读同样的方式读取其他的块，读取完毕时返回读取到的字节数。

该流程中比较关键的函数如下：

int ncfs_read(path, buf, size, offset, fi)

功能描述：实现并行化读

参数描述：path 是字符型指针，指向文件路径，buf 是指向文件缓冲区的指针，size 是要读取的数据的大小，offset 指的是文件偏移，fi 是指向 FUSE 提供的指向文件信息的指针

返回值描述：返回读取到的字节数

int CacheLayer::FileInfoRead(id, buf, size, offset)

功能描述：从缓存中读取文件信息

参数描述：id 是 FUSE 提供的指向文件信息的 ID，buf 表示数据缓冲区，size 表示缓冲区大小，offset 表示读取地址的偏移量

返回值描述：返回成功读取到的字节数

4.2.2 并行化写实现

并行化写流程图如图 4-3 所示。当用户发出向文件系统写入的命令后，首先会对相关的变量初始化，比如写入地址、文件大小等。然后，判断当前系统的状态，与并行读不同，即便文件系统处于降级读模式，此时也不支持向文件系统写入文件。

同样地，当文件系统由于失效节点过多而不可用时，写入过程同样无法进行。上述两种情形，均会给出错误提示信息。

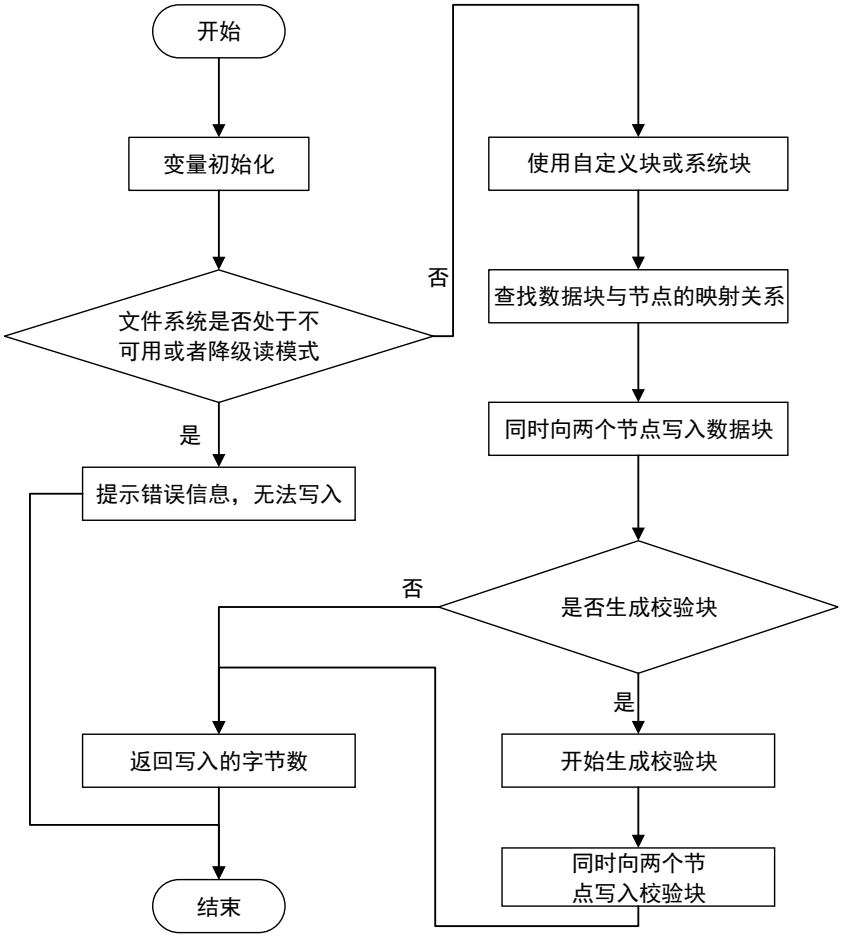


图 4-3 并行化写

当文件系统中节点没有损坏时，此时可以进行正常的写入操作。写入的时候，可以以用户自定义的块大小进行写入操作，也可以以系统定义的默认块大小进行写入操作，默认块大小是 4KB，自定义块大小一般是 4KB 的倍数，因而使用自定义块进行写入也称之为大块写入。写入时，会按照文件流依次同时写入同一个数据块到两个不同的节点上，针对有校验块的情形，会调用编码模块生成校验块，然后并行地写入校验块所对应的 2 个不同的节点上。该模块的主要实现函数如下：

int ncfs_write(path, buf, size, offset, fi)

功能描述：实现向文件系统中并行写入

参数描述：字符型指针 path 指向文件路径，buf 指向数据缓冲区，size 表示要写入的文件大小，offset 表示文件的偏移，fi 指向 FUSE 提供的文件信息

返回值：返回写入的字节数

struct data_block_info CodingLayer::encode(buf, size)

功能描述：实现编码块的生成和写入

参数描述：buf 指向要被写入的磁盘的数据缓冲区，size 表示数据缓冲区的大小

返回值描述：返回写入的块所在的磁盘 ID 和块号

int CacheLayer::FileInfoWrite(id, buf, size, offset)

功能描述：向缓存写入文件信息

参数描述：id 表示文件信息的 ID，buf 为指向写入缓存的数据缓冲区，size 为数据缓冲区的大小，offset 为写入文件地址的偏移

返回值描述：成功写入的字节数

4.3 并行化修复实现

并行化修复是通过一个专门的修复工具类 recovery.cc 实现的，总体的修复流程如图 4-4 所示。首先，用户在命令行输入参数，主要包括新的设备以及失效的节点 ID，在实验中我们是通过人为设置的方法使磁盘处于失效状态，具体来说，每个磁盘都有一个状态标识，标识为 0 表示磁盘正常，标识为 1 则表示磁盘已损坏。如果用户输入参数不规范，则会提示错误信息。否则，开始进入正常修复流程，先会初始化文件系统、配置文件，接着会对文件系统中状态相关的部分变量赋初值，然后会从用户预先定义好的配置文件中读取与文件系统状态相关的参数，然后会从 metadata 文件中获取元文件。

接下来，会判断文件系统所采用的编码类型，如果是 MBR 编码的话，会计算 MBR 编码相关的一些参数，调用 Jerasure 库中的范德蒙矩阵产生生成矩阵，如果是其他编码类型，则采用默认方式设置编码参数。编码参数设置完毕后，会获取磁盘当前状态以及当前的文件系统的运行模式，接下来会初始化全局对象 NCFS_DATA，调用 recovery_init 函数，recovery_init 主要是初始化缓存层、存储层、编码层。

