分 类 号 _____ 学号 M201072422

学校代码 10487 密级 _____

华中科技大学

硕士学位论文

分布式文件系统存储效率优化研究

学位申请人: 王 敬 轩

学 科 专 业: 计算机系统结构

指 导 教 师: 冯 丹 教授

答 辩 日 期: 2013 年 1 月 22 日

A Thesis Submitted in Partial Fulfillment of the Requirements
For the Degree of Master of Engineering

Storage Efficiency in Distributed File System

Candidate : Wang Jingxuan

Major : Computer Architecture

Supervisor : Prof. Feng Dan

Huazhong University of Science and Technology

Wuhan, Hubei 430074, P. R. China

Jan., 2013

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

随着当今互联网环境中信息呈爆炸式的增长，人们对信息的存储要求已经变得越来越高。而大型互联网公司内部的分布式文件系统的集群规模更是急剧扩张，存储成本也在不断上升。为了保证数据可靠性，现有分布式文件系统通常使用完全数据冗余的方法对文件进行容错，其中采用最多的方法为三副本备份方法。但是三副本备份方法给系统带来了相当大的空间消耗，存储效率低下，所以针对分布式文件系统的存储效率进行优化越来越成为研究中的热点。

纠删码技术通过对文件编码生成校验来达到容错目的，能够极大地提升分布式文件系统的存储效率。但是由于纠删码方法需要对文件进行编码解码操作，并且在系统中仅保存了文件的一份副本，对系统性能带来了影响。因此针对此问题，提出一种基于冷热数据区分的混合存储方案，在提高分布式文件系统存储效率的同时，将纠删码技术对系统性能的损耗降低到最小。

所提出的方案结合了三副本备份方法和纠删码方法的优点，将新写入的数据以三副本备份方法存储，保证系统性能，当数据变为冷数据时，对文件进行编码转换，并减少副本数，节约存储空间。在编码过程中系统参考了 RAID（Redundant Array of Independent Disk）中“条带化”的思想以一个条带做为编码单元。

测试结果表明，此存储效率优化方案相对于三副本备份方法在存储效率方面提升了约 25%，并且此方案将纠删码技术所带来的系统性能损耗由 50%降低至 5%以内，其性能表现与三副本备份方法基本一致。测试说明了基于冷热数据区分的存储效率优化方案确实集合了多种方案的优势，并在存储成本、数据可靠性和系统性能之间达到了平衡。

关键词：存储效率，分布式文件系统，纠删码，可靠性

Abstract

With the rapid increasing data produced by users, the storage pressure for distributed file systems has never been so challenging. Major company usually uses cluster systems to store massive amounts of data, and since the cluster scale is growing up to thousands of nodes, the cost for data storage is becoming more and more expensive. To achieve high reliability, distributed file system usually uses triplication, but it wastes a lot of disk space and has low storage efficiency, so the optimization of storage efficiency is becoming a hot area of research.

To improve storage efficiency, this paper provides a storage strategy based on erasure code. Through the parity blocks, erasure code can ensure the data reliability with only one copy of the source data. But the disadvantages of this approach are the extra time of encode process and decode process, and the delay time when large amount of concurrent clients access the only copy of source data at the same time.

In order to reduce the effect of erasure code's latency, the strategy only encoding cold data in the system. Since the new data are most likely to be the hot data, we use triplication to store the file. When the data has been determined to be cold, it will be coded to generate parity blocks and reduce the number of source blocks to only one copy. In other words, this strategy combined the strengths of both triplication and erasure coding. And the encoding process with an independent unit called "striping".

The result shows this strategy has upgraded the storage efficiency and fault-tolerant capability of distributed file system. And the average client response time is similar to the system that uses triplication. In conclusion, this erasure code based storage strategy achieved balance in storage cost, data reliability and system performance.

Key words： Storage Efficiency, Distributed File System, Erasure Code, Reliability

目 录

1 绪论

1.1 课题背景

随着今天人们的生活向着信息化的方向积极不停地发展，信息的产生正在呈现爆炸式的增长，所以人们对信息的存储要求已经变得越来越高，特别是最近几年，各大互联网公司的业务数据量翻倍的增长。越来越庞大的数据存储压力使得传统意义的文件系统已经不能满足现有上层应用对其在容量上的要求。如何对互联网中海量的数据进行快速、高效、安全的存储是当前互联网发展的最大挑战之一，所以分布式存储技术应运而生。

分布式文件系统相较于传统的文件系统具有以下优势：1. 可扩展性。分布式文件系统由于可以随着机器数的增加而平滑的扩展，因此其存储能力也能够随之增长；2. 可靠性。使用冗余备份可以提供数据的容错，保证其在磁盘损坏或者机器宕机的情况下不会丢失；3. 高吞吐率。使用数据分散技术保障数据批处理时的高吞吐率，且系统能提供给更多用户使用；4. 系统成本低。一般分布式文件系统对计算机设备没有性能和硬件上的要求，只需要将闲置的存储设备整合起来就可以提供满足业务需求的服务，所以不需要购置专用的服务器和路由设备。

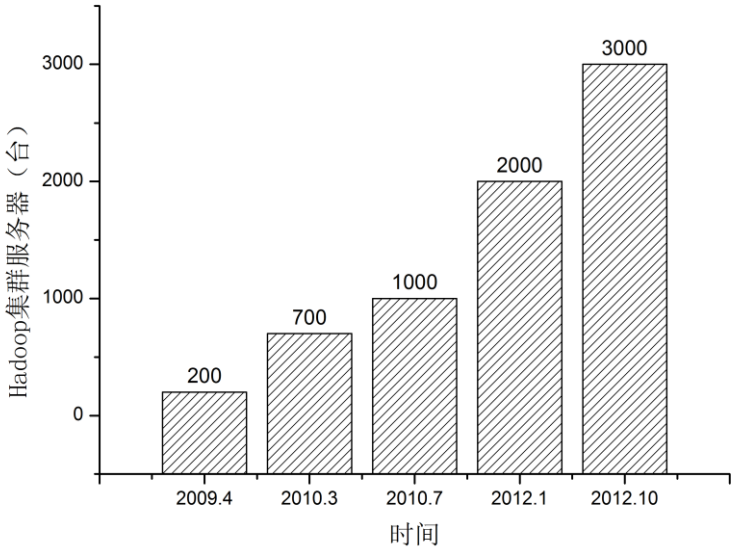

图 1.1 阿里巴巴 Hadoop 集群规模

鉴于以上分布式文件系统的优点，各大互联网公司也根据业务需要，在公司内部建立起自己的分布式文件系统集群。比如阿里巴巴公司搭建了由 3200 台服务器组成的 Hadoop 集群，集群中运用了大约 36000 块磁盘，存储的数据总量达到 60PB，每天的扫描数据量为 7.5PB，扫描文件数为 4 亿[1]，具体集群规模发展如图 1.1 所示。而作为世界上最大的社交网络公司脸谱（Facebook），其公司内部也搭建了自己的分布式集群[2]，该集群包含了超过 3000 台服务器，超过 30PB 的有效逻辑数据，并且存储的数据在 2011 年的一年之内就增加了 10PB[3]。

由这些数据可以看到，当今大型互联网公司内部的分布式文件系统集群磁盘容量在不断扩大，数据规模更是急剧扩张，集群规模也在不断增长。对于大型规模的分布式文件系统集群来说，节点故障的机率也随之提升了，所以保护数据的技术研究也变得更加重要。为了取得较高的数据可靠性，并且要求磁盘和网络 I/O 尽可能的高，现在分布式文件系统中通常使用的数据冗余容错的方法主要为完全数据备份和磁盘阵列。比如 GFS[4]（Google File System）和 HDFS[5]（Hadoop Distributed File System）中都采用了每个文件保存 3 个副本的形式，这种通过牺牲磁盘空间来换取数据可靠性的方案是具有优势的。因为随着磁盘的成本不断下降，采用副本无疑是一种简单、可靠、有效且实现难度也最小的方法，并且文件副本越多，系统的可用性越好，可靠性也越高，但随着分布式文件系统规模和数据量的扩大，完全数据备份带来了相当大的带宽消耗和空间消耗。磁盘阵列技术也是通过冗余来提供数据的可靠性保证，并且通过将多个独立的磁盘组织成为一个统一的逻辑盘，提供了更大的存储容量。但是早期 RAID 技术至多只能容忍一个或者两个磁盘出现故障，当系统规模较大时，数据的可靠性会迅速降低。

近年来，有关基于纠删码的存储效率优化研究在分布式文件系统中越来越成为研究热点[6-10]。本文将在此基础上研究实验室自主研发的分布式文件系统 RaccoonFS 中存储效率的优化方案。

1.2 国内外研究现状

分布式文件系统在短时间的快速发展中，发展出了多种系统原型，其中最为著名并为大家广泛接受的系统为谷歌公司在 2001 年开发的 GFS[4]，在此基础上，Apache 基金会仿照 GFS 的架构在 2004 年开始开发开源的分布式文件系统 HDFS[5]。

1.2.1 现有分布式文件系统中的存储效率

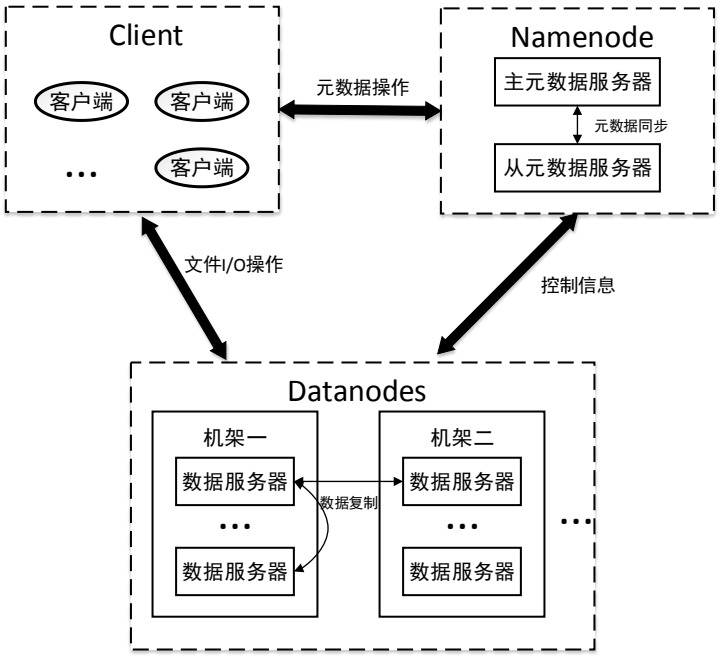

图 1.2 HDFS 系统架构图

在 HDFS 文件系统中，数据的可靠性是通过完全数据备份的方式来保证的。每份数据在系统中保存了三个以上的备份，当有节点失效时，通过数据恢复可将副本数目控制在三个以上，并且可容忍两个节点同时故障。HDFS 在架构上仿照了 GFS，主要通过 NameNode、DataNode 和 Client 端的三方架构来管理系统数据，系统架构图如图 1.2 所示。其中 NameNode 主要负责管理文件系统的命名空间、集群的配置、和存储块的复制。DataNode 主要负责管理文件系统的文件存储，它将文件块存储在本地文件系统中，保存了所有数据块的信息，同时周期性的把所有的信息发送给 Namenode。Client 为上层应用程序提供系统访问接口。

HDFS 将每个文件组织成一组数据分块（Blocks）来存储，其中默认块大小为 64M。使用数据块的好处为：1. 使系统能够存储大文件，单个文件大小可以超过一块硬盘的总容量；2. 将系统中最小存储单元简化为数据块，使系统中数据更加便于管理；3. 数据复制更加可靠方便，当文件中某个数据块丢失时，可以只复制丢失的数据块而不是整个文件。对于每个数据块，出于容错的考虑都会存放三个副本。在系统运行过程中，如果有数据块的副本出现丢失或不可恢复等状况，Namenode 会自动将该副本复制到其他 DataNode，从而确保副本保持一定的个数。

采用这种机制能够保障数据的高可靠性，并且技术实现简单，文件创建和读取不

需要编解码操作，读写效率比较高。但需要浪费两倍的额外磁盘空间，带来了巨额的存储成本。这种成本增长对于大规模集群尤其明显，如果按照 1GB 存储空间的成本是 1 元来算，当数据规模是 5TB 时，整个系统多存储一份副本的成本只有 5,000 元。但是如果数据规模到了 5PB 的话，成本就有 5,000,000 元。由 1.1 小节可知现在公司的数据规模已经达到几十 PB 的级别，可见研究存储效率优化的重要性。

存储效率是指在不丢失或丢失部分系统性能的前提下，以尽可能少的空间来存储和管理数据的能力。网络存储行业协会（Storage Industry Networking Association）将存储效率定义[11]为：

存储效率 = （有效存储量 + 空闲存储量） / 原始存储量

这表明存储效率实际上指的是系统中存储有效数据的存储空间在整体系统中所占的比例，在 HDFS 中，由于系统采用三副本策略，存储效率约等于 33%，大量的空间存储着非有效的数据。所以对存储效率进行优化的重点在于提高有效存储量，即在不损失数据可靠性的前提下，尽量减少冗余数据的存储。

由于分布式文件系统多采用廉价的存储设备，存储成本一直都在可接受的范围之内，因此，尽管存储效率优化在磁盘阵列及其他存储系统中被广泛研究，但直到近些年来，有关分布式文件系统中存储效率的讨论才在 Hadoop Summit 和 Hadoop World 中被提起。文献[12, 13]也对如何对分布式文件系统中的存储效率进行优化做了研究。GFS 和 HDFS 也在优化存储效率方面做了尝试。

1.2.2 现有存储效率优化的研究概况

为了保障分布式文件系统中数据的可靠性，必须对存储数据进行冗余备份，这势必会带来磁盘空间的浪费，但是，合理的优化存储效率能够有效的降低存储成本，并且同样保证系统的高可靠性。

现有一种对存储效率进行优化的方案为将磁盘组织为磁盘阵列。而在磁盘阵列中，为解决磁盘错误带来的数据损失，提高可靠性，采用的冗余技术主要有两类[14]：

1. 多路镜像技术。这种技术将数据通过磁盘镜像实现冗余，在独立的磁盘上产生备份的数据。当原始数据访问繁忙时，可直接从镜像拷贝中读取数据。这种技术的一个典型的代表是采用了 2 路镜像的 RAID 1[15]。

2. 纠删码技术。这种技术将 K 个磁盘的数据存储到 N 个磁盘中（N>K），通过使用某种纠删码对一个条带的数据进行编码操作，生成相应的校验信息，操作完成

后系统能够容忍最多 N-K 个磁盘发生节点故障。这种技术的一个典型的代表是采用了 1 个冗余磁盘的 RAID 5[15]。

由此可见，纠删码技术相比于多路镜像技术，优化了磁盘阵列的存储效率，但是却带来了部分的系统性能损失。随着分布式文件系统中数据规模的扩大，原先使用在磁盘阵列中的纠删码也在分布式文件系统中尝试使用。基于纠删码的技术被应用于许多高容错的存储系统中，如 OceanStore[16]，Allmydata[9]。

早期在分布式存储系统中使用的纠删码技术是简单的 N+1 奇偶校验码，能够容忍单个磁盘故障，随着磁盘阵列规模的扩大，磁盘失效的概率也提升了，于是系统的可靠性不能得到保证。在随后的发展中，阵列纠删码的研究成为热点。因为阵列纠删码在编码的过程中使用简单的异或运算，计算复杂度较低，于是效率高于大部分纠删码。阵列纠删码大多能够纠双错[17]，如 EVENODD 码[18]、X 码[19]、B 码[20]、S 码[21]等。在后来的发展中，更多的人开始关注提高阵列纠删码的容错能力，于是出现了能够容忍三个以上节点故障的 R5X0 码[22]、HoVer 码[23]、WERVER 码[24]、STAR 码[25]，在 2010 年，清华大学提出了一种容错能力最高达 15 的 GRID 码[26]。