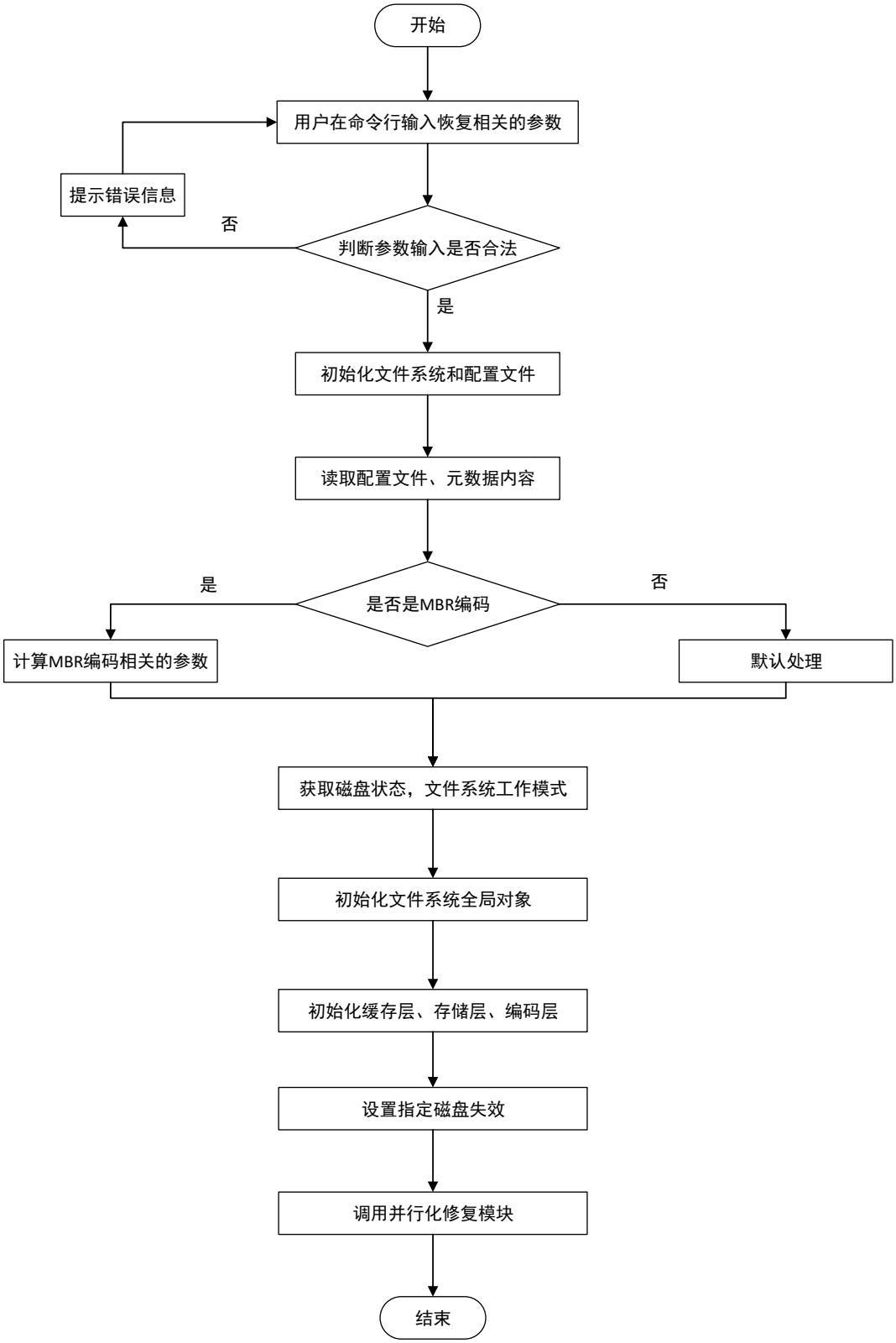

开始
用户在命令行输入恢复相关的参数
提示错误信息
否
判断参数输入是否合法
是
初始化文件系统和配置文件
读取配置文件、元数据内容
是
是否是MBR编码
否
计算MBR编码相关的参数
默认处理
获取磁盘状态，文件系统工作模式
初始化文件系统全局对象
初始化缓存层、存储层、编码层
设置指定磁盘失效
调用并行化修复模块
结束

图 4-4 并行化修复

上述工作就绪后，会根据用户传入的失效磁盘 ID，设置该磁盘状态为失效状态，达到降级读模式，此时是支持恢复的。然后就会调用修复模块 RecoveryTool::recover()开始进行并行化修复，在新的节点上恢复出失效节点上的数据。

函数接口 RecoveryTool::recover()主要是根据不同的编码类型，调用相应的修复方法。MBR 编码情形下，会将失效磁盘 ID 指向新的设备，然后会通过解码操作来读取失效节点所对应的数据块。读取失效数据块时，通过多线程的方法，使得同时读取同组内失效节点上的数据块成为可能，因为同一个节点上的数据块均匀分布在不同的节点上。此外，读的时候是在磁盘缓存中进行读取，这样一来也可以加速修复过程。线程的数量也是可以设置的，但是线程数量并不是越多越好，有可能会增加系统开销。当一个组读取完毕后，就会开始调用写函数 cacheLayer->DiskWrite，向新的节点上写入数据块，写入的过程基本是与读取的过程是相反的。写的时候，也是按照单个组依次写入的。如果有多个组，重复上述过程即可。

上述并行化修复流程中涉及到的关键函数如下：

int main(argc, args[])

功能描述：修复工具的入口函数

参数描述：argc 代表用户输入的参数个数，args 则存储了用户输入的参数

返回值描述：成功返回 1

void RecoveryTool::recover()

功能描述：修复工具类，主要是针对不同的编码类型，调用相应的修复方法

参数描述：该函数接口无参数传递

返回值描述：无返回值

int CodingLayer::decode(disk_id, buf, size, offset)

功能描述：节点失效时，通过解码操作，读取失效数据

参数描述：disk_id 表示磁盘 ID 号，buf 表示用于数据读取的缓冲区，size 代表缓冲区大小，offset 数据块在磁盘上的偏移量

返回值：根据在配置文件中所使用的编码方式，返回相应的解码函数接口，进行解码操作

4.4 本章小结

本章主要详细阐述了文件系统以及 MBR 编码并行化的实现过程。首先，详述了文件系统的实现流程，介绍了系统实现中比较重要的数据结构，以及一些关键函数的功能。然后，阐述了并行化读的实现，主要包括并行化正常读和降级读，并行化读是通过封装 FUSE 的读函数接口实现的。然后，给出了并行化写的实现过程，并行化写主要是按照文件流的顺序依次并行地写入。最后，通过一个专门的修复工具类，实现了并行化修复。并行化主要是通过多线程实现的，缓存的使用也加速了上述读、写、修复过程。

5 实验结果与分析

本章基于前文的实现，给出了 MBR 编码并行化的性能测试结果，主要包括并行化读、写和修复。首先介绍了实现 MBR 编码的实验环境以及实验方案，接着从功能和性能两方面给出了实验结果。为了更好地验证本次实验的成果，将支持单节点容错级别的 RAID5 以及具有双节点容错级别的 RAID6 编码分别与对应容错级别的 MBR 编码进行对比分析。

5.1 实验环境

本次实验在六台虚拟机模拟的服务器上进行，其中一台服务器我们称之为发起端（Initiator）,另外五台服务器称之为目标端（Target）,Target 端中有一台服务器仅用于节点修复。Initiator 主要是运行分布式文件系统，提供对 Target 的访问；Target 主要是作为一个存储设备供 Initiator 访问。Initiator 和 Target 之间是通过 Aoe (ATA over Ethernet)协议来进行通信的， 二者处于同一局域网下。限于实验环境，各服务器的网络带宽为 100Mbps，采用 Aoe 协议，理论上 Initiator 与 Target 之间的传输速度大约为 10MB/S。具体的实验环境如表 5-1 所示：

表 5-1 实验环境

设备	配置
CPU	Intel(R) Xeon(R) E3-1225 v5 @ 3.30GHz 4 核
内存	DDR4 2400MHz 64G
硬盘	SATA 2T @7200R/M
操作系统	Ubantu-14.04.4 64 位
虚拟机设置	磁盘 40G 内存 4G 4 核
虚拟机带宽	100Mbps
FUSE 版本	2.9.2

5.2 实验方案

本实验主要从功能和性能两个方面进行了测试。功能测试主要测试了 MBR 编码的读、写、修复操作，读操作包括正常读和降级读两方面。性能测试主要测试了 MBR 编码的读、写、修复吞吐率，带宽，时间开销以及存储成本。

5.2.1 功能测试方案

MBR 编码写操作方案：客户在 initiator 端运行文件系统，将指定文件从本地传输至文件系统中，原始数据通过 MBR 编码后存储在 target 端的 4 个节点上，传输完成后在文件系统的挂载点上可以看到所传输的文件；

MBR 编码正常读操作方案：在文件写入文件系统后，从文件系统中读取该文件到本地，读取操作完毕后，将该文件与原文件对比，测试二者的一致性；

MBR 编码降级读操作:在文件写入文件系统后，人为的设置某个节点为失效状态，此时从文件系统中读取文件到本地，降级读操作完毕后，测试该文件与原文件是否一致；

MBR 编码修复操作：文件写入文件系统后，设置某个节点失效，然后开始修复操作，在新的节点上生成失效节点上的数据，修复操作完成后，将该文件下载到本地，测试该文件与原文件的一致性；

功能测试如表 5-2 所示：

表 5-2 功能测试

测试项目	描述
正常写	将文件从 initiator 端传输到 target 端
正常读	将文件从 target 端传输到本地，对比文件传输前后的差异
降级读	设置节点失效后，将文件从 target 端传输到本地，比较文件差异
修复操作	节点失效后，对节点进行修复，将修复后的文件传输至本地

5.2.2 性能测试方案

性能测试主要包括三个方面。第一，测试 MBR 编码读、写、修复操作的吞吐率，

针对 MBR 编码测试了不同文件大小下的读、写、修复性能，针对同一文件大小对比测试了 RAID5、RAID6、MBR(k=n-1)、MBR(k=n-2)读写修复性能；第二，对比各编码方案，测试了不同编码方案下的系统存储空间开销；第三，测试了不同编码方案下的网络带宽开销。具体测试项目如表 5-3 所示：

表 5-3 性能测试

测试项目	描述
正常写	测试 MBR 编码写文件的吞吐率，并与 RAID5、RAID6 对比
正常读	测试 MBR 编码下读文件的吞吐率，并与 RAID5、RAID6 对比
降级读	测试 MBR 编码降级读文件时吞吐率，并与 RAID5、RAID6 对比
修复操作	测试 MBR 编码修复失效节点的吞吐率，并与 RAID5、RAID6 对比
存储开销	测试不同编码方案下，针对同一文件的存储开销对比
带宽消耗	测试不同编码方案下，针对同一文件的带宽消耗对比

5.2.3 性能测试指标

为了评估实验效果，文中设置了三个评价指标，分别如下：

吞吐率：吞吐率主要反应了文件读、写、修复时的速度，单位时间内处理的数据量越大，那么吞吐率就越高，系统的性能也就越好。如果单位时间 T 内处理的数据量大小为 S，吞吐率用 Throughput 表示，那么吞吐率的计算公式可以表示为公式 5-1：

$$Throughput = \frac{S}{T} \tag{5-1}$$

存储开销：针对同一文件，不同的编码方案所占用的系统存储空间也不同，文中将其作为该编码的存储开销。

带宽消耗：带宽消耗主要指的是节点失效时，为了修复失效节点，系统所需要传输的数据量。在同样的失效节点个数的情况下，文中将不同的编码方案为修复失效节点所传输的数据量称之为带宽消耗。

5.3 实验结果

5.3.1 功能测试

根据 5.2.1 小节的测试方案，依次测试了 MBR 编码的读写、修复功能。首先测试正常写操作，如果从 initiator 端将文件传输至 target 端后，在 initiator 端的文件系统挂载点中能看到传输的文件，则可以认为文件写入成功；接着，测试正常读操作，将文件从 target 端读取到 initiator 端后，利用 linux 的 diff 命令与原件进行对比，如果二者一致，则认为读功能正常；然后，测试降级读操作，先设置某个节点失效，然后将文件从 target 端读取到 initiator 端，将之与原文件进行对比，如果一致则认为降级读功能正常；最后，测试修复操作，设置某个节点失效，然后在新的节点上恢复出失效节点上的数据，将修复后的文件读取至 initiator 端并与原文件进行对比，如果二者一致则认为修复功能正常。

在开始功能测试之前，先启动文件系统，启动后在 initiator 端可以查找到 5 个服务器，这个 5 个服务器是 target 端的存储设备，如图 5-1 所示。此外，在本地创建一个大小为 2MB 的文件 test。

```
root@initiator:~/ncfs-1.1.0# ./setup_aoe.sh 4 32
      e5.1          0.033GB       eth0 1024  up
      e5.2          0.033GB       eth0 1024  up
      e5.3          0.033GB       eth0 1024  up
      e5.4          0.033GB       eth0 1024  up
      e5.5          0.033GB       eth0 1024  up
```