RS（Reed Solomon）类纠删码是除阵列纠删码之外，另一种在存储系统中应用的纠删码技术。RS 类纠删码的技术最早运用于 RAID 的研究中，采用 RS 类纠删码产生冗余校验块，能够容忍可配置的多个节点失效。RS 纠删码具有很强的纠错能力，根据其编码时生成矩阵的不同可以分为范德蒙码和柯西码两种。目前，在第二代 GFS 系统 Colossus 中就采用了 RS 纠删码[8]。

除此之外，低密度纠删码（LDPC）也在分布式文件系统中得到应用，典型代表有旋风（Tornado）码。例如在 OceanStore 的子系统 Typhoon 中，采用了旋风码和 RS 类纠删码混合使用的方式来实现存储效率的优化工作[27]。但是同时，由于 LDPC 技术在译码过程中具有概率性，所以无法完全保证译码的成功，因此并不完全适用于对数据完整性要求很高的分布式文件系统，而是适合于网络包传送机制中。

可以看到，近些年来，对存储效率优化的研究主要集中在纠删码的技术研究上，通过改进纠删码技术，提高其容错能力和编码解码效率，改善数据恢复时的所需网络 I/O 和时间，来优化分布式文件系统中的存储效率。

1.3 主要工作和论文组织结构

本文充分研究了现有分布式文件系统中存储效率优化的相关技术，以现阶段国内

外研究现状作为背景，提出了对自主研发的分布式文件系统 RaccoonFS 中存储效率优化的方案，设计实现了结合三副本备份方法和纠删码技术的混合存储方案，以保障系统高可靠性和低存储成本为前提，利用对冷热数据的区分存储提高系统响应时间。本文共有五章：

第一章介绍了分布式文件系统存储效率优化的背景以及国内外研究现状，分析了现有分布式文件系统中的存储效率，由此得出研究存储效率优化的意义，然后分析了现有存储效率优化主要有哪些方案以及这些方案的研究重点。

第二章介绍了分布式文件系统在存储效率优化方面面临的几个关键问题，即如何在优化存储效率同时保障系统可靠性，降低存储成本，并且保证系统性能。然后调研了现有的为解决这些问题普遍采用的研究方案，并对这些方案做出了对比。由此提出了 RaccoonFS 系统的设计需求和本系统中存储效率优化的方案。

第三章介绍了 RaccoonFS 系统在存储效率优化上的设计方案和具体实现，具体讨论了方案架构，模块设计和实现等，提出了基于冷热数据区分并结合三副本备份方法和纠删码技术的混合存储方案。

第四章介绍了对 RaccoonFS 系统中存储效率优化方案进行的性能测试，包括测试目的、测试环境、测试过程，对比了不同方案下系统的性能，并对结果进行了详细分析。

第五章对全文进行总结，分析并提出了可以进一步完善的地方。

2 存储效率优化面临的问题及解决方法

随着用户产生数据的爆炸式增长，分布式文件系统进入了发展的黄金时期，从高性能计算到数据中心，从数据共享到互联网应用，分布式文件系统的应用已经渗透到各方各面。在设计 RaccoonFS 系统之前，分布式文件系统中面对海量数据的存储效率优化还存在着一些急需解决的问题。例如如何在减少数据冗余量，提高存储效率的前提下，仍然保持数据的高可靠性；如何在保证了数据的可靠性后尽量降低存储成本；如何在使用纠删码技术的同时保证系统高性能。本章将对这些问题进行分别研究，并讨论现有的解决方案，提出基于现有方案的优化方法。

2.1 面临的问题

在对分布式文件系统 RaccoonFS 的存储效率优化方案进行设计前，首先需研究面临的问题。分布式文件系统的设计主要面临三个问题，如何在系统中实现下面的三个特性：1. 高可靠，2. 低成本，3. 高性能。在分布式文件系统中，高可靠性是基本要求，因此在分布式文件系统对数据进行存储的方法往往在低成本和高性能之间取一个平衡。对于中小规模数据的集群来说，存储成本一般可以忽视，于是往往为了达到高性能对数据做多备份处理来分散客户端请求，舍弃了成本节约。在海量数据面前，基于纠删码的技术则重点集中在研究如何节约存储成本，但减少副本数目和编解码过程影响了系统的整体性能，有时甚至降低了系统可靠性。下面将对分布式文件系统中面临的三个问题作出详细的分析。

2.1.1 节点不可靠

由于分布式文件系统大多部署在廉价的计算机设备上，所以节点服务器的失效要被当做常态来处理，而不能简单的看作是系统异常。

2011 年 4 月 21 日，亚马逊的虚拟主机服务 EC2 发生大规模停机，导致 Foursquare、Quora 等众多网站的服务受到影响。在事故发生后，亚马逊对事故发生原因做了调查。发现 EC2 停机是由于亚马逊在对集群网络做日常维护升级时发生了操作上的失误，导致正常的网络流量切换到了备用的网络，于是造成了备用网络的过载。过载导致备用网络出现问题，于是系统认为备用网络的服务器发生节点故障，并已停止服务，于是为了保障数据的可靠性，系统将“停止服务”的服务器上的数据副本复制到其

他存储资源上，引发了“镜像风暴”。由镜像风暴带来的大量流量又进一步的加重了网络的过载，这样的恶性循环导致故障越发的严重。亚马逊从发现问题到恢复服务总共用了两天半的时间，损失巨大。

由此可见，在分布式文件系统中需要特别关注系统的可靠性。文献[28]对真实环境中磁盘失效的情况作了分析和统计，结论得出在实际应用中，集群中每年的磁盘更换率大约在 2%~4%，最高的统计数据达到了 13%。面对节点失效率如此频繁的系统，若想要保障系统中数据的高可靠性，就需要对数据做冗余备份。

2.1.2 存储成本高

由于采用了分布式的数据存储和数据管理方式，对单个节点的处理能力没有很高的要求，所以谷歌公司在部署 GFS 集群时不需要购买价格昂贵的服务器和存储设备，而是采用廉价的存储设备和普通硬盘来构建服务器集群，这大大减少了建设投资。并且由于在主流分布式文件系统如 HDFS 中，大部分的节点都是同构的，即存储设备配置相同，所以很容易实现标准化，只需要批量采购专门定制的计算机主板，还可以减少显示器、外设接口等硬件设备，进一步降低设备投资。

但是由于云存储给了人们廉价获取存储容量的能力，于是用户不必去买大容量的存储资源或者租用数据中心的软硬件资源。并且，人们每天都在社交媒体上创造有趣的数据，而宽带的普及也使得人们时刻保持在线状态，随时随地可以产生数据，导致如今海量数据的规模越来越大。

例如百度公司，由于其包含搜索引擎产品，社区产品，媒体产品等多种互联网产品，所以百度存储的数据量大概是数百个 PB，每天处理的数据大约为几十 PB。并且数据规模还在惊人的增加，大概是五年前的 500~1000 倍。于是在惊人的数据量下，存储资源成本就变的很庞大，此时如果只考虑数据可靠性，采用完全数据备份的存储方式，公司只能扩大集群规模，采购大量相关设备，这些设备的累积成本因而变得非常昂贵，随之产生的能源成本、带宽成本和管理成本也一并上涨。例如目前大多数的分布式文件系统中数据是靠三备份（triplication）来保证冗余的，显然这只是一个简单有效的方法而不是一个非常优雅聪明的方法。

2.1.3 性能损失

针对海量的数据存储，分布式文件系统需要提供与数据规模相适应的 I/O 处理速

率和用户响应时间。例如航空领域、医学影像等应用对于存储系统的需求就包含了系统高性能。美国能源部用于核动力和核武器研究的 2006-purple 系统的存储容量为 2.5PB，总带宽达到了 100GB/s，而美国国安局的信息检索系统，对后台存储的压力为每秒生成 32,000 个文件，并且一万亿个文件要求在 10 分钟内检索完成。

高性能计算技术的快速发展和 I/O 密集型应用的不断增多，也使得分布式文件系统越来越多的重视系统性能。由于现在系统中多使用多个 CPU，系统计算能力与 I/O 速度已经越来越不匹配。为了增强系统性能，提高系统吞吐率和响应时间，现有分布式文件系统将访问请求分布到多个节点上去，采用了将数据分散并且移动复制的方法，如图 2.1 所示。

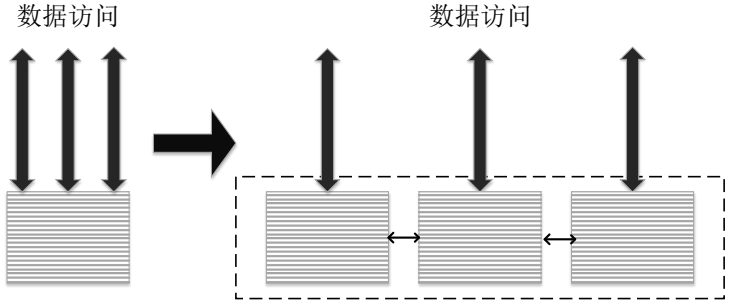

图 2.1 数据分散提高系统性能

在分布式文件系统中，由于交换机的链路带宽限制，多个客户端整体的读取速度有理论极限值。但是在带宽利用方面，有些带宽因为被用于数据重建或者副本迁移，所以有效的数据访问带宽没有达到极限值。如何在不损失系统可靠性的前提下，尽量快速响应有效数据的访问也是存储效率优化中必须考虑的问题。

现有分布式文件系统中的存储效率优化方法由于侧重点不同，分别对上面所述的三个问题的解决做了不同的取舍。其中最为广泛应用的方案有副本备份方法和纠删码方法。下面将对每种方法做详细的说明和对比。

2.2 副本备份方法分析

副本备份方法侧重于保障分布式文件系统的高可靠性和高性能，其具体方案已在 1.2.1 小节中有所讨论，本节不再详述，只是讨论一下该方案是如何保障系统高可靠与高性能的。

假设集群中共有 N 个节点计算机，它们的故障概率都为 P，节点 x 内的故障概率记为 P_x。那么 $P_0=(1-P)^N$，$P_1=P_0+(1-P)^{(N-1)}*P$，$P_x=P_{(x-1)}+(1-P)^{(N-x)}*P^x$。图 2.2 显示了当

N 为 128，P 为 0.01 时不同节点内宕机的概率，可知宕机概率当超过节点数为 3 时增长已经不再明显，即保存三个以上副本所获的可靠性收益趋势减缓，因此三副本备份就能够保障数据的高可靠性了。

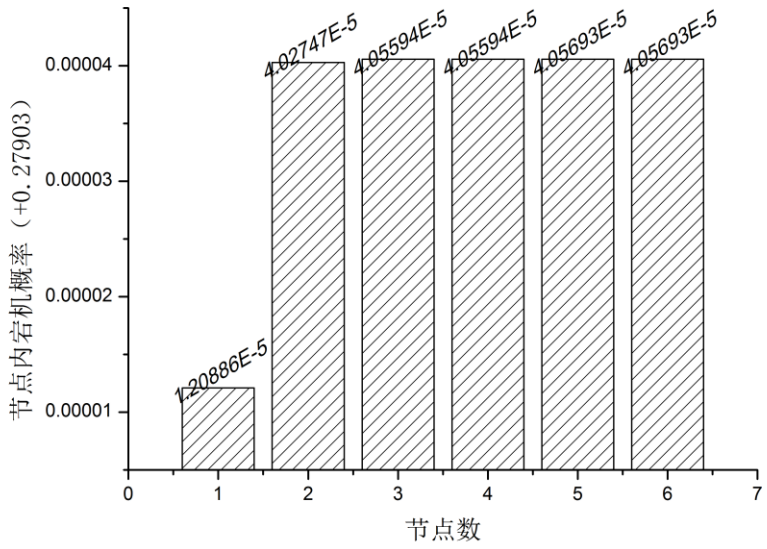

图 2.2 副本备份策略中不同节点内宕机的概率图

另外，由于分布式文件系统在三副本的放置策略上也做了可靠性方面的设计，有一个副本放置在不同于另外两个副本的机架中，在同一个机架中的两个副本也放置在该机架的不同机器上，这样的放置策略避免了机架整体出现故障时的数据丢失。因此使用副本备份方法时将副本数目设置为 3 能够保障系统的高可靠性。

另一方面，由于数据使用了三副本备份方法保存，除了能够对各种软硬件故障进行容错外，还能够提高文件系统数据读服务的能力。因为每个单独的副本都可以提供读服务，并且由于根据负载均衡策略使多个副本分布在不同的网络交换机上，可以充分的利用网络带宽。这样不仅提高了服务能力，也避免了少量数据的高频访问导致的单机访问热点。

2.3 典型纠删码方法分析

纠删码方法的主要思想是对 K 个数据块进行编码，编码后总共产生 K+M 个数据块，从编码后的 K+M 个数据块中任意取出 $K'(K'\geq K)$块后经过解码可以得到原始数据，如图 2.3 所示。

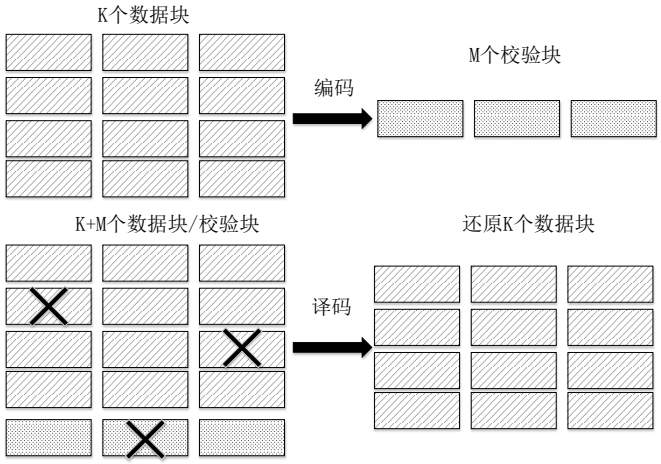

图 2.3 纠删码原理图

一般来说，纠删码可以用一个四元组（n, k, b, k’）来表示[27]，其中，n 是编码后的文件块数，k 是编码前文件块数，b 是每个文件块所含比特数，k’是一个正整数且 k’ ≥k。首先将要存储的数据分为 k 个数据块，每个数据块 b 比特，则编码前文件可以表示为 F = (F_1, F_2, …, F_k)。对原文件编码得到 Encode(F)=(F_1’, F_2’, … F_n’)，文件经过编码得到 n 个数据块，其中 F_i’(1≤i≤n)大小仍为 b 比特。通过 Encode(F)中任意 k’个数据块就可以解码还原文件。本节分析了几种不同类型的当前应用在分布式存储系统中的纠删码技术，主要有阵列纠删码，RS 类（Reed-Solomon）纠删码和低密度纠删码。

2.3.1 阵列纠删码

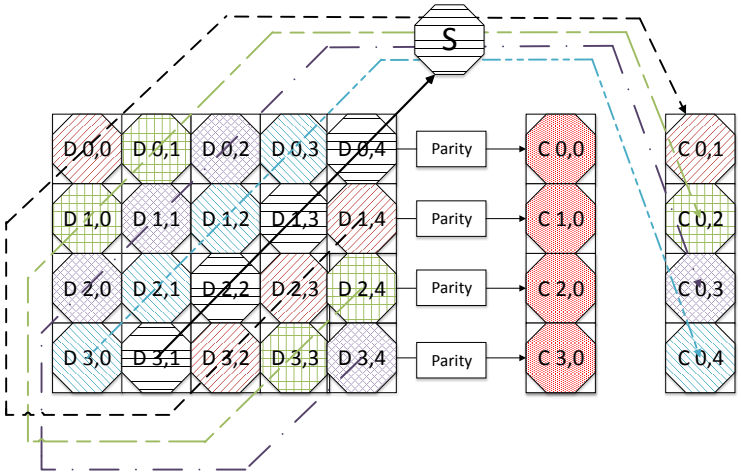

图 2.4 EVENODD 编码原理图

阵列纠删码是一种特殊的线性纠删码，它将数据放在一个二维或多维阵列中，在

编译码计算过程中只需要简单的二进制异或运算，实现比较简单。由于阵列纠删码的编译码复杂度低，软硬件实现简单，并且其结构恰好与磁盘阵列的结构相符，被广泛应用于存储系统中，特别是 RAID 技术中[29]。

EVENODD 码是被公认的阵列码的始祖，是一种纠双错的水平码。码字可以用一个(p-1)*(p+2)大小的二维阵列来描述，其中前 p 列放信息位，后两列放校验位，p 是一个大于 2 的素数。p=5 时的 EVENODD 编码原理如图 2.4 所示。EVENODD 码能够容忍两个节点故障。

从图 2.4 中可以看出，EVENODD 的编码只需要经过简单的异或运算，编码的构造具有几何特性，两列校验列分别是通过斜率为 0 和斜率为 1 的直线所经过的信息位经过异或操作得到的。在两个节点同时失效的情况下，EVENODD 的解码过程会先解码出只丢失了一个信息位的某一个对角线上的信息，然后解码出只丢失了一个信息位的某一行上的数据，重复此过程可以解码得出所有丢失的数据。

但由于第二列校验列的计算需要一个共同的异或运算，所以一个信息位有可能被用到多个校验位的计算，当这种情况发生时，数据解码和数据更新都没有达到最佳性能。

2.3.2 RS 类纠删码

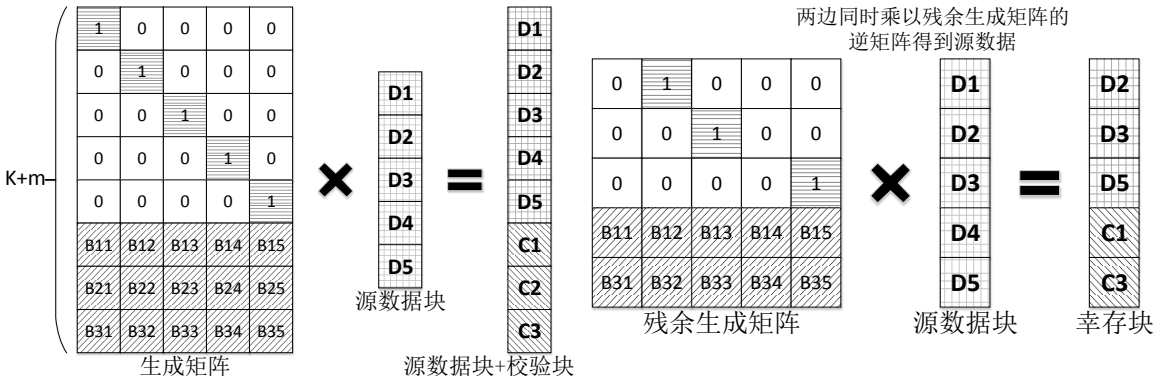

图 2.5 RS 类纠删码编解码原理图

RS 编码是唯一可以满足任意的数据磁盘数 k 和冗余磁盘数 m 的 MDS (Maximum Distance Separable)编码方法。RS 编码是在 Galois 域 GF(2^w)上进行所对应的域元素的多项式运算的编码方式[30]。1997 年，文献[31, 32]中提出了一种 RS 类纠删码的编译码软件实现算法，RS 编码通常分为 2 类：一类是范德蒙 RS 编码(Vandermonde RS codes)[33]；另一类是柯西 RS 编码(Cauchy RS codes)[34]。

RS 类纠删码的生成矩阵 G 为一个 k*(k+m)的矩阵，并且该生成矩阵由两部分构成 $G = (I_{k*k}|P_{k*m})$，其中 I_{k*k} 是一个 k 阶的单位矩阵，矩阵 P_{k*m} 根据编码方式的不同而不同，范德蒙 RS 编码使用的矩阵是范德蒙矩阵，而柯西 RS 编码所用的矩阵则为柯西矩阵。

RS 类纠删码的编解码原理如图 2.5 所示。其编码原理是利用生成矩阵与数据列向量的乘积来计算得到校验信息列向量的。而其解码原理是利用没有发生故障的信息所对应的残余生成矩阵的逆矩阵，乘以没有发生故障的校验信息列向量来恢复原始数据的[35]。但是在 RS 进行解码的过程中，限制条件为存活信息所对应的残余生成矩阵在 $GF(2^w)$域上必须可逆，只有满足此限制条件 RS 类纠删码才能进行成功解码。即对于任意的 k 个元素(剩下的 k 个存活的元素)，对应的残余生成矩阵都要在 $GF(2^w)$域上可逆，这样的高要求导致了 RS 类纠删码的生成矩阵需要满足很高的条件。

1. 范德蒙 RS 码：

范德蒙矩阵的定义为：形如下面形式的矩阵称为范德蒙矩阵。

$$\begin{bmatrix} 1 & a_2^0 & \cdots & a_n^0 \\ a_1 & a_2 & \cdots & a_n \\ a_1^2 & a_2^2 & \cdots & a_n^2 \\ \cdots & \cdots & \cdots & \cdots \\ a_1^{n-1} & a_2^{n-1} & \cdots & a_n^{n-1} \end{bmatrix}$$

利用范德蒙码对文件编码生成校验块的具体过程如下：将文件分为 n 个数据块，而 m 个校验块则通过有限域（又称为伽罗瓦域）运算得到。首先需选取有限域范围，即确定 w 值，$GF(2^w)$的范围是 $0\sim2^w-1$ 的整数。把每个数据块都分为字进行处理，字的长度是 w bits，将一个文件的字都读入到一个矩阵中，每个字的内容组成矩阵的一个元素。有限域内的加法和减法都是异或运算，乘法和除法使用的是多项式相乘除并模除基本多项式。

在做有限域运算时，乘法运算是将矩阵元素的二进制形式先转化为多项式的形式，然后进行多项式的乘法运算，再将结果对本原多项式（例如 $GF(2^4)$中本原多项式为 x^4+x+1）求余，最后再把结果转化为二进制的形式。当 w 较小的时候，可以使用两个对数表加快有限域乘法运算的速度，除法运算亦然。

对范德蒙矩阵进行初等变换，将其前 n 行变成一个单位矩阵，就得到满足 RS 编码要求的生成矩阵。基于这个生成矩阵的 RS 编码就是范德蒙 RS 编码，形如下面形式。但是在 $GF(2^w)$域上，复杂的乘法运算和矩阵求逆运算使得范德蒙 RS 编码的性

能不理想。

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & n \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & 2^{m-1} & 3^{m-1} & \cdots & n^{m-1} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}$$

2. 柯西 RS 码：

Blomer 在 1995 年提出了另一种 RS 码[36]，他使用了柯西矩阵来代替范德蒙矩阵。设$\{x_1, x_2, \dots, x_k\}$，$\{y_1, y_2, \dots, y_{n-k}\}$为有限域 $G(p^r)$（p 是素数）的两个元素集，如果满足：

a) 对于任意 $i \in \{1, 2, 3, \dots, k\}$和任意 $j \in \{1, 2, 3, \dots, n\}$，有 $x_i + y_j \neq 0$；

b) 对于任意 i 和 $j \in \{1, 2, 3, \dots, k\}(i \neq j)$，有 $x_i \neq x_j$；对于任意 i 和 $j \in \{1, 2, 3, \dots, n\}(i \neq j)$，有 $y_i \neq y_j$，那么下面形式的矩阵即为柯西矩阵。

$$\begin{bmatrix} \frac{1}{x_1 + y_1} & \frac{1}{x_1 + y_2} & \cdots & \frac{1}{x_1 + y_n} \\ \frac{1}{x_2 + y_1} & \frac{1}{x_2 + y_2} & \cdots & \frac{1}{x_2 + y_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{x_{m-1} + y_1} & \frac{1}{x_{m-1} + y_2} & \cdots & \frac{1}{x_{m-1} + y_n} \\ \frac{1}{x_m + y_1} & \frac{1}{x_m + y_2} & \cdots & \frac{1}{x_m + y_n} \end{bmatrix}$$

柯西 RS 码在以下两个方面做了改进：1. 使用了柯西矩阵使得编解码复杂度降低；2. 使用异或运算来代替基于 Galois 域的乘除运算，大大提高了运算效率。柯西 RS 码之所以能够使用异或运算是因为它将(n+m)*n 的矩阵在 $GF(2^w)$范围内转化为了 w(n+m)*wn 的矩阵，使得每个矩阵元素为 1 或者 0。

2006 年 Plank 提出了一种对柯西 RS 码的改进方法[34]，通过对柯西分布矩阵的优化，使得编码性能提升了 10%。

2.3.3 低密度纠删码

旋风码（Tornado）是一种典型的低密度纠删码（Low Density Parity Check Codes），

并且是一种基于稀疏矩阵的码，本节以旋风码为例介绍 LDPC 码。旋风码每个校验节点都由两个或更多源数据节点通过异或操作得到，其对应的规则由稀疏矩阵中“1”的个数和位置决定，旋风码的编码原理如图 2.6 所示。

由于旋风码的生成矩阵是稀疏矩阵，所以校验块的编码效率和源数据块的个数成正比。因此旋风码的编码效率是线性的，而当源数据块的数据丢失时，可以通过校验块来恢复。可是如果其对应的校验块的数据也同时发生节点故障，则源数据不可恢复[37]。

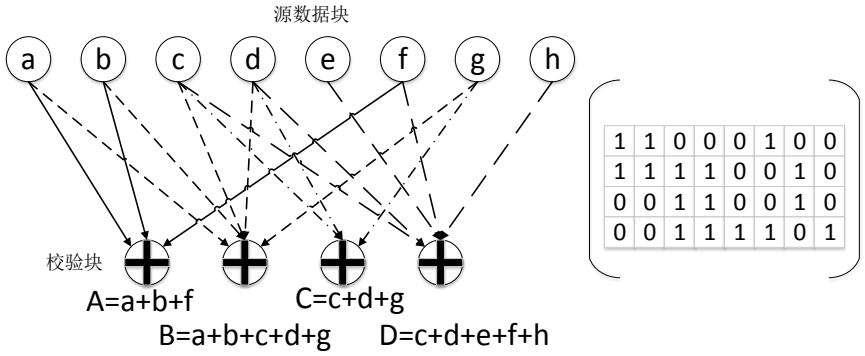

图 2.6 旋风码编码原理

旋风码为了得到更快的解码速度，在回复数据包时需要略大于源数据包个数 k 的数据，所以在数据恢复时有一定的带宽损耗，并且由于其不能保证一定解码成功，具有一定的风险性。

2.4 现有解决方法的对比

通过上面的分析，得知存储系统中对文件的存储可以使用三副本备份方法或者纠删码方法。在存储系统中 RAID 技术也经常被用来对数据进行存储，RAID 方法是纠删码技术的一种应用。

RAID 技术一般分为硬件 RAID 和软件 RAID。硬件 RAID 重点考虑了系统的高性能和高可靠，软 RAID 则考虑了系统的低成本和高性能。但传统 RAID 方法在分布式文件系统中并没有得到广泛的应用，因为硬 RAID 所需的设备价格昂贵，并且无法适应异构的存储设备。而软件 RAID 的安全性不好，当有一块硬盘损坏时，不能实现重建的功能，并且无法满足上层应用负载均衡的要求。而且随着磁盘容量的不断扩大，RAID 技术的数据恢复时间也变得越来越长，往往需要几个小时的时间，有时甚至需要几天。因此传统 RAID 方法并不适合在分布式文件系统中使用。表 2.1

重点比较了三副本方法和纠删码方法的优势和劣势。

表 2.1 三副本和纠删码方法比较

	三副本	纠删码
存储成本	较高（3X）	经济（<3X）
容错能力	2	可配置（>2）
实现复杂度	简单	较复杂
系统性能	相对无延迟	需编码解码

由于对于中小规模数据的集群来说，存储成本可控，所以多采用三副本策略进行数据存储，如 GFS 和 HDFS。但是在海量数据面前，在相同容错能力下，纠删码技术大大减小了存储成本，并且能够保证系统可靠性。于是谷歌（Google）在第二代 GFS(即 Colossus)系统[8]中使用了 RS 类纠删码实现了成本更低的可靠存储。微软（Microsoft）的 Azure 平台[7]也使用了类似的纠删码技术来降低存储成本。脸谱（Facebook）在开源 Hadoop 的基础上也实现了一套基于纠删码的 RAID 方案。纠删码技术已经开始在分布式文件系统领域得到更多的应用，文献[38-40]都分别对纠删码方法在分布式文件系统中对存储效率的优化做了研究，但是由于仅仅使用纠删码方法会对系统性能带来影响，文献[41]对如何减小纠删码方法的系统性能影响做了探讨，其方法是在纠删码的编解码方式上做了改进。回顾 HDFS 和 GFS 采用三副本策略的优势，除了容错以外三副本还能够提高文件系统数据读服务的能力，充分的利用网络带宽，避免出现单机访问热点。

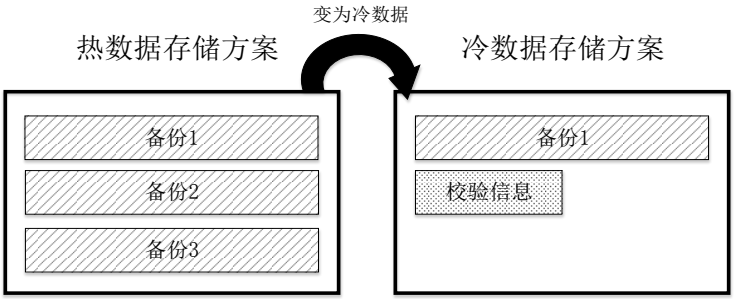

图 2.7 冷热数据混合存储方案

综合考虑三副本备份方法和纠删码方法的优点，为了不影响大部分客户端的响应时间，本文提出了一种基于纠删码技术的存储效率优化方法。即只有当数据变为冷数据时，才对文件进行编码转换，节约存储空间。对于系统中的热数据，仍然采用三副本策略，保障系统性能，如图 2.7 所示。由于新上传的文件数据往往为热点数

据，所以本文提出的方案在新数据写入时采用流水线的方式保存源文件的三份副本，当数据被判定为冷数据时，系统会将数据用纠删码技术进行编码存储，同时将原文件在系统中的副本数降为 1 份。另外，对于一些小的文件（Blocks 数目较少），本方案并不会为其计算校验块，而是仍然采用三副本备份方法存储，因为当文件较小时，副本方案与纠删码方案的存储成本开销相差不大，并不能起到优化存储效率的作用，反而降低了文件并行服务的能力，增加了 block 丢失时的恢复开销。

另外，在对冷数据的编码形式上借鉴了传统 RAID 的"条带化"（stripe）思想，以一个条带为独立的编码校验单元，编解码都以条带为单位。

2.5 本章小结

本章详细论述了在分布式文件系统中对存储效率进行优化所面临的挑战，比如节点不可靠、存储成本高和性能损失这几个关键问题，并且对这些问题分别进行了分析与研究。同时研究了业界普遍采用的一些现有解决方法。比如三副本备份方法重点考虑了分布式文件系统的高可靠和高性能，纠删码方法则重点考虑了低成本和高可靠。经过详细的分析，为了结合各方法的优势，提出了基于冷热数据区分的混合存储方案。即新数据以三副本备份方法存储，当数据变为冷数据后，采用纠删码方法存储，并在编码过程中参考 RAID 中“条带化”的思想以一个条带做为编码单元。

3 RaccoonFS 系统中存储效率优化方案设计与实现

信息存储与应用实验室自主研发的分布式文件系统 RaccoonFS 主要为解决海量大文件存储，以及非结构化数据的持久化存储等需求而设计，为上层服务提供高吞吐量的数据访问，非常适合大规模数据集上的应用，并且在系统的存储效率方面做了基于冷热数据区分和副本纠删码混合的优化存储方案。

RaccoonFS 系统的上层应用多为“一次写入多次读取”的文件访问模型。一个文件经过创建、写入和关闭之后就不需要改变。这一假设简化了数据一致性问题，并且使高吞吐量的数据访问成为可能。