图 5-1 target 端存储设备

（1）正常写操作

正常写操作过程如图 5-2 所示。MBR 编码是基于 FUSE 实现的，因而读写操作命令与正常的 linux 下操作命令几乎一致。Initiator 端的挂载点为 mountdir/，首先进入挂载点目录，查看该目录下的文件，可以发现此时没有任何文件，接着返回到本地目录，通过 cp test mountdir/ 命令将事先创建好的文件 test 写入到文件系统中，此时查看挂载目录下的文件，可以发现有 test 文件，因而可以认为写功能正常。写命令的操作步骤如图 5-2（a）所示，文件系统的启动在另外一个终端，开启调试模式

时，写入的过程可以在另外一个终端中看到，部分过程如图 5-2（b）所示。

```
root@initiator:~/ncfs-1.1.0/mountdir# ls
root@initiator:~/ncfs-1.1.0/mountdir# cd ..
root@initiator:~/ncfs-1.1.0# cp test mountdir/
root@initiator:~/ncfs-1.1.0# cd mountdir/
root@initiator:~/ncfs-1.1.0/mountdir# ls
test
```

（a）写命令

```
write[6] 4096 bytes to 2088960 flags: 0x8001
   write[6] 4096 bytes to 2088960
   unique: 549, success, outsize: 24
unique: 550, opcode: WRITE (16), nodeid: 2, insize: 4176, pid: 37137
write[6] 4096 bytes to 2093056 flags: 0x8001
   write[6] 4096 bytes to 2093056
   unique: 550, success, outsize: 24
```

（b）写过程

图 5-2 正常写操作

（2）正常读过程

正常读操作的过程如图 5-3 所示。

```
root@initiator:~/ncfs-1.1.0# ls
cachelog     Makefile  raid_health     recovery_testcase  setup_aoe.sh  test
config.xml   mountdir  raid_metadata   rootdir            setup.sh      test1
doc          ncfs      raid_setting    settings_template  src           time_data
root@initiator:~/ncfs-1.1.0# cp mountdir/test test_read
root@initiator:~/ncfs-1.1.0# ls
cachelog     mountdir       raid_setting       setup_aoe.sh  test1
config.xml   ncfs           recovery_testcase  setup.sh      test_read
doc          raid_health    rootdir            src           time_data
Makefile     raid_metadata  settings_template  test
root@initiator:~/ncfs-1.1.0# diff test test_read
root@initiator:~/ncfs-1.1.0#
```

（a）读命令

```
unique: 2366, opcode: READ (15), nodeid: 4, insize: 80, pid: 37231
read[10] 131072 bytes from 1966080 flags: 0x8000
   read[10] 131072 bytes from 1966080
   unique: 2366, success, outsize: 131088
```

（b）读过程

图 5-3 正常读操作

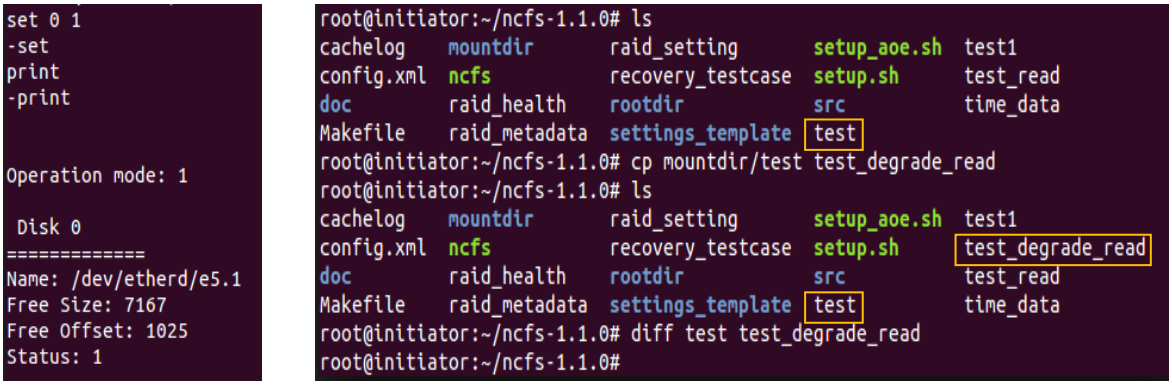
首先在本地目录下查看所拥有的文件，可以发现此时在本地目录中仅有 test 文件，接着通过 cp 命令读取挂载点目录 mountdir/下文件到本地目录中，新的文件命名为 test_read。查看本地目录中的文件，可以发现此时目录中增加新的文件 test_read。

通过 diff 命令可以发现二者一致，因而可以认为读过程是正常的。读命令的操作步骤如图 5-3（a）所示，读过程的部分截图如图 5-3（b）所示。

（3）降级读

降级读如图 5-4 所示。首先，在运行文件系统的终端上通过 set 0 1 命令设置 0 号磁盘失效，输入命令 print 查看当前文件系统的状态，可以看到 0 号磁盘的状态 Status=1，意味着节点 0 已经失效。

在另一个终端，查看本地目录下的文件，可以发现此时有文件 test。接着，在 0 号磁盘已经失效的情况下，向文件系统请求读取文件 test，通过 cp mountdir/test test_degrade_read 命令将文件读到本地目录，并且重命名为 test_degrade_read。查看本地目录文件，可以发现本地目录中新增了文件 test_degrade_read，通过 diff 命令比较二者，可以发现二者一致，因而降级读操作是正常的。图 5-4（a）表示设置节点失效，图 5-4（b）表示降级读的操作步骤。



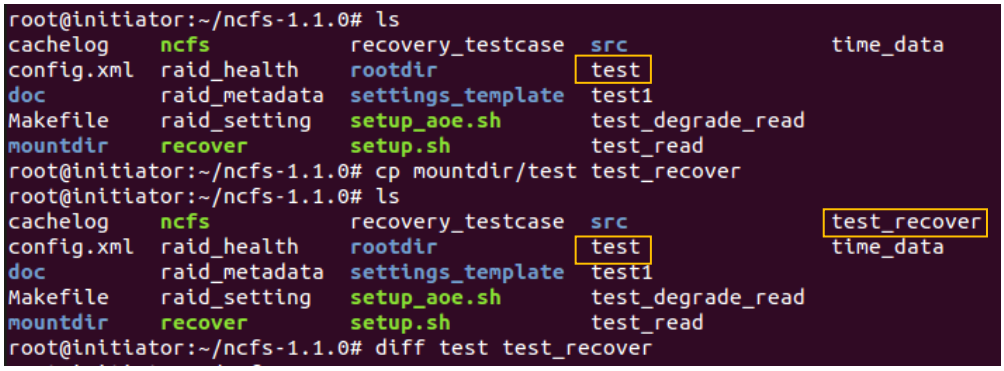
（a）设置节点失效　　（b）降级读命令

图 5-4 降级读操作

（4）修复

修复操作功能测试如图 5-5 所示。修复操作是通过一个专门的修复工具类 recovery.cc 进行的，对 recovery.cc 编译最后生成可执行文件 recover。命令 ./recover /dev/etherd/e5.5 0 表示将 0 号节点设置为失效状态，在新的节点/dev/etherd/e5.5 上生成失效节点上的数据，修复完成后 0 号节点的实际存储位置指向/dev/etherd/e5.5。当修复完毕后，可以看到提示修复完成，修复的过程如图 5-5（a）所示。