3.1 架构设计

基于冷热数据区分的存储效率优化方案在 RaccoonFS 系统中实现。RaccoonFS 分布式文件系统的架构采用了 GFS/HDFS 的三方架构。其中三方包括：元数据服务器（NameServer），数据节点（DataServer），客户端（Client），系统架构如图 3.1 所示。

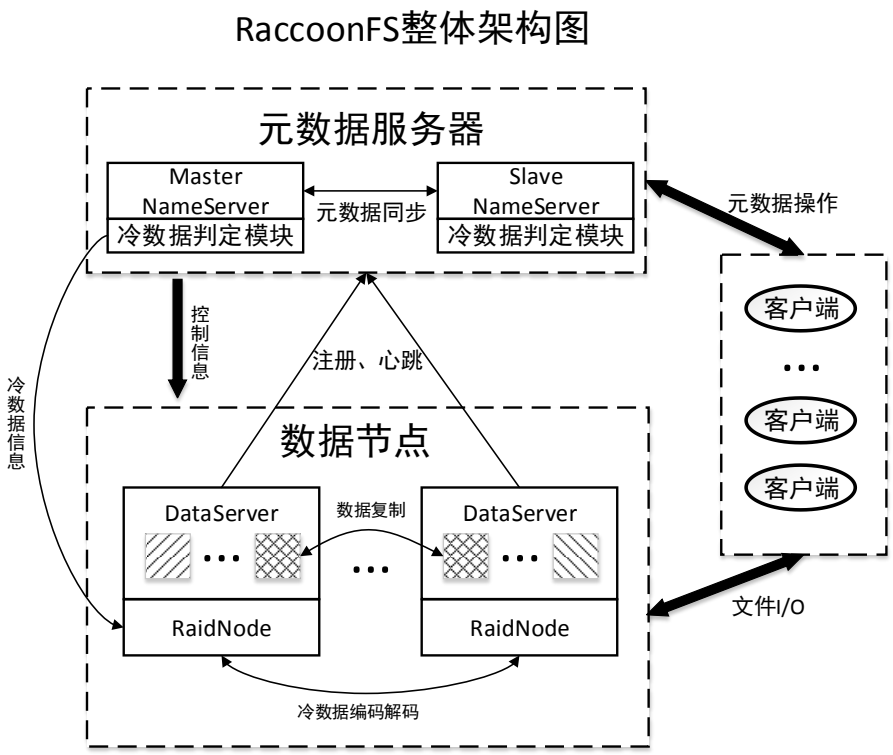

图 3.1 RaccoonFS 分布式文件系统架构图

其中，引入 RaidNode 对系统中的冷数据进行编解码和数据恢复操作。写入新数

据时以三份副本的方式在系统中存储，当数据被判定为冷数据后，RaidNode 会选择在系统负载较轻时（这里默认配置为当天凌晨时间）对冷数据进行编码，并删除系统中多余的副本，优化存储效率。当检测到节点发生故障或数据块发生故障时，RaidNode 需要组织对数据进行恢复，保证系统中数据的高可靠性。

NameServer 是主控节点，存放 DataServer 的节点信息和三级元数据信息，即包括 Filename 到 FileID 的映射，FileID 到 BlockID 的映射，BlockID 到 DataServers 的映射，NameServer 有多台机器热备，防止单点故障。由于存储的是元数据信息，数据量很小，所以本系统中将元数据信息都存放于内存，并定期更新到磁盘。此外，Client 端与 NameServer 端只有控制流（交互 Block 位置信息），没有数据流，Client 端通过缓存部分数据元信息，可以不需要每次都去访问 NameServer，因此对于 NameServer 来说，负载是比较小的。主 NameServer 与从 NameServer 拥有共同的虚拟 IP，通过 heartbeat 管理，主 NameServer 失效后，从 NameServer 能立即获得虚拟 IP，并对外提供服务。

DataServer 是真正存储数据的地方，将大文件切割后，按照一定负载均衡策略存放在磁盘。DataServer 是以 Block 为最小单位存储（默认块大小为 64M），每份数据在刚写入时都存储 3 个备份（可配置）。写入时，只有流水线中所有的 DataServer 均写操作成功之后才返回成功，保证强一致性；读取时，每台存储备份的 DataServer 均可提供读取服务，避免单机热点访问。

Client 提供给用户使用，提供 read、write，delete 等操作接口，让分布式文件系统对用户透明。Client 与 NameServer 通信，获得文件 Blocks 信息以及相应 Block 到机器 IP 的映射关系，然后根据获得的元数据信息直接与该 DataServer 交互完成相应的读、写删除操作以及文件的切割合并操作。在读取经过编码的冷数据并且有源数据块发生丢失时，需要校验块参与解码工作，Client 采用了 Swap 的方式将损坏的源数据块替换为对应校验块，完成解码操作。

上面介绍了 RaccoonFS 分布式文件系统的整体框架设计，下面重点介绍存储效率优化方案中涉及到的系统模块设计与具体实现。

3.2 冷热数据区分

RaccoonFS 系统中对存储效率的优化体现在对原有的三副本备份方法做了改进，引入纠删码技术提升系统的存储效率，同时基于冷热数据区分对冷数据和热数据分

别采取不同的存储方式，在提升存储效率的同时将对系统性能的影响降低到最小。方案中由于采用了对热数据和冷数据分别使用三副本备份和纠删码编码的混合存储方法，所以需要对系统中的数据做出合理的冷热判断。

在大部分的应用环境下，对系统中 80%的访问对象都是所存储数据中的固定的 20%，这就是著名的 80/20 规则[42]。80/20 规则说明了对于绝大多数的客户端访问请求都存在时间局部性和空间局部性。Gomez 和 Santonja[42]在 2002 年对三组 I/O 请求做了调查研究，发现在这些请求中对系统中部分数据的访问非常频繁，而有些数据则很少被访问甚至从未被访问。另外文献[43]经过研究发现系统中 90%的对媒体数据的访问是针对固定的 14%~30%的文件的。所以存储系统中的热数据尽管只占所有数据的少部分，但是却占据了绝大多数的访问请求，对于这部分数据如果采用纠删码方法进行存储，将影响绝大部分客户端的访问请求，所以需使用三副本方式存储，当热数据转变为冷数据后，在系统负载较轻的情况下将其转换为纠删码编码后的数据并减少文件副本数。

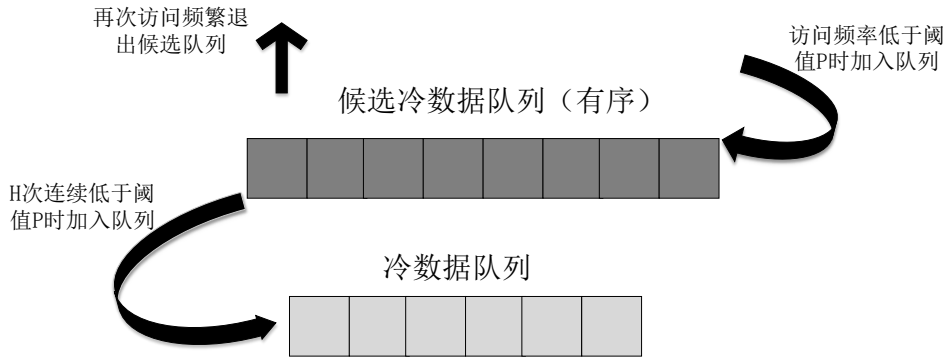

图 3.2 两层冷数据判定队列

本系统中对冷热数据的区分主要从两个属性来判断：1）文件创建时间；2）文件访问频率。文献[42]中对企业中实际工作负载访问做了研究，发现在对一个固定文件的所有访问请求中，51%~52%的访问都来自于文件创建后的第一个星期。所以，本系统中认为新创建的数据有很大的可能成为热数据，因此，新上传的数据被判定为热数据并采用三副本备份方式进行存储。这种判定方式的另一个优势为能够向下兼容早期版本的系统和其提供的 API。

鉴于新上传的数据采用热数据存储方式进行储存，那么需要对热数据何时变为冷数据进行判断，本系统中借鉴了文献[44]中提出的运用在闪存中的两层 LRU（Least Recently Used）队列，使用了一个候选冷数据有序队列和一个冷数据队列来对冷数据

进行判断，如图 3.2 所示。

使用两层冷数据判定队列能够简单而有效的对系统中的冷数据加以判定，冷热文件元数据信息在系统中通过 B+树来进行管理，文件节点的元数据结构如表 3.1 所示。

表 3.1 RaccoonFS 中文件的元数据结构

file ID	文件标识符
file type	文件类型
replica number	文件的副本数目
mtime	文件的修改时间
crtime	文件的创建时间
chunk count	文件包含的数据分块个数
file size	文件的大小
access count	文件当日的访问次数

其中，file ID 表示文件在系统中的唯一标识符，标识符使用一个 64 位整型数表示；file type 对文件的类型进行了区分，在系统中文件总共分为四种类型：未初始化文件、目录文件、未编码文件、编码文件；replica number 的默认值为 3，当文件变为编码文件后，默认值为 1；mtime 和 crtime 分别表示该文件的修改时间和创建时间；chunk count 表示该文件包含的数据分块的个数；file size 表示该文件的大小；access count 表示文件当日的访问次数，当访问次数小于阈值 P 时，将该文件插入候选冷数据队列，每天检查完文件当日访问次数后，该值将被重置为 0。

表 3.2 RaccoonFS 中数据分块的元数据结构

file ID	文件标识符
offset	数据分块在所属文件中的偏移量
chunk ID	数据分块在所属文件中的标识符
chunk version	数据分块的版本号
replica number	数据分块的副本数目
parity	数据分块对应的校验块信息

数据分块的元数据结构如表 3.2 所示。其中，file ID 表示该数据分块属于哪一个唯一的文件；offset 和 chunk ID 分别表示该数据分块在文件中的偏移量和标识符；chunk version 表示数据分块的版本号，通过对比版本号保证一致性；replica number 表示数据分块当时的副本数目；parity 表示该数据分块对应的校验块，当该数据块属

于冷数据时，若该数据分块在 DS 中访问失败，客户端会根据 parity 中的信息访问相应的校验块信息，达到客户端数据恢复的目的。

冷热数据区分模块中包含了两个不定长度的队列，其中候选冷数据队列为有序队列，每个节点包含了 file id 和 count 属性，其中 count 表示该文件在队列中的时间计数。当文件的当日访问量（access count）小于阈值 P（默认为 5 时）时，将该文件插入候选冷数据队列，若候选冷数据队列中已存在该文件，则将该文件对应的时间计数（count）加 1，如果不存在该文件，则将文件插入队列末尾并将初始时间计数记为 1。当 DataServer 做完数据块上报操作后，检查候选冷数据队列中是否存在未进行时间计数加 1 操作的文件，若有则表示此文件当日访问量超过阈值 P，表示该文件仍然被频繁访问或再次成为热点数据，不应再作为冷数据的候选，因此将其从候选冷数据队列中移除。根据此方案，当候选冷数据队列中的文件时间计数超过阈值 H（默认为 7）时，认为该数据已经变为冷数据，将该文件加入冷数据队列，同时将该队列信息发送给 RaidNode 进行冷数据编码操作，该队列中的所有文件将以冷数据存储方案进行存储，方案的具体操作流程如图 3.3 所示。

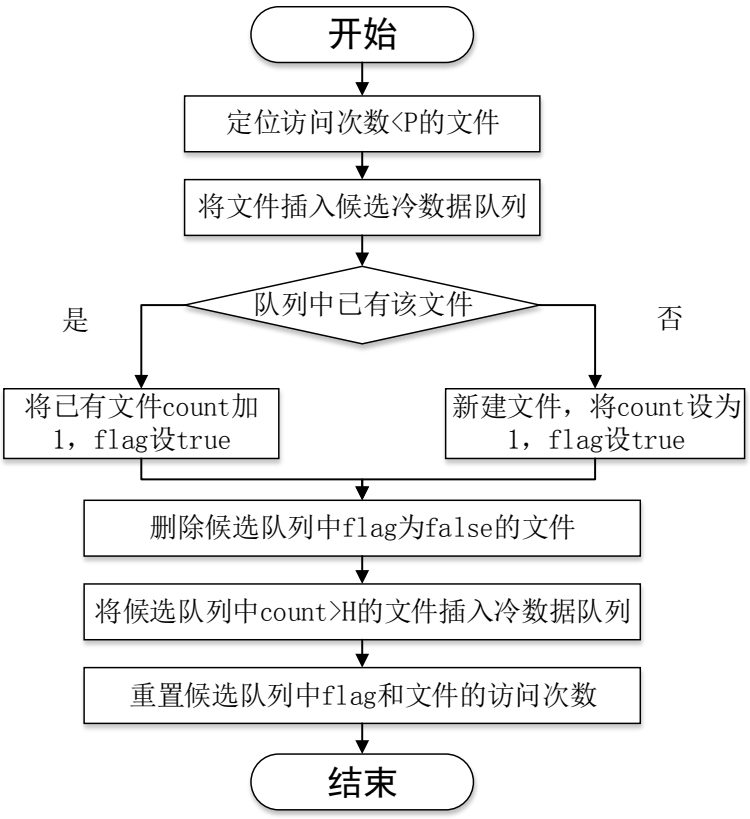

图 3.3 冷数据判定流程

3.3 数据存储与恢复

在对系统中的数据做出冷热区分之后，方案中对不同的数据采用了不同的存储方式，其中冷数据采用了纠删码方法存储，热数据采用了三副本备份方法存储。

3.3.1 冷数据存储

当文件被判定为冷数据后，NameServer 在 DataServer 进行数据块上报操作后，将需要编码的文件信息发送给 RaidNode，其首先收集需编码的源文件数据，将文件编码生成校验块，然后将校验块重新写入系统，并保证校验块与源数据块分别存储在不同的 DataServer 上以增强系统可靠性。最后，RaidNode 通过与 NameServer 通信更新文件元数据信息，将源文件的源数据块相对应的校验块信息加入到元数据结构中，并减少源文件的副本数。

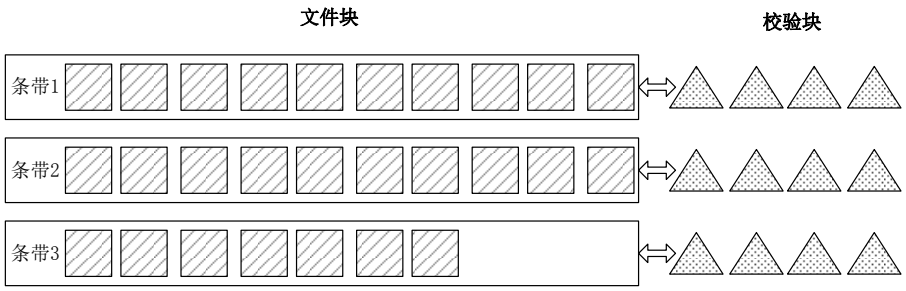

图 3.4 冷数据的条带化编码存储

本方案中对文件的编码在形式上借鉴了传统 RAID 的"条带化"（stripe）思想，将文件每 10 个 Block 作为一个条带进行编码运算，生成 4 个校验块，其中 10 就是条带大小（stripe length）。文件最后不到 10 个 Block 的一组 Blocks 也被看做一个条带，计算生成同样数目的校验块。每个条带是一个独立的编码校验单元，编解码都是以条带为单位的，如图 3.4 所示。

同时，同一条带的 Block（包括校验块）被分散放置在不同的 DataServer 上，这样避免某个 DataServer 发生故障时，影响对这些 Block 的恢复，这个道理和采用冗余备份时不能将同一 Block 的三个副本放在同一个 DataServer 上的道理是一样的。

采用此方案，将冷数据的存储空间从原来的 3 倍降低到了现在的 1.4 倍，并且将容错能力从原来的容忍丢失 2 个备份提升到了容忍丢失 4 个备份，如图 3.5 所示。而且由于冷数据的存储除了校验块信息，同样保存了源文件的一份副本，在绝大多数时候（即没有发生节点故障时），访问冷数据也不需要对文件进行额外的解码操作。