（a）修复过程



（b）修复前后文件的一致性对比

图 5-5 修复操作

修复完成后，需要将修复完毕的文件与原文件进行对比。首先，在 initiator 端本地目录中查看当前文件夹下的文件，可以发现有文件 test，无文件 test_recover。接着，通过 cp 命令将修复后的文件读取至本地目录中，重命名为 test_recover。此时，查看本地目录中的文件，可以发现新增了 test_recover 文件，通过 linux 下的 diff 命令与原文件进行对比，可以发现二者一致，因而可以认为修复操作是正常的。

5.3.2 性能测试

性能测试分别测试了 MBR（k=n-1）、MBR（k=n-2）、RAID 5、RAID 6 等四种编码的读、写、修复性能。之所以选取 RAID 5 以及 RAID 6 编码作为对比实验，主

要是从容错的角度来考虑的。MBR（k=n-1）编码以及 RAID 5 编码均能容忍一个节点故障，MBR（k=n-2）以及 RAID 6 编码均能容忍两个节点故障，也就是说它们分别有着相同的容错级别。此外，RAID 5 以及 RAID 6 编码也有着良好的读、写、修复性能，非常适合作为实验的对比对象。

（1）正常写性能

写性能测试结果如图 5-6 所示。实验中分别测试了向文件系统写入 1MB、2MB、4MB 的文件时，MBR 编码与 RAID5、RAID6 编码的写入性能。可以发现，MBR（k=n-1）编码有着最好的写性能，这主要是因为文件以 MBR（k=n-1）编码写入的时候，不用生成校验块，因而不存在编码过程，所以有着最好的写性能。但实际上，MBR 编码在写入的时候有着较高的存储负载，因而写性能提升不大，相较于 RAID 5 编码而言，总体写性能仅提升 10%~20%左右。采用 MBR（k=n-2）以及 RAID6 编码时，系统的写性能几乎是一致的，这主要是因为 MBR（k=n-2）编码也需要生成校验块，同时本身有一定的存储开销，而 RAID 6 编码由于需要在每个条带上生成两个校验块，因而也导致写入速度较低。

相对而言，RAID 5 编码比 RAID 6 以及 MBR（k=n-1）有着更好的写性能，这主要是因为 RAID 5 编码仅仅需要在每个条带上生成一个校验块，而存储开销却又比后两者低。总体而言，MBR（k=n-1）有着较好的写性能，RAID 5 次之，余下两者写性能是类似的。

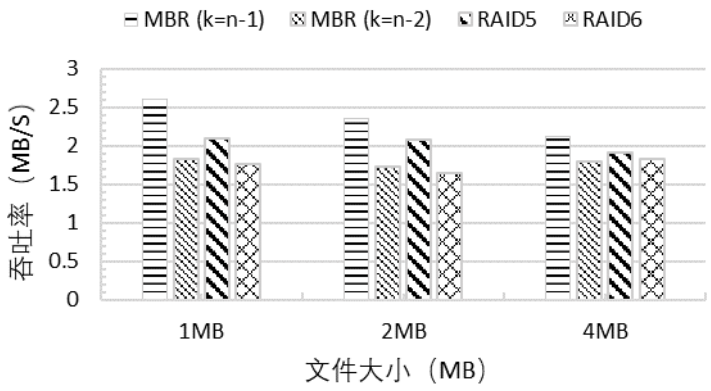


图 5-6 不同编码的写入性能对比

（2）正常读性能

正常读性能测试结果如图 5-7 所示。可以发现正常读情况下，MBR（k=n-1）、MBR（k=n-2）、RAID 5、RAID 6 等四种编码有着类似的读性能。这主要是因为，正常读情形下，不存在编解码过程，不存在额外的开销，只需要直接读取数据块即可。虽然 MBR 编码采用了并行化读的设计，但是 RAID 5 以及 RAID 6 编码也有着较好的并行读性能。因而，上述四种编码的读性能是类似的，但测试中也发现 MBR（k=n-1）在读性能方面也稍微有一些优势。

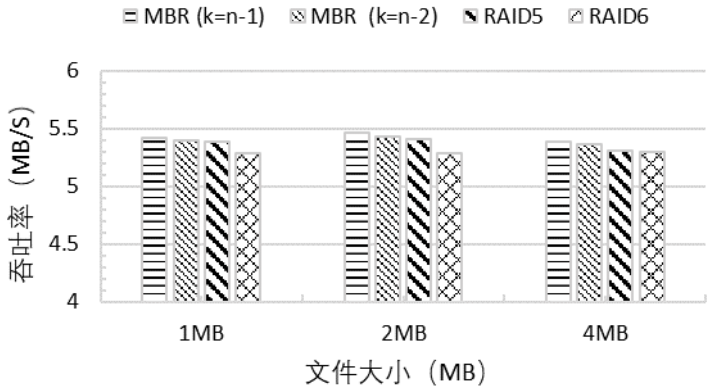


图 5-7 不同编码的正常读性能对比

（3）降级读性能

降级读性能如图 5-8 所示。降级读操作均是测试在单节点失效的情况下，系统的降级读性能。

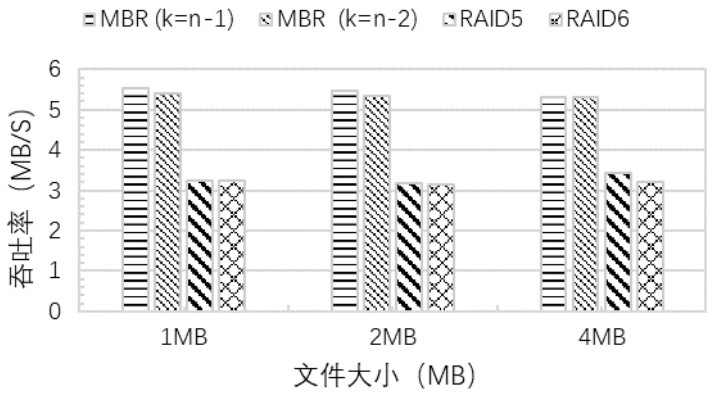


图 5-8 不同编码的降级读性能对比

可以发现 MBR 编码有着极好的降级读性能，相较于 RAID 5 以及 RAID 6 编码性能提升大约 70%。MBR 编码降级读性能好的原因在于，降级读操作时，MBR 编码无需进行解码操作即可读取到数据块。关于这一点在 3.4.1 节 MBR 编码的构造方法中进行了阐述。基于完全图的 MBR 编码，每一个节点上的数据块在另外一个节点上都有副本，因而可以直接读取。与之相对，RAID 5 以及 RAID 6 编码在降级读操作时，为了读取失效节点上的数据块，必须先读取同一条带上的其他编码块（包括数据块或校验块），进行解码来恢复出失效节点上的数据，但是并不在新的节点上写入，仅仅只要能供用户读取即可。但是，由于增加了读取以及解码操作，也会很明显的增加系统开销，从而导致 RAID 5 以及 RAID 6 编码有着较差的降级读性能。

（4）修复性能

修复性能测试结果如图 5-9 所示。同样地，修复操作主要测试了单节点失效情形下，系统的修复性能。可以发现，MBR 编码在修复性能方面仍然有着较好的性能，相对于 RAID 5 提升了 40%左右，相对于 RAID 6 提升了 60%左右。

不同于降级读操作仅读取失效节点上的数据，修复操作需要将失效节点上的数据在新的节点上进行修复，从而能恢复出整个文件。也就是说，相较于降级读操作而言，修复操作增加了将恢复出来的失效数据写到新节点上的操作，无疑会影响系统的总体性能，这也就可以解释修复操作的吞吐率普遍不高。

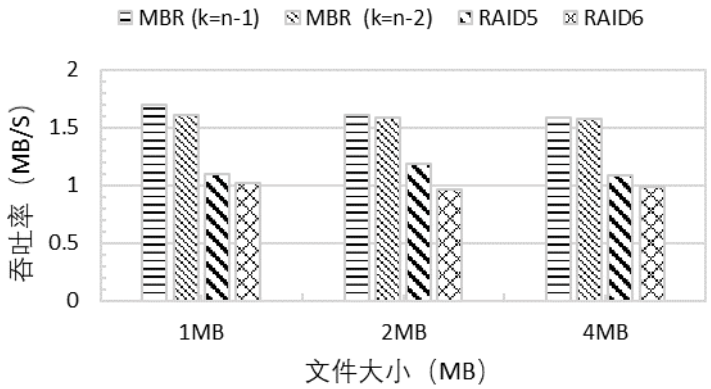


图 5-9 不同编码的修复性能对比

类似的，在单节点失效时，MBR 编码仍然无需解码即可读取到失效数据，不过

在读到失效节点上的数据后，需要将其写到新的节点上，因而相较于降级读操作而言，降低了系统的修复吞吐率。RAID 5 以及 RAID 6 编码在修复操作时，均需要进行解码操作恢复出失效数据，进而写到新结点上，因而二者的修复吞吐率更低。

（5）存储开销评估

存储开销评估结果如图 5-10 所示。可以发现 MBR 编码存储开销比较大，其中 MBR（k=n-2）编码存储开销最大，RAID 6 编码稍小，MBR（k=n-1）次之，RAID 5 编码的存储开销最小。此处，RAID 6 与 MBR（k=n-1）存储开销相同，主要是因为刚好有 4 个节点，随着节点数增多，RAID 6 的存储开销会比 MBR 编码低。具体来说，MBR（k=n-1）的存储开销是大约是 RAID 5 的 150%，MBR（k=n-2）的存储开销大约是 RAID 6 的 120%。

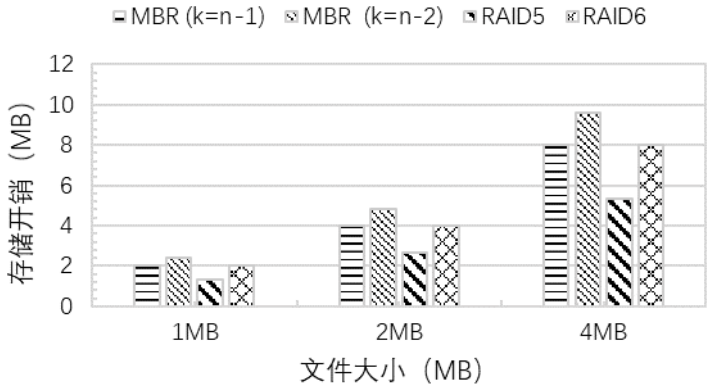


图 5-10 不同编码的存储开销对比

上述结果的本质是编码形式的不同。MBR 编码对于每一个编码块（包括数据块和校验块）都会存储在两个不同的节点上，MBR（k=n-1）编码由于无校验块，因而刚好存储了原始文件的两倍，MBR（k=n-2）由于增加了校验块，理论上存储负载会超过原始文件的 2 倍。RAID 5 编码对于每一个条带上的数据块会生成一个校验块，RAID 6 对于每一个条带上的数据块会生成两个校验块，因而它们会增加部分存储负载，但比对应容错级别的 MBR 编码的存储成本要低不少。

（6）带宽消耗分析

带宽消耗主要是指，在单节点失效时，为了修复失效节点所需传输的数据量。

根据第 2.3 小节的分析，可以得出单节点失效时 MBR 编码的带宽消耗。RAID 5 以及 RAID 6 在修复时总是需要传输与原文件同样大小的数据量[26]，这在许多研究工作中已被证明。

针对不同大小的文件，单节点修复时各编码的带宽消耗如图 5-11 所示。可以看到，MBR 编码的带宽消耗很低，其中 MBR（k=n-1）有着最低的存储消耗，MBR（k=n-2）编码存储消耗稍高，RAID 5 以及 RAID 6 的带宽消耗最高。MBR（k=n-1）的带宽消耗仅为原文件的 25%，MBR（k=n-2）的带宽消耗为原文件的 60%，RAID 5 以及 RAID 6 的带宽消耗为原文件的 100%。根据最大流最小割定理，本文所设计的 MBR 编码达到了修复带宽的理论下界。可以预见，随着数据量的增大，MBR 编码能节省的带宽将极为可观。