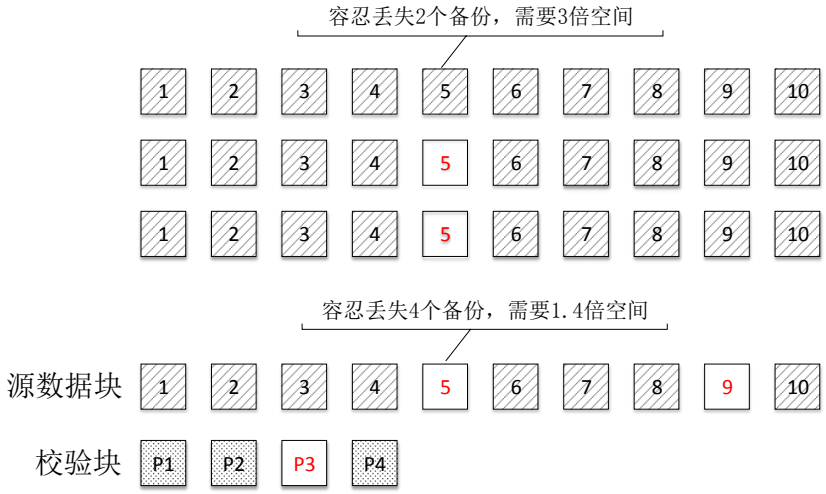

图 3.5 冷数据优化存储效率分析

下面将详细分析冷数据存储模块中的数据编码过程：

1. 获取需要编码文件的信息。首先 RaidNode 从冷热数据判定模块获得需要编码的冷数据文件信息，其中包括文件 ID、文件名称、Block 数目、Block 所在 DataServer 的位置。

2. 下载冷数据的条带单元。获得了冷数据文件信息后，RaidNode 节点首先判断该文件是否为小文件（Block 数目小于 10），如果为小文件则放弃编码。否则下载该文件的第一个条带（默认为 10 个 Block）的大小进行编码。编码完成后循环下载第二个条带进行编码。

3. 对条带进行编码。使用纠删码对文件的一个条带进行编码，生成 4 个校验块，如果条带小于 10 个 Block，修改编码参数，同样生成 4 个校验块。

4. 上传校验文件。文件编码结束后，需要将对应的校验块写入系统，这里将新生成的校验块放置到与条带中的 Block 不同的 DataServer 上以保证容错。

5. 修改元数据信息。减少源文件的副本数，并且将文件标记为编码文件，然后在条带中所有的 Block 信息中加入对应的校验块信息。

3.3.2 热数据存储

热数据在系统中采用了三副本备份的方式进行存储，并且新上传的数据被判定为热数据。当新文件被写入系统时，只有在其三个副本都写入成功时系统才认为该次写操作成功。这里采用了流水线的写入方式来保证数据的一致性，在文件读写流程会详细介绍。

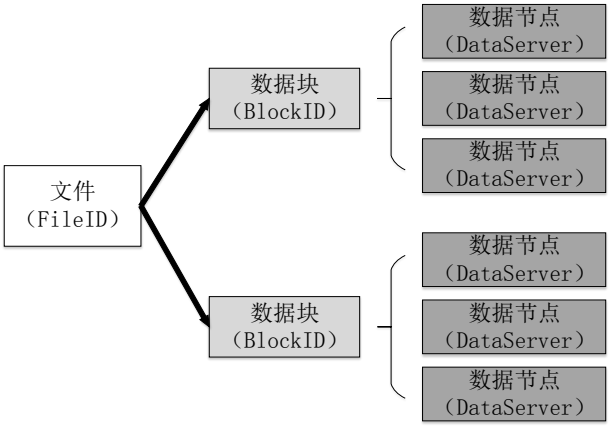

图 3.6 热数据的元数据结构

热数据的元数据结构如图 3.6 所示，每个文件被分为多个 Block 存储，每个 Block 对应 n（默认为 3）个 DataServer，表示该 Block 的副本所在位置。其中，主要采用 B+树来管理以上的元数据信息，节点的插入删除等操作采用 B+树的 COW（Copy On Write）操作流程，B+树中主要是保存了文件名（Filename）到文件 ID（FileID）的映射和文件 ID（FileID）到数据块 ID（BlockID）的映射关系，使得 Client 能够识别系统的整体目录树结构，并且能够对目录、文件和数据块进行查询和相关操作。系统运行时常驻内存，B+树每隔一段时间会 Checkpoint 固化到称为 FSImage 的文件中，以提供系统的异常恢复。

数据块到 DataServer 的映射关系保存在 BlocksMap 结构中，在 Client 读写数据块时按照负载均衡策略分配 DataServer 列表，并且定时检查数据块副本数与需求是否一致，在不一致时对相应 DataServer 发送副本的复制和删除操作命令。这部分元数据在系统运行时通过 DataServer 上报的数据块信息动态建立，常驻内存，不会进行持久化操作。

对热数据的三副本备份的容错主要通过以下三个列表来实现。待检查副本数目的 Blocks 集合 BlockReplicationCandidates，实际副本数小于需求副本数的 Blocks 集合 UnderReplicatedBlocks，和正在进行复制操作的 Blocks 集合 PendReplications。通过这三个列表对热数据的副本数进行管理和检查，保证实际副本数能够达到要求。因为保存在 DataServer 上的 Block 可能会因为磁盘损坏或者校验数据出错而导致数据不可用。在这种情况下，这个 Block 对应的副本数会下降，于是将此 Block 的信息加入 UnderReplicatedBlocks 列表。NameServer 在一定时期内会对该列表中的 Blocks 进行副本拷贝的操作，在拷贝期间，将这些 Blocks 的信息加入 PendReplications 列表。

在 Block 做副本拷贝时，正常情况下能够拷贝成功，但是有些拷贝操作因为网络或者机器原因会失败，当失败发生时，重新将对应 Block 信息加入 UnderReplicatedBlocks 列表等待下次复制。通过这种机制能够保证热数据的副本数始终达到期望的数目。

3.3.3 本地文件管理

RaccoonFS 将大文件分割成 64M（可配置）的 Block 进行存储。一般来说一个磁盘启动一个 DataServer。Block 存放在本地 Linux 系统的目录中，DataServer 存放数据的目录是 storage（目录名字可在配置文件中更改），storage 下有两个目录，一个是 data 目录，用来存放数据块；一个是 tmp 目录，用来存放正在复制的数据块，复制完成后，数据块会移到真正存放数据的 data 目录中。一个 Block 存储的时候，对应一个文件名为 blk_*blockid* 的文件，以及一个名为 blk_*blockid_blockversion*.meta 的元数据文件，这个元数据文件用于存放 Block 的检验信息。

图 3.7 中是 storage 文件夹的结构，data 文件夹中有固定数目的子目录。每个子目录下都是用来存放数据块和对应的元数据信息的。每个子目录能存放的最大 Block 的数目也是固定的（默认为 512 个），如果第一层所有目录的数据已写满，则从 subdir0 目录开始再生成固定大小（默认为 512）的子文件夹。tmp 文件夹中的文件存放方式，是完全扁平化的，tmp 下没有子目录，直接存放正在复制的数据块。存放数据块的数据结构包含以下内容：

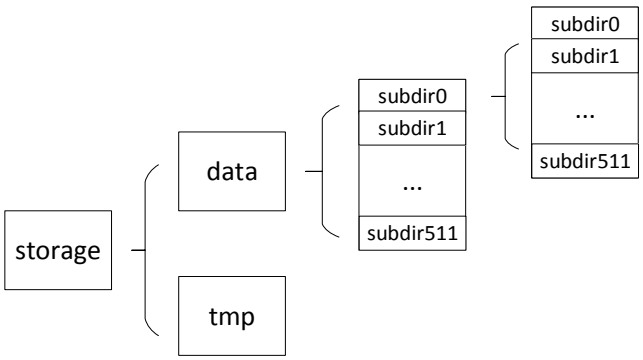

图 3.7 本地文件管理结构

1. Block

Block 以文件的形式存储在 DataServer 上，是 DataServer 上的最小的数据结构，一个 Block 最大为 64M（可配置，默认为 64M）。每个 Block 有一个对应的元数据文件，存放的是校验信息。Block 的属性有三个：1）Block ID，每个 Block 的 Block ID

都是唯一的；2）Block 的字节数，即一个 Block 的长度；3）Block 的版本号。

2. FSDir

FSDir 用来存放 Block 的目录，有固定数目的子目录，每个子目录能存放固定数目的数据块文件和元数据文件。

3. FSDataSet

FSDataSet 是 DataServer 上的最大的数据结构。管理整个存储结构，对外提供创建、读取、删除 Block 的操作，以及获得本地数据信息的操作。

3.3.4 数据恢复

数据恢复分为在客户端对数据的恢复和在系统中对数据的恢复。客户端对冷数据进行读操作时，如果遇到源数据块丢失的情况，系统中采用了 Swap 的方法将该丢失的源数据块使用相应的校验块进行替换，然后在客户端进行相应的解码操作恢复出原文件，如图 3.8 所示。但此时系统中丢失的 Block 仍然没有恢复，客户端并不负责对系统中丢失 Block 的恢复。因为此时客户端的目的是以最快的响应时间获取文件，所以系统中丢失的 Block 将会由另外的线程进行恢复。客户端对热数据进行读操作时，遇到源数据块丢失的情况，因为系统中存有三个副本，所以会尝试从相应的 DataServer 列表的下一个 DS 中读取数据，当三个 DS 都读取失败时，客户端返回错误。

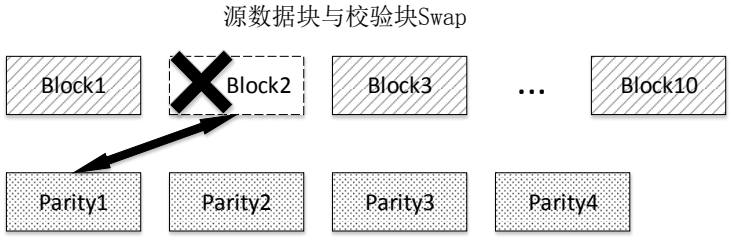

图 3.8 客户端读文件时的 Swap 操作

为了及时检测出丢失的 Block，并对丢失数据进行恢复，系统会定期对数据的 Block 状态进行收集，目前该收集工作主要在 DataServer 进行数据块上报操作时进行。DataServer 定期（默认为每天凌晨）向 NameServer 汇报自身保存的所有数据块信息，NameServer 将上报的数据块信息与 BlocksMap 进行对比，分别记录哪些 Block 需要添加，删除和恢复，DataServer 上报的所有数据块放置在待检查列表中等待副本数检查，情况如下：

1. BlocksMap 中找不到上报的 Block，说明在元数据的结构中已经将该 Block

删除了，因此将该 Block 加入 toremove 列表，DataServer 在收到回复后会将该 Block 从本地删除；

2. BlocksMap 中包含该 Block，但 DataServer 上报的列表中没有该 Block，说明该 Block 在 DataServer 上不可用，当该 Block 所属文件为热文件时，将该 Block 加入相应列表，系统会使用复制的方法恢复该 Block，当该 Block 所属文件为冷数据时，将该 Block 加入 todecode 列表，RaidNode 会负责对该 Block 进行恢复操作。首先 RaidNode 需要下载该 Block 所在的整个条带，然后下载相应的校验块进行恢复操作；

有时候整个 DataServer 会因为发生网络故障或硬件出现问题而导致整个节点故障，这种情况不能等到数据块上报的时候再处理，而必须及时发现。因此，每个 DataServer 都需要定时向 NameServer 发送心跳来汇报信息，发送的心跳内容有：Dataserver 的 ID，总容量，剩余的可用空间，网络链接数目等。

3.3.5 文件读写流程

本节将对 RaccoonFS 系统中的文件读写过程做详细的介绍，系统中文件写操作不需要区分冷热数据，因为系统判定刚写入的数据为热数据。系统中文件读操作在没有发生节点失效时也没有区分冷热数据，因为冷热数据都至少保存了一份副本数据在系统中，但是当发生节点失效时，系统对冷热数据的处理不同，冷数据需要经过 Swap 操作下载检验块并对文件进行解码，而热数据则需要尝试其他存储该副本的 DataServer 是否同样发生了节点失效。

下面首先介绍写流程，客户端对文件系统写入数据的流程如下：

1. 客户端首先向 NameServer 发起写文件的请求，NameServer 检查文件路径长度、安全模式、租约，然后在名字空间的目录树中增加一个文件。

2. 当客户端开始写入文件的时候，采用三副本备份策略，将文件按照 Block 的粒度写入，客户端从 NameServer 获取用来存储 replicas（副本）的合适的 DataServers 列表，并创建租约，同时 NameServer 增加相应的元数据信息。

3. 客户端申请建立流水线，如图 3.9 所示。如果其中某个 DataServer 出现异常，则放弃该 Block，向 NameServer 返回错误信息，并重新申请 Block，返回步骤 2。

4. 开始以流水线的形式将数据块（包含校验信息）写入所有的 replicas（副本）中。写数据的时候只在流水线中最后一个 DataServer 上做数据校验，校验的单位长度为 16k。

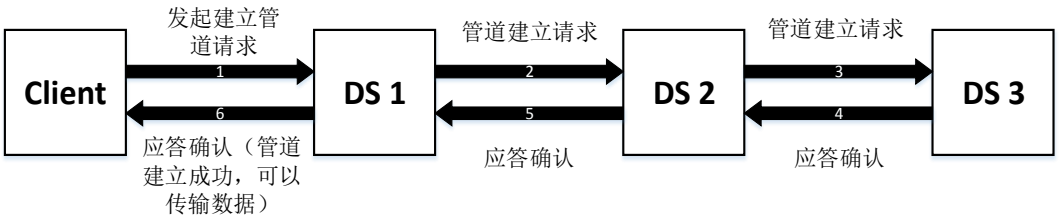

图 3.9 流水线的建立

5. 当 Block 传输完毕，客户端会移除租约，重新返回步骤 2 传递新的 Block，直到整个文件都写完。整个操作的流程图如图 3.10 所示。

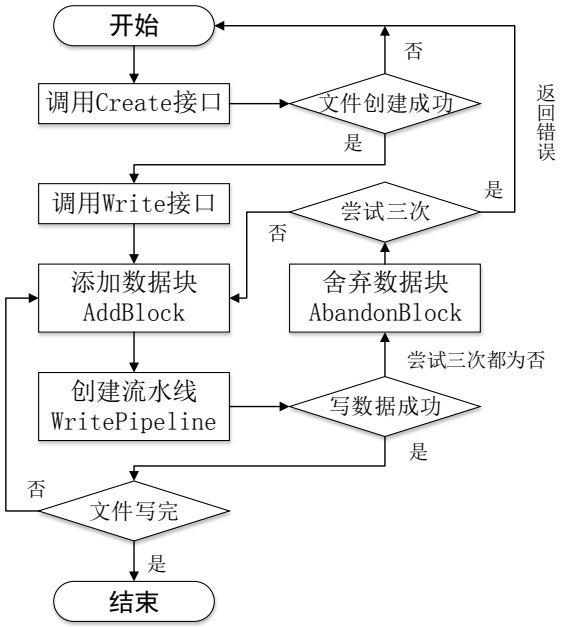

图 3.10 写文件流程图

下面介绍读流程，客户端从系统中读取文件的流程如下：

1. 首先查询本地缓存的元数据，如果本地缓存中有要读取文件的元数据信息，则直接根据元数据信息到 DataServer 上读取文件，跳到步骤 3，否则跳到步骤 2。

2. 客户端向 NameServer 发起读取文件请求。NameServer 根据请求找到指定文件的文件 ID 和 Block 信息并返回，NameServer 会将 DataServer 列表信息根据副本读取策略进行排序，如果数据为冷数据则只有一个副本；