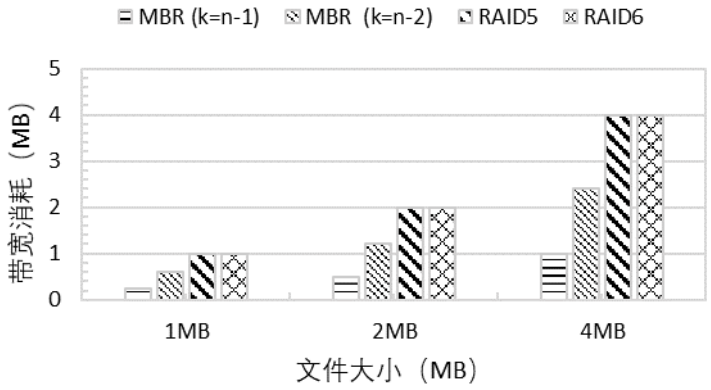


图 5-11 不同编码的带宽开销对比

5.4 本章小结

归纳起来，MBR 编码有着与 RAID 5 类似的读性能，甚至稍好于 RAID 5，写性能也优于 RAID 5。在降级读以及单节点修复场景下，MBR 编码则有着很明显的优势，不仅有着极高的修复吞吐率，而且在修复失效节点时有着最小的修复带宽。因而，可以得出如下结论：本文所设计的 MBR 编码特别适宜于带宽资源紧张、频繁的单节点失效场景。单节点失效在实际应用中也是最常见的，除非系统中同时有多个节点失效，否则总能通过 MBR 编码来进行恢复。

6 总结与展望

当下，正处于一个万物互联的时代，海量的数据对分布式存储系统提出了更高的要求。为了提升系统的可靠性，副本技术以及纠删码技术被广泛应用。但二者各有缺点，副本技术极大的增加了系统存储开销，传统的纠删码技术极大地增加了节点修复的带宽消耗。为了平衡系统的存储开销与带宽消耗，研究者开始研究各种新的编码方式，再生码是当前的最新成果。MBR 编码是一种特殊的再生码，单节点修复时有着最小的修复带宽，相较于三副本而言，也有着更低的存储负载。另一种特殊的再生码 MSR 编码研究较多，关于 MBR 编码的研究较少。因此，本文试图从并行化的角度对 MBR 编码的读、写、修复性能进行研究。本文的主要工作如下：

（1）基于 FUSE 平台，设计并实现了 P-MBR 系统。通过对编码块进行分片设计，对 FUSE 读函数进行封装，研究了 MBR 编码的正常读、降级读性能。实验验证了 MBR 编码有着良好的正常读性能，相较于 RAID 5 以及 RAID 6 编码，有着极佳的降级读性能。

（2）研究了 MBR 编码的写性能。利用 MBR 编码的每一个块分布在两个不同节点上的特性，对 FUSE 的写文件函数进行封装，使得可以同时向两个节点写入数据块，实验结果表明 MBR 编码与对应容错级别的 RAID5、RAID6 有类似的写性能。

（3）研究了 MBR 编码的单节点修复性能。通过对比实验，证明了 MBR 编码有着优异的修复性能，而且在修复时的带宽消耗低，达到了理论修复带宽的下界。

总结起来，我们可以得出如下结论：MBR 编码适宜于带宽资源有限且单节点失效频繁的场景。MBR 编码可以看作是以存储开销换取低带宽开销，但是相较于广泛使用的三副本，MBR 编码仍有着较低的存储开销。进一步的工作：

（1）本文的主要工作是针对单节点失效的修复场景，如果有多个结点失效，该如何修复。

（2）如何进一步提升系统的并行性，比如修复时可以将数据块分片，以更小的粒度进行修复，从而使得各个分片的修复过程可以同时进行。

致谢

硕士生涯转瞬即逝，回首走进学校的那天，仿若昨日。华科是一所很大的学校，不仅学校大，还有很多大师。两年来我从众多老师、同学身上学到了很多，正是看到他们如此优秀，我也因而有了加入他们同列的愿望。值此毕业之际，我衷心地感谢所有关心和帮助过我的老师、同学、家人和朋友们。

首先，要特别感谢我的导师胡燏翀副教授。胡老师以其严谨的治学态度、深厚的理论水平、务实的工作作风，引导我一步步走进学术的殿堂，令我获益匪浅。论文的开题、选题、定稿，胡老师都给予了悉心指导。生活上，胡老师总是以春风沐雨般的态度真诚地同我沟通，记得第一次跟胡老师交流时，我说我的动手能力不是特别强，胡老师说没关系，只要肯下功夫、愿意学就不怕。胡老师的鼓励，总能给我继续探索的勇气。在此，向辛苦培育教导我的胡老师表示最诚挚的感谢和敬意。

其次，感谢读研期间计算机学院以及实验室的各位老师的辛勤教导和培养。他们求真务实的教学态度以及精彩的讲课内容，使我获益良多。实验室也为我们提供了良好的学习环境，各种学术讲座、论坛、定期年会等大大地拓宽了我的视野。

然后，还要感谢实验室的各位同学给予的帮助。张晓阳、陈文祥、程良锋、左春雪、肖仁智等同学在我的学习过程中给予了很大帮助，在此深表感谢。还要感谢实验室的其他各位积极向上的小伙伴，他们的支持和帮助使我不断成长，他们是刘振池、潘再余、周嘉伟、薛愫、李宗纬、方翔、赵振凯、张游、许佳豪、郭灵杰、吴雅锋、李承欣、李春光、黄月、胡蝶、刘旭哲、卢梦婷、徐公明等。

接着，还要感谢我的室友王琪、韩雨、彭立发，他们在生活和学习上给予了我莫大的帮助和支持，非常感谢他们的陪伴。

此外，还要感谢百忙之中参与我的论文评审和答辩的各位老师，你们辛苦了。

最后，我要感谢我的父母和亲人朋友们，是你们的支持与强大的后盾，让我有了继续前行的动力。尤其是父母这么多年来的无私奉献，才让我得以顺利完成学业，谢谢你们。

参考文献

[1] Safaei, Safaei B, Monazzah A M H, Bafroei M B, et al. Reliability side-effects in internet of things application layer protocols. In: 2017 2nd International Conference on System Reliability and Safety (ICSRS). Milan, Italy: IEEE, 2017. 207-212

[2] Milojicic D. Microsoft's Jim Gray on computing's breakthroughs, lessons, and future .Distributed Systems Online, 2004, 5(1): 4.1~4.8

[3] Erevelles, Sunil, Nobuyuki Fukawa, and Linda Swayne. Big Data consumer analytics and the transformation of marketing. Journal of Business Research, 2016, 69(2): 897-904