3. 客户端选取列表中的第一个 DataServer 读取 Block 数据，如果读取操作成功，则继续读取下一个 Block。若读取热数据时出现读出错，则从列表中下一个 DataServer 上读数据；若所有 DataServer 都无法读取，则向客户端返回错误；若读取冷数据时发生错误，则经过 Swap 操作跳过当前 Block，然后下载对应的校验块做替换，最后对文件进行解码；

4. 当读完所有 Block 后，文件的读取还没有结束，客户端会继续向 NameServer 获取下一批的 Block 列表，否则读流程结束。整个操作的详细流程如图 3.11 所示。

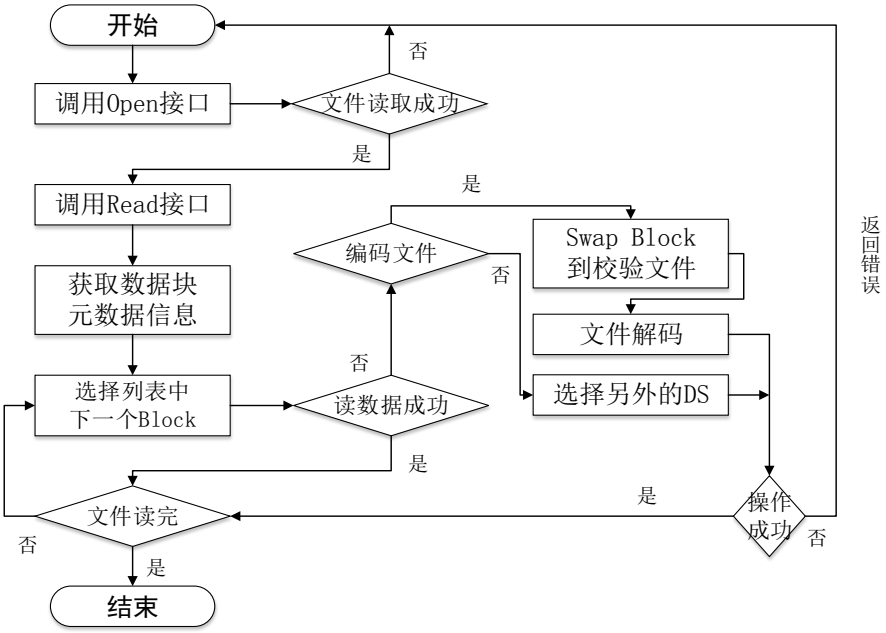

图 3.11 读文件流程图

3.4 纠删码的选择和参数配置

本文第二章中介绍了一些在存储系统中使用的典型纠删码。本节中将对这些纠删码进行比较，从存储效率、容错能力和运算性能上进行详细的分析，对系统中纠删码的使用进行选择并对参数进行配置。

阵列纠删码在 RAID 技术中应用很多，因为其二维码字的结构很符合磁盘阵列的构造，所以大多数阵列纠删码的研究和实现也是针对磁盘阵列所提出的。阵列纠删码的编解码运算为简单的异或运算，所以软硬件的实现比较简单并且运算效率高。但阵列纠删码的容错能力不强，大部分阵列纠删码的容错能力都不超过 3。并且阵列纠删码的几何形式往往要满足一些特殊的性质，比如 GRID 码的条带长度和条带宽度的限制。而分布式存储系统往往要求高可扩展性，条带参数限制不利于系统节点数量扩展。所以阵列纠删码多应用在磁盘阵列而不是分布式的存储系统中。

以旋风码为首的低密度纠删码在数据恢复上具有优势，并且编译码也是采用的简单的异或操作，所以复杂度较低，目前低密度纠删码主要用于高速网络传输。但是由于低密度纠删码不能保证译码准确性，是一种非确定的码，所以在存储系统中的

应用并不广泛，因为它可能丢失部分源数据，而且不可恢复。所以低密度纠删码主要用于网络传输，并不能保障分布式文件系统中的高可靠性。

RS 类纠删码的优势为它是唯一的一种可以满足任意的数据磁盘数 k 和冗余磁盘数 m 的 MDS 的编码方法，可以满足不同码率的要求，达到任何想要配置的容错能力。所以从分布式文件系统的存储效率和容错能力上来看，RS 类纠删码是最符合要求的。但是 RS 类纠删码的运算效率并不理想，需要进行有限域上的多项式运算，虽然有一些方法能够避免有限域上运算，但是在解码过程中，RS 类纠删码仍然需要进行矩阵求逆运算，当数据量很大时，时间的延迟仍然非常严重。因此很多研究者针对降低 RS 类纠删码的编解码时间复杂度进行了努力，可是其性能还是远远不如阵列纠删码，因此，RS 类纠删码往往应用于对实时性要求并不高的只读分布式存储系统中。

表 3.3 对几种典型纠删码的优劣势比较做出了总结。

表 3.3 纠删码的比较

	EVENODD 码	**范德蒙 RS 码**	**柯西 RS 码**	**GRID 码**	**旋风码**
存储效率	83%	可配置	可配置	80%	可配置
容错能力	2	可配置	可配置	≤15(固定)	可配（风险）
实现复杂度	简单	较复杂	较复杂	简单	简单
性能	高	低	偏低	高	高

由分析可看出，由于可自由配置容错能力，能够提供系统高可靠性，最适合应用于分布式文件系统的纠删码为 RS 类纠删码，但是 RS 类纠删码的主要劣势在于其运行性能。所以下面对几种 RS 类纠删码的性能在各种参数下做了测试，并加入了阵列纠删码 EVENODD 进行性能对比。首先对测试中涉及到的参数进行说明：k 为编码前数据分块数目；m 为编码生成的校验分块数目；w 为 word size 大小（bits），一个条块（strip）单元由一个 w-bit 的 word 组成；在柯西 RS 码中，因为对生成矩阵中 Galois 域元素（即一个 word）用 w*w 的 0-1 矩阵进行了替换，导致一个条块被分为 w 个 packets，所以有另外的参数 packet size，单位为 Bytes。

1. 采用不同大小的数据文件时的编解码速度（使用相同的 k=5，m=2，w=16）：

图 3.12 为使用不同的纠删码编解码 1MB~2GB 大小的数据，其中解码是在丢失一个源数据块的情况下得出的，范德蒙 RS 码由于只丢失一个源数据块和丢失多个数

据块的解码方法不同，所以丢失一个数据块时的解码比丢失多个块时的解码速度快很多，在丢失多个块时，范德蒙 RS 码的解码速度和编码速度一致。从结果可以看出，纠删码的编解码速度与数据大小成正比，在编码速度上，改进的柯西 RS 码是 RS 类码中速度最快的。并且各种纠删码在文件大小达到 100MB 以后编解码速度基本达到稳定。

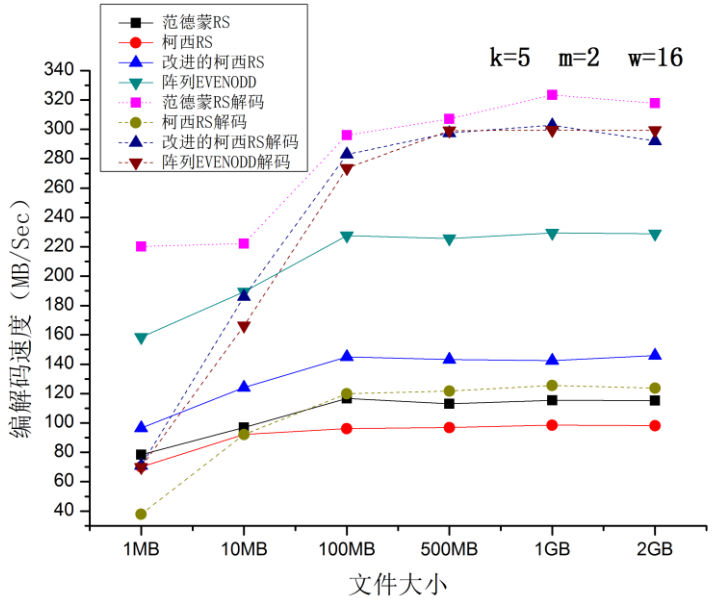

图 3.12 编码不同大小的数据时的编解码速度

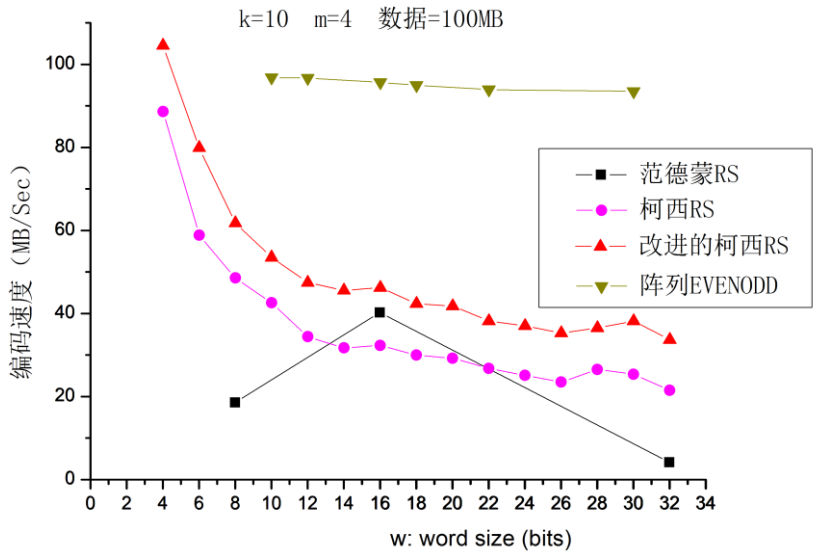

图 3.13 使用不同 w 值时的编码速度

2. 采用不同的 w 值（word size）时的编码速度（使用相同的 k=10，m=4，数据=100MB）：

图 3.13 为几种典型的纠删码分别使用 word size 为 4~32 时编码 100MB 的数据时的速度，其中范德蒙 RS 码只能取 w 值为 8，16 和 32，而 EVENODD 码只能取 w+1 为素数时的值。由结果可以看到，范德蒙 RS 码在 word size 为 16 时性能最佳，而柯西 RS 和改进的柯西 RS 码在 word size 为 4 时性能最佳，EVENODD 码对 word size 不敏感。

3. 采用不同数目的数据块时的编码速度（使用相同的 m=4，w=8，数据=100MB）

图 3.14 为几种纠删码在数据块的数目分别为 4~10，校验块数为 4（容 4 错）的情况下编解码 100MB 的数据所得到的编码速度，由于 EVENODD 码对 word size 有所限制，所以此组测试中 EVENODD 码选择的 w 值为 10。由结果可以看到，数据块数目越多时，虽然存储成本越低，但编码速度也越慢。但是当编码的数据块数目越多时，系统的存储效率也越高。

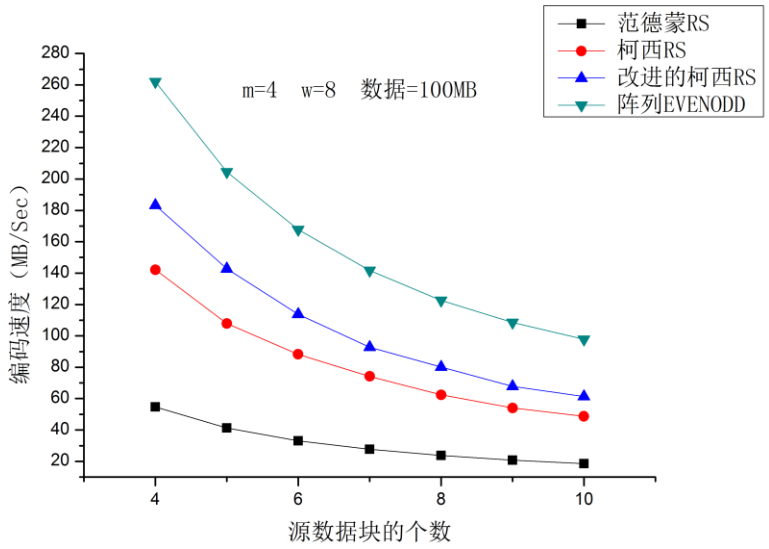

图 3.14 采用不同数目的源数据块的编码速度

4. 采用不同的 packet size 时的编码速度（使用相同的 k=10，m=4，w=4，数据=100MB）

图 3.15 为柯西 RS 码在不同的 packet size 下编码 100MB 的数据时的编码速度，因为范德蒙 RS 码没有这一参数，所以没有测试范德蒙 RS 码的数据。其中柯西 RS 码对 packet size 的要求为能被 8 整除。由结果可以看到，packet size 较小时，其编码

速度也非常的慢，但是当 packet size 增加至 8192 及以上时，packet size 的增加对编码速度的影响已经不大了。

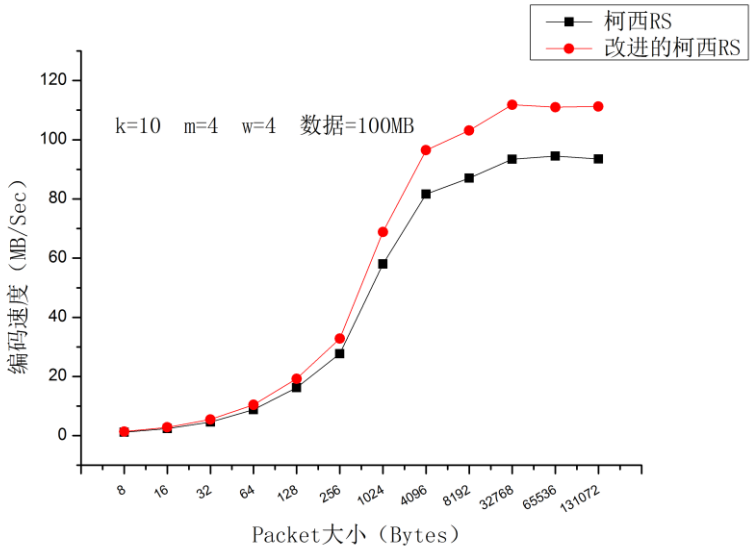

图 3.15 柯西 RS 码在不同 packet size 下的编码速度

由这节的分析和测试可以看到，因为分布式文件系统要求高扩展性和高可靠性，所以虽然阵列纠删码 EVENODD 具有简单高效的特点，但是因为其容错能力不强，并且不利于分布式文件系统的节点扩张，所以本系统中没有选用 EVENODD。而测试表明，改进的柯西 RS 纠删码由于能够保证在任意的容错度下算法仍具有 MDS 的特性，并且具有相对较高的编解码性能，所以在 RaccoonFS 系统中选用了改进的柯西纠删码作为编码的技术。

3.5 本章小结

本章详细论述了 RaccoonFS 系统中存储效率优化方案的设计与实现，介绍了方案的架构设计、核心模块设计与实现，包括如何判定冷热数据，如何对冷热数据进行分别存储，如何对数据进行恢复，以及本地文件的管理和文件读写流程的分析。最后，对纠删码的选择和参数配置做了详细的分析和测试。

4 测试与结果分析

为了评估 RaccoonFS 系统中存储效率优化的方案是否在存储成本、数据可靠性和系统性能之间达到平衡，本节将主要针对存储效率优化方案进行以下的功能测试和性能测试：柯西 RS 类纠删码在单机时的编解码性能；利用纠删码将系统中的热数据转换为冷数据存储时所需要的时间和转换后的数据访问时间；冷数据在系统中所占比例不同时系统的存储效率；冷数据在系统中所占比例不同时客户端的响应时间；多客户端分别并发访问冷数据/热数据时平均响应时间的对比；客户端请求中热数据请求量为 80%时系统平均响应时间的对比。通过几组测试说明了基于冷热数据区分和副本纠删码混合的存储方案确实起到了优化分布式文件系统存储效率的作用，提高了数据可靠性，并且此方案尽可能的降低了采用纠删码技术对系统性能造成的损耗。

4.1 测试环境

测试所采用的服务器配置均相同，其配置如表 4.1 所示。单独使用 1 台服务器作为中心节点，使用 3 台服务器作为数据节点，3 台服务器作为客户端，各个服务器之间用千兆局域网连接。