[4] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 4

[5] Ghemawat S, Gobioff H, Leung S T. The Google file system[J]. ACM SIGOPS Operating Systems Review,2003, vol. 37(no. 5): 29-43

[6] Borthakur D. Hadoop architecture and its usage at facebook. Presented at Microsoft Research, Seattle, 2009

[7] Thusoo, Ashish, et al. Data warehousing and analytics infrastructure at facebook. in: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. Indianapolis, Indiana, USA : ACM, 2010. 1013-1020

[8] Mitra S, Panta R, Ra M R, et al. Partial-parallel-repair (PPR): a distributed technique for repairing erasure coded storage. in: Proceedings of the eleventh European conference on computer systems. London, United Kingdom : ACM, 2016

[9] Shum, Kenneth W., and Yuchong Hu. Exact minimum-repair-bandwidth cooperative regenerating codes for distributed storage systems. in: 2011 IEEE International Symposium on Information Theory Proceedings. St. Petersburg, Russia: IEEE, 2011

[10] Xie, Guojun, Jiquan Shen, and Huanhuan Yang. Generalized RDP Code Based Concurrent Repair Scheme in Cloud Storage Nodes. Information 2019

[11] Li, Runhui, et al. Repair pipelining for erasure-coded storage. in: 2017 Annual Technical Conference (ATC'17). Santa Clara, CA , USA: USENIX Association, 2017. 567-579

[12] The Hadoop project, http://hadoop.apache.org/

[13] Xia M, Saxena M, Blaum M, et al. A Tale of Two Erasure Codes in HDFS. in: 13th USENIX Conference on File and Storage Technologies (FAST'15). Santa Clara, CA, USA: 2015. 213-226

[14] Weil S A, Brandt S A, Miller E L, et al. Ceph: A scalable, high-performance distributed file system. in: Proceedings of the 7th symposium on Operating systems design and implementation. Seattle, Washington,USA: USENIX Association, 2006. 307-320

[15] Acquaviva L, Bellavista P, Corradi A, et al. Cloud Distributed File Systems: A Benchmark of HDFS, Ceph, GlusterFS, and XtremeFS. in: 2018 IEEE Global Communications Conference (GLOBECOM). Abu Dhabi, United Arab Emirates: IEEE, 2018. 1-6

[16] Reis G A, Chang J, Vachharajani N, et al. SWIFT: Software implemented fault tolerance. in: Proceedings of the international symposium on Code generation and optimization. Washington, DC, USA: IEEE Computer Society, 2005. 243-254

[17] Doerner D. Distribution and replication of erasure codes: U.S. Patent 9,450,617[P], 2016

[18] 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] Reed I S, Solomon G. Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics, 1960, 8(2): 300-304

[20] HDFS-RAID, https://wiki.apache.org/hadoop/HDFS-RAID

[21] Huang C, Chen M, Li J. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. ACM Transactions on Storage (TOS), 2013, 9(1)

[22] Kralevska K, Gligoroski D, Jensen R E, et al. Hashtag erasure codes: From theory to practice. IEEE Transactions on Big Data, 2018, 4(4): 516-529

[23] Pamies-Juarez L, Blagojevic F, Mateescu R, et al. Opening the Chrysalis: On the Real Repair Performance of MSR Codes. in: 14th USENIX Conference on File and Storage Technologies (FAST'16). Santa Clara, CA, USA: USENIX Association, 2016. 81-94

[24] Li J, Li B. On data parallelism of erasure coding in distributed storage system. in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). Atlanta, GA, USA: IEEE, 2017. 45-56

[25] Li J, Li B. Parallelism-Aware Locally Repairable Code for Distributed Storage Systems. in: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). Vienna, Austria: IEEE, 2018. 87-98

[26] Hu Y, Yu C M, Li Y K, et al. NCFS: On the practicality and extensibility of a network-coding-based distributed file system. in: 2011 International Symposium on Networking Coding. Beijing, China: IEEE, 2011. 1-6

[27] Narayanamurthy S. Modern erasure codes for distributed storage systems[C], SNIA, Santa Clara. 2016

[28] Elias P, Feinstein A, Shannon C. A note on the maximum flow through a network[J]. IRE Transactions on Information Theory, 1956, 2(4): 117-119

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

[30] 胡燏翀. 基于网络编码的分布式存储容错机制研究. 中国科学技术大学, 2010

[31] Dimakis A G, Godfrey P B, Wu Y, et al. Network coding for distributed storage systems[J]. IEEE transactions on information theory, 2010, 56(9): 4539-4551

[32] Rashmi K V, Shah N B, Kumar P V, et al. Explicit construction of optimal exact regenerating codes for distributed storage. in: 2009 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton). Monticello, IL, USA: IEEE, 2009. 1243-1249

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

[34] FUSE: Filesystem in Userspace. http://fuse.sourceforge.net/

[35] 吴一民,刘伟安.基于 Fuse 的用户态文件系统的设计.微计算机信息,2010,26(06):159-168

[36] Goparaju S, Fazeli A, Vardy A. Minimum storage regenerating codes for all parameters. IEEE Transactions on Information Theory, 2017, 63(10): 6318-6328

[37] Tamo I, Ye M, Barg A. The repair problem for Reed-Solomon codes: Optimal repair of single and multiple erasures with almost optimal node size. IEEE Transactions on Information Theory, 2018

[38] Rawat A S, Koyluoglu O O, Vishwanath S. Progress on high-rate MSR codes: Enabling arbitrary number of helper nodes. in: 2016 Information Theory and Applications Workshop (ITA). La Jolla, CA, USA: IEEE, 2016. 1-6

[39] Hu Y, Lee P P C, Zhang X. Double regenerating codes for hierarchical data centers. in: 2016 IEEE International Symposium on Information Theory (ISIT). Barcelona, Spain: IEEE, 2016. 245-249

[40] Ye M, Barg A. Explicit constructions of high-rate MDS array codes with optimal repair bandwidth. IEEE Transactions on Information Theory, 2017, 63(4): 2001-2014

[41] Li J, Tang X, Tian C. A generic transformation to enable optimal repair in MDS codes for distributed storage systems. IEEE Transactions on Information Theory, 2018, 64(9): 6257-6267

[42] Sohn J, Choi B, Moon J. A class of MSR codes for clustered distributed storage. in: 2018 IEEE International Symposium on Information Theory (ISIT). Vail, CO, USA: IEEE, 2018. 2366-2370

[43] Mahdaviani K, Mohajer S, Khisti A. Product matrix MSR codes with bandwidth adaptive exact repair. IEEE Transactions on Information Theory, 2018, 64(4): 3121-3135