表 4.1 测试配置

设备	配置
CPU	Intel(R) Xeon(R) CPU E5620 @ 2.40GHz
内存	DDR3 12GB
硬盘	SATA 300G @ 7200R/M
操作系统	Red Hat Enterprise Linux Server release 5.4

4.2 性能测试

1. 纠删码对系统性能的影响。

本文提出的分布式文件系统存储效率优化方案中，使用了改进的柯西 RS 类纠删码对冷数据进行编码转换，经过上一章节的讨论，在参数上使用了 k=10, m=4, w=4, packet size = 8192 Bytes 以求在编解码性能上达到最佳，并对系统的性能影响降低到

最小，首先测试了在单机上选用上述参数后改进的柯西 RS 码的编解码速度。

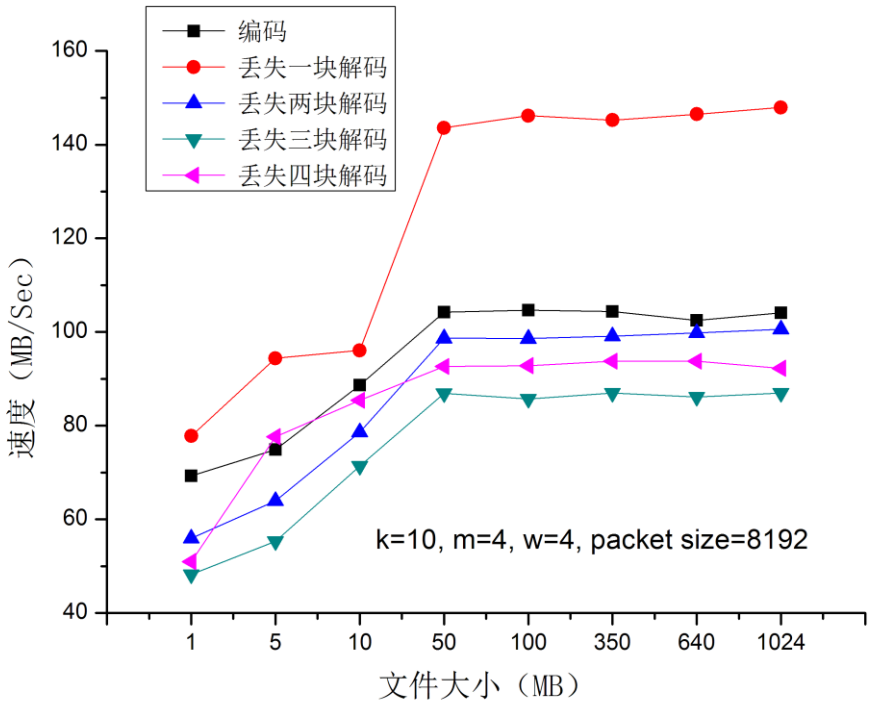

图 4.1 柯西 RS 码在单机的编解码性能

由图 4.1 可知改进的柯西 RS 类纠删码在编解码不同大小的文件时的速度，并且由于系统可以容忍最多丢失 4 个数据块，所以对改进的柯西 RS 码在丢失不同数目的数据块时的解码表现也做了对比。可以看出，当文件大于 50MB 时，编解码速度趋于稳定，所以对于选择 10 个 Block 作为一个条带的 RaccoonFS 系统来说，正常的文件编码时间为 6 秒钟左右，而在丢失一个源数据块的情况下，正常的解码时间为 4.5 秒左右。

经过上面的测试可以知道，改进的柯西 RS 类纠删码在单机测试中的编解码时间为 4~6 秒，下面将对此纠删码运用在系统后的性能进行测试。按照冷热数据判定的规则，当一个文件连续一个星期的访问量都低于阈值（默认为 5）时，系统会将此文件判定为冷数据。经过判定后，系统会将冷数据使用纠删码编码进行存储，首先，RaidNode 需要将一个完整的条带下载到本地为编码做准备，然后利用改进的柯西 RS 码对该条带进行编码，生成校验文件，最后，将校验文件写回系统，并减少原文件的副本数目。需要对整个冷数据转换的过程进行测试，记录了整个过程的时间消耗，并且测试几种节点故障下用户对文件进行访问的响应时间，结果如图 4.2 所示。

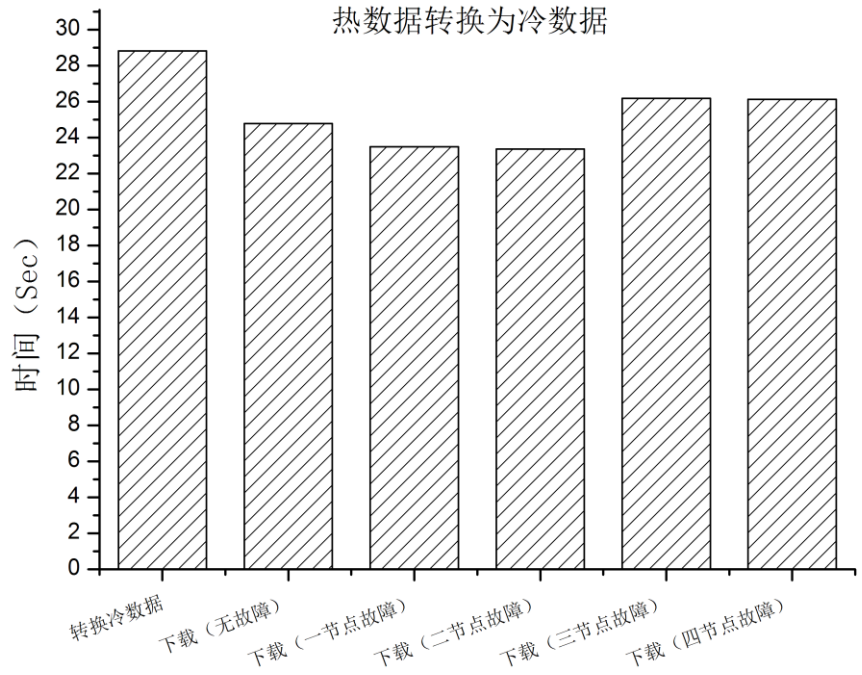

图 4.2 热数据转换为冷数据后在系统中的性能

由图 4.2 可知当把纠删码应用到系统中后，编解码时间占上传下载文件总时间的百分比并不大。比如一个用户对一个 640MB 的条带进行下载操作时，总共响应时间需要 26 秒左右，使用 IB 网络测试时响应时间需要 17 秒左右，而其中解码时间在实测中均为 4~5 秒，所以系统在响应用户请求时的大部分时间仍然用在网络传输中对文件分块的获取上。并且分析测试结果可知，在发生节点故障的情况下，因为采用了 Swap 的方式直接使用校验块对丢失的源数据块进行替代，暂时不在系统中对文件进行修复，所以用户的响应时间与没有发生节点故障时相似。因此方案中采取不在客户端对系统中丢失的数据块做恢复处理，确实能够避免延迟用户的响应。另外从测试结果中得出，系统在发生节点故障的时候，响应时间与故障的节点数目无关，这是因为改进的柯西 RS 码在发生节点故障时解码速度几乎相同。

2. 存储效率的比较。

下面将对系统在选用三副本策略和冷热数据混合存储方案时的存储效率做出比较。其中存储效率的计算方法在 1.2.1 节中提到过。采用三副本策略时，每个文件存储了对应的三个副本，采用冷热数据混合存储方案时，每个条带（10 个 Block）存储了对应的一个副本和校验块（4 个 Block）。图 4.3 为冷数据占总数据的不同比例

时的存储效率与三副本的存储效率的比较。

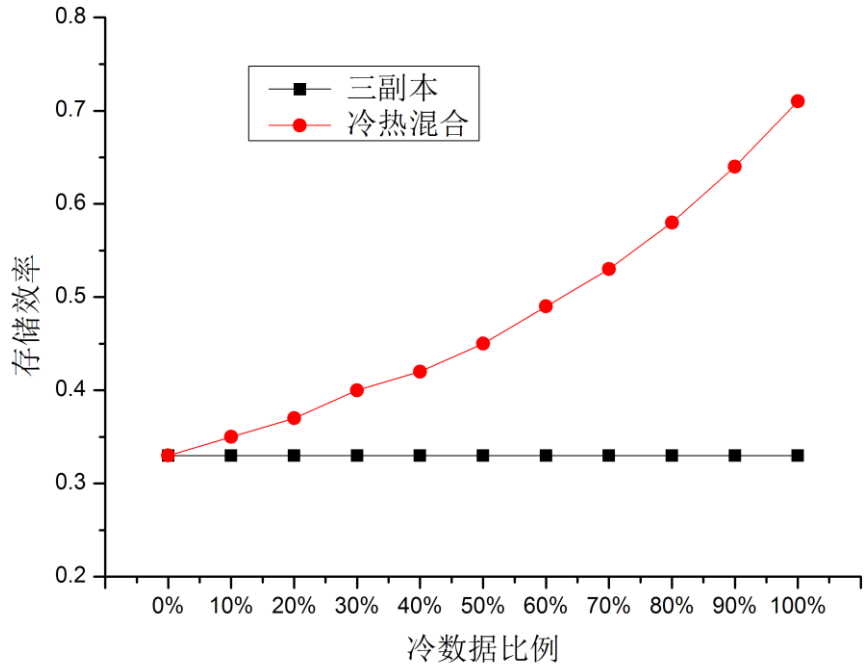

图 4.3 存储效率的比较

由图 4.3 可知，采用三副本策略时，系统的存储效率只有 33%，意味着存储 1PB 的有效数据需要 3PB 的实际存储空间。采用冷热数据混合存储方案时，当冷数据在系统中的比例越大时，系统的存储效率越高，最高可达到 71%，意味着存储 1PB 的有效数据只需要 1.4PB 的实际存储空间，相比于三副本策略，节省了 1.6PB 的存储成本。而由 3.3.1 节的分析可知，实际运行中冷数据的比例约为 80%，所以实际存储效率约为 58%，每 1PB 的数据节省了 1.28PB 的存储空间。由结果分析可知，采用冷热数据混合存储的方案大大提高了分布式文件系统的存储效率。下面将对不同负载下客户端的响应时间进行测试，检查此方案是否对系统性能造成了影响。

3. 冷数据占总数据不同比例时，客户端访问的响应时间。

首先测试了当冷数据占总数据的比例不同时，客户端访问的响应时间。这里使用了 3 台不同的服务器作为客户端，每个客户端启动 10 个线程并发访问，每个线程分别访问不同的文件，每个文件为 1 个条带大小（640MB）。同时，使用了 3 台不同的服务器作为 DataServer 存储数据。这样对于每个冷数据/热数据文件，总共有 3 个客户端对其进行并发访问。

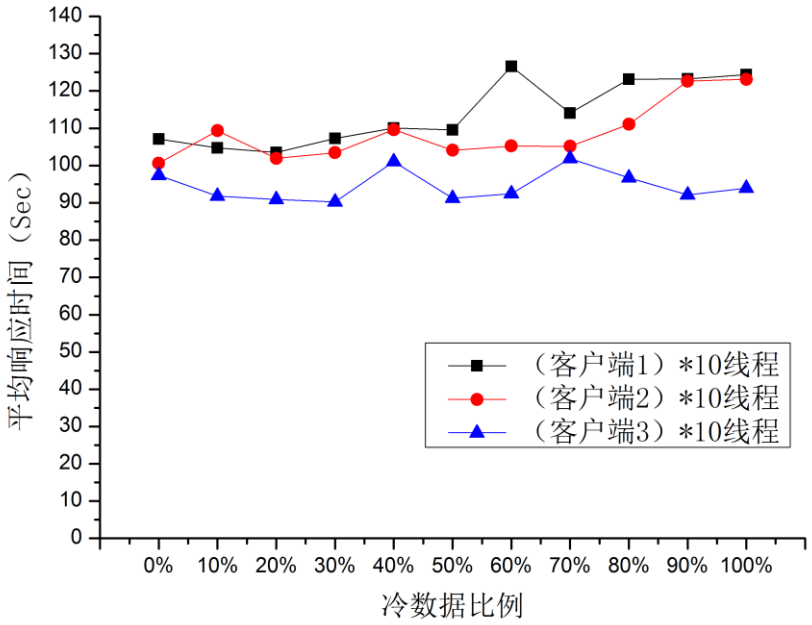

图 4.4 冷数据占总数据不同比例时的响应时间

由图 4.4 可知，不同的客户端对文件完成访问的响应时间都在 100 秒左右，这里冷数据所占比例对客户端的访问时间并没有影响。在前面的测试中，得出一个条带的数据（640MB）在单客户端访问时，平均响应时间约为 26 秒，这里不同客户端对文件并发进行访问时，响应时间延长至 100 秒左右。这是因为每个客户端启动了 10 个不同的线程，虽然每个线程访问不同的文件，但是这些线程需要通过同一个网卡进行数据传输，所以并发情况下对响应时间造成了影响。另一方面，每个文件都有 3 个客户端对其进行并发访问，这也造成了响应时间的一些延迟。因此，客户端的平均响应时间在此测试环境中比单客户端要长。

另外，由于此测试中受到测试环境的一定影响，每个文件实际的客户端并发访问量为 3，所以冷数据比例对于响应时间的影响并没有体现出来，这点将在下面专门针对大量客户端并发访问同一个文件的测试案例中进行测试。而在实际环境中，每个冷数据的实际客户端访问量已经很低，所以并不存在大量客户端并发访问冷数据的情况。

4. 大量客户端并发访问冷数据/热数据时，客户端访问的响应时间。

在上个测试案例中，大量客户端并发访问文件的响应时间由于环境的限制没有进行测试。在本案例中，将文件数目缩减到一个文件（大小为 640MB），将所有客户端

的访问都指向此文件，以此来测试在大量客户端并发访问时的平均响应时间。实验中采用了 3 台机器做客户端，每个客户端分别启动 n 个线程对同一个文件进行访问，这样文件的并发客户端访问数目为 3*n。分别对热数据文件，无故障的冷数据文件，和分别丢失了一个块和两个块的冷数据文件做了测试。

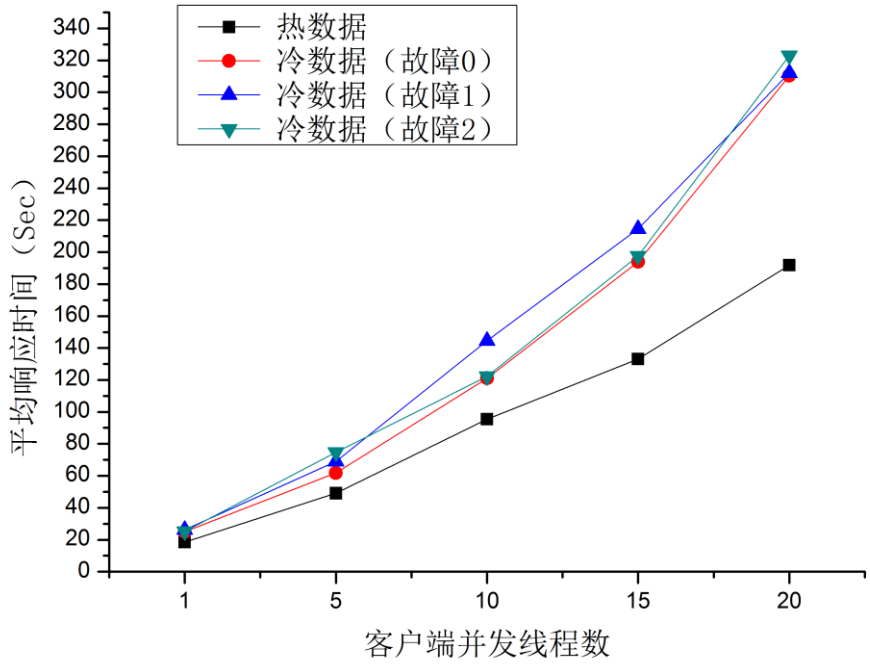

图 4.5 大量客户端并发访问的响应时间

由图 4.5 可知，在面对大量客户端并发访问的情况下，冷数据的响应时间比热数据的响应时间要长。测试结果表明，当客户端数目达到 3*20 时，访问一个 640MB 的冷数据的平均响应时间为 320 秒左右，而此时访问同样的热数据的平均响应时间为 190 秒左右。这是因为冷数据在系统中只存有一个副本，所以不能让多个客户端均衡的到多台服务器访问不同的副本，对于热数据来说，并发的客户端可以到不同的服务器访问同一个文件的不同副本，因此在并发情况下响应时间也会相对更快。由此可见，如果在系统中将访问次数很多的热数据也进行编码存储，会极大地影响系统性能。因此基于冷热数据区分的混合存储方案保留了三副本策略带来的系统高性能，弥补了编码后的文件在客户端并发访问时对系统性能的影响。

5. 并发客户端的请求按照 80%和 20%的比例分别请求冷数据/热数据时，客户端访问的响应时间。

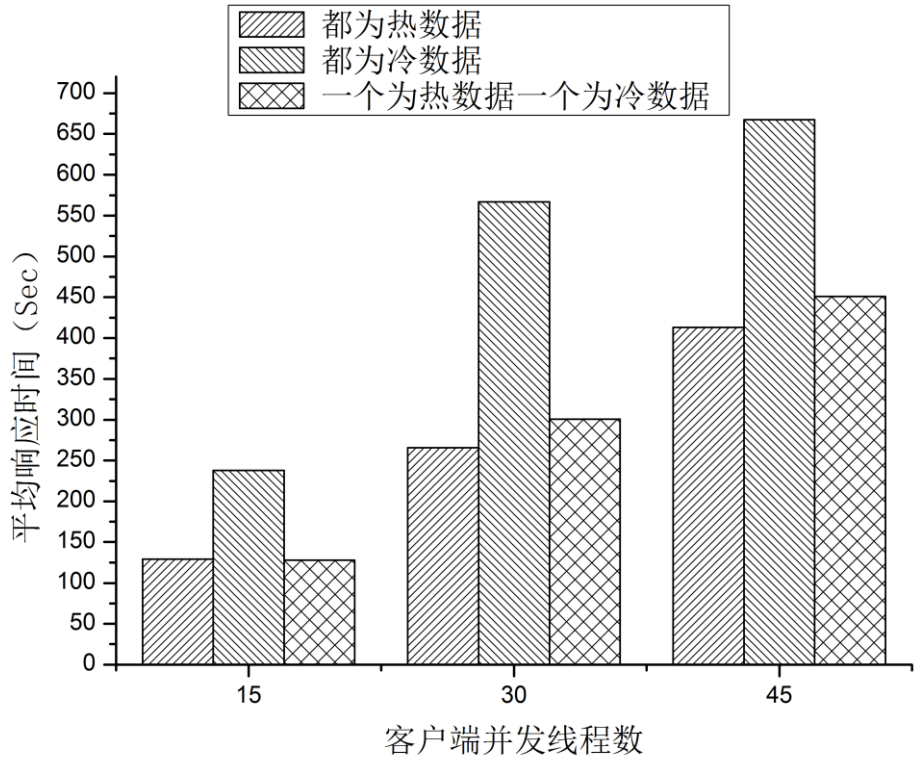

图 4.6 并发客户端的请求按照 80%和 20%比例时的响应时间

为了测试在实际情况中，本文提出的存储效率优化方案和三副本方案、数据全编码方案的用户响应时间对比，将客户端的请求按照 3.3.1 节中讨论的实际负载的情况进行了模拟。其中客户端的请求按照 8:2 的比例进行了区分，分别访问不同的文件。为了尽量减小服务器网卡限制对于测试的影响，并且结合上个测试案例中多个客户端进行并发访问对结果的影响，这里选用了两个 640MB 大小的文件，分别测试了当两个文件都为热数据，两个文件都为冷数据，两个文件分别为冷数据和热数据时的情况。

图 4.6 中，客户端并发线程数目为 15 表示在 3 个客户端上分别都开启了 15 个线程对文件进行访问，其中 12 个线程访问文件 1，另外 3 个线程访问文件 2，两个文件的访问请求比例为 8:2。由测试结果可以知道，当两个文件全部为热数据（即不对文件进行编码，每个文件都存储三个副本）时，其访问平均响应时间与两个文件分别为冷数据/热数据（即 80%请求访问热数据，20%请求访问冷数据）的平均响应时间基本持平。而当两个文件都为冷数据（即对所有文件进行编码，每个文件存储一个副本和校验码）时，其访问平均响应时间要远远高于上面两种方案。因此，基于

冷热数据区分的混合存储方案在实际负载中的整体性能表现与采取三副本策略时的系统性能相似，而远远要优于对所有文件都进行编码存储的方案。

4.3 本章小结

本章主要对提出的分布式文件系统存储效率优化方案进行了相应的性能测试。从测试结果可以看到此方案相对于三副本备份方法在存储效率上有极大的提升，并且此方案避免了纠删码方法带来的系统性能损耗，达到了和三副本备份方法相似的系统性能。测试说明了基于冷热数据区分和副本纠删码混合的存储方案确实集合了多种方案的优势，并在存储成本、数据可靠性和系统性能之间达到了平衡。

5 全文总结

随着信息的爆炸式增长，企业中分布式文件系统集群规模急剧的扩张，存储成本也在不断上升。由于分布式文件系统中传统的三副本备份方法给系统带来了相当大的空间消耗，存储效率低下。本文在充分研究了现有分布式文件系统中存储效率优化的方法后，提出了一种基于冷热数据区分和副本纠删码技术的混合存储方案，在提高分布式文件系统存储效率的同时，将系统性能的损耗降低到了最小。本文所做的主要工作具体如下：

1. 分析了现有常用分布式文件系统的存储效率和国内外研究概况，简要介绍了对分布式文件系统的存储效率进行优化的必要性和相关背景。

2. 分析了存储效率优化方案所面临的三个挑战：如何在节点不可靠的前提下保障数据可靠性；如何在数据持续增长的情况下降低存储成本；以及如何保障系统性能。并且对现有的解决方案进行了分析与研究：三副本备份方法保证了系统高可靠和高性能，纠删码方法保证了低成本和高可靠，每种方法都各有优劣。提出了基于冷热数据区分的混合存储方案，将新数据以三副本备份方法进行存储，采用了两层冷数据判定队列用于区分冷数据，当文件被判定为冷数据后，采用纠删码进行编码存储，提高系统存储效率和容错能力。

3. 设计与实现了 RaccoonFS 系统中存储效率的优化方案，对方案进行了架构设计，实现了冷热数据判定模块，冷数据存储模块，热数据存储模块，数据恢复模块，本地文件管理模块和文件读写流程。并且对各种纠删码的性能和参数配置做了详细的分析和测试，最终选定了改进的柯西 RS 纠删码为系统中的冷数据编码。

4. 详细测试了基于冷热数据区分的存储效率优化方案。主要测试了柯西 RS 类纠删码的性能；此方案对分布式文件系统存储效率的提升；冷数据在系统中所占比例不同时客户端的响应时间；多客户端分别并发访问冷数据/热数据时的平均响应时间；访问请求按 8:2 比例分别访问热数据/冷数据时的系统平均响应时间。通过测试验证了基于冷热数据区分的存储方案优化了分布式文件系统的存储效率，提高了数据可靠性，并且在系统性能上没有造成严重的损耗。

本文提出的方案解决了存储效率优化的问题，但是系统仍然可以进行进一步的优化，具体有以下可以改进的地方：

1. 减少数据恢复时的网络带宽消耗。由于冷数据经过编码之后在系统中只存储

了一份副本，当发生数据丢失时，需要将整个条带下载到本地进行恢复，代价较大，再生码的研究正在逐渐成为业界研究的热点，经过再生码编码的文件不再需要整个条带来恢复全部数据，而只需要条带的子集就可以恢复文件。另外，对恢复过程的流程进行更加合理的设计也能够减少恢复时所需的网络开销。

2. 解决可能发生的冷数据再次变热问题。在实际情况中，冷数据有可能在某一时间点再次变为热数据。虽然这种转变只适应于部分数据集，但是针对可能发生的情况可以采取增加冷数据副本数目的方法提高系统性能。

致 谢

转眼两年半的硕士研究生生活就要结束了，算来在华中科技大学的学习也已经有将近七个年头，在毕业之际，这一路有许多人需要感谢。

首先要感谢的是我的导师冯丹教授。冯老师有着深厚的学术造诣，并且一直保持着一种严谨的治学态度。实验室在冯老师的指导下取得了累累硕果，科研能力和研究工作一直保持着国内存储领域的领先地位。并且，实验室的同学们在冯老师的严格要求下也都真正的在学术和实践能力中得到了提升。在此，要特别感谢冯老师的奉献和为实验室所做的贡献。

感谢我的指导老师施展老师。还记得大四时，未进实验室就对施老师的好名声有所耳闻，经过接触后，更加觉得施老师为人亲切，学识渊博。在论文写作过程中，施老师从开题时就一直给予我学术上的指导，有时候凌晨还在为我修改论文。施老师非常尊重学生的意见，一直启发我们要进行自主思考，而不是一味跟随。施老师总是站在我们的立场为我们着想，让每一届毕业的同学都切实感受到了来自老师的温暖，即将离开实验室，我多么希望自己曾经也能多站在施老师的立场，对实验室的建设有更多贡献。不管以后在哪里工作，我都会对施老师怀着敬佩和感激的心情。

感谢实验室的李洁琼老师、王芳老师、刘景宁老师、曾令仿老师、谭支鹏老师、华宇老师、童薇老师、田磊老师、熊双莲老师等，他们的辛勤付出为我们营造了一个良好的学术氛围和融洽的工作环境。

特别要感谢的是小分队的各位战友，无论是我们一起的吃饭讨论组，面试突击组还是最后的论文攻坚战，我们都一起奋斗，互相鼓励，人生中遇见知己不易，很高兴我遇见了你们。陈碧砚、陈云云、涂丹辉、叶为民、姚莹莹，感谢你们一直对我的包容和肯定，希望大家都能够敢于追求主导生活的权利，梦想成真。

感谢广域网存储组的各位学长学姐，周峰、赵恒、贺燕、刘小军、郑颖、王霁欣以及柳青、李勇、李宁、焦田峰、尹祎的指导，各位学弟学妹，付宁、王艳萍、赵千、桂莅、方波以及李白、黄力、韩江、郭鹏飞、张成文的帮助，因为大家的存在，我们才是一个温暖又热闹的集体。

最后要感谢我的父母，他们的付出我无以回报，感谢多年来在我求学路上的支持和肯定，祝愿我的家人幸福安康。

参考文献

[1] 梁李印. 阿里Hadoop集群架构及服务体系. 北京: Hadoop与大数据技术大会（HBTC)，2012: 1-29

[2] Thusoo A, Shao Z, Anthony S, 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

[3] Borthakur D, Gray J, Sarma J S, et al. Apache hadoop goes realtime at Facebook. in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. Athens, Greece: ACM, 2011. 1071-1080

[4] Ghemawat S, Gobioff H, Leung S. The Google file system. in: Proceedings of the nineteenth ACM symposium on Operating systems principles. Bolton Landing, NY, USA: ACM, 2003. 29-43

[5] Shvachko K, Kuang H, Radia S, et al. The Hadoop Distributed File System. in: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). IEEE Computer Society, 2010. 1-10

[6] Dimakis A G, Prabhakaran V, Ramchandran K. Decentralized erasure codes for distributed networked storage. IEEE/ACM Trans. Netw., 2006, 14(SI): 2809-2816

[7] Huang C, Simitci H, Xu Y, et al. Erasure coding in windows azure storage. in: Proceedings of the 2012 USENIX conference on Annual Technical Conference. Boston, MA: USENIX Association, 2012. 2

[8] Corbett J C, Dean J, Epstein M, et al. Spanner: Google's globally-distributed database. in: Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation. Hollywood, CA, USA: USENIX Association, 2012. 251-264

[9] Wilcox-O'Hearn Z, Warner B. Tahoe - The least-authority filesystem.in: Proceedings of the ACM Conference on Computer and Communications Security. Alexandria, VA, United states: Association for Computing Machinery, 2008. 21-26

[10] Aguilera M K, Janakiraman R, Xu L. Using erasure codes efficiently for storage in a distributed system. in: Proceedings. 2005 International Conference on Dependable Systems and Networks. Los Alamitos, CA, USA: IEEE Comput. Soc, 2005. 336-345

[11] Wikipedia. Storage Efficiency. 2013, http://en.wikipedia.org/wiki/Storage_efficiency

[12] Peter K, Reinefeld A. Consistency and fault tolerance for erasure-coded distributed storage systems.in: Proceedings of the fifth international workshop on Data-Intensive

Distributed Computing Date. Delft, The Netherlands: ACM, 2012. 23-32

[13] Li X, Lillibridge M, Uysal M. Reliability analysis of deduplicated and erasure-coded storage. SIGMETRICS Perform. Eval. Rev., 2011, 38(3): 4-9

[14] 李明强. 磁盘阵列的纠删码技术研究:[博士学位论文]。北京: 清华大学图书馆，2011

[15] Chen P M, Lee E K, Gibson G A, et al. RAID: high-performance, reliable secondary storage. ACM Comput. Surv., 1994, 26(2): 145-185

[16] Kubiatowicz J, Bindel D, Chen Y, et al. OceanStore: an architecture for global-scale persistent storage. in: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems. Cambridge, Massachusetts, USA: ACM, 2000. 190-201

[17] Jin C, Jiang H, Feng D, et al. P-Code: a new RAID-6 code with optimal properties. in: Proceedings of the 23rd international conference on Supercomputing. Yorktown Heights, NY, USA: ACM, 2009. 360-369

[18] Blaum M, Brady J, Bruck J, et al. EVENODD: an optimal scheme for tolerating double disk failures in RAID architectures. in: Architecture P O T S.ISCA '94. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994. 245-254

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

[20] Xu L, Bohossian V, Bruck J, et al. Low-density MDS codes and factors of complete graphs. IEEE Trans. Inf. Theor., 2006, 45(6): 1817-1826

[21] Katti R, Xiaoyu R. S-code: new distance-3 MDS array codes. in: IEEE International Symposium on Circuits and Systems (ISCAS). Piscataway, NJ, USA: IEEE, 2005. 348-351

[22] Hartiline J. R5X0: An efficient high distance parity-based code with optimal update complexity. IBM Research Division, 2004

[23] Hafner J L. HoVer erasure codes for disk arrays. in: Proceedings of the International Conference on Dependable Systems and Networks(DSN 2006). Philadelphia, PA, United states: Institute of Electrical and Electronics Engineers Computer Society, 2006. 217-226

[24] Hafner J L. WEAVER codes: highly fault tolerant erasure codes for storage systems. in: 4th USENIX Conference on File and Storage Technologies. San Francisco, CA, USA: USENIX Association, 2005. 211-224

[25] Cheng H, Lihao X. STAR: an efficient coding scheme for correcting triple storage node failures. IEEE Transactions on Computers, 2008, 57(7): 889-901

[26] Li M, Shu J, Zheng W. GRID codes: Strip-based erasure codes with high fault tolerance for storage systems. Trans. Storage, 2009, 4(4): 1-22

[27] 万武南. 分布式安全存储系统纠删码技术的研究：[博士学位论文]。 成都: 中国科学院研究生院图书馆，2006

[28] Schroeder B, Gibson G A. Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you? in: Proceedings of the 5th USENIX conference on File and Storage Technologies. San Jose, CA: USENIX Association, 2007. 1

[29] Corbett P, English B, Goel A, et al. Row-Diagonal Parity for Double Disk Failure Correction. in: Proceedings of the 3rd USENIX Conference on File and Storage Technologies. San Francisco, CA: USENIX Association, 2004. 1-14

[30] 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

[31] Plank J S. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Softw. Pract. Exper., 1997, 27(9): 995-1012

[32] Plank J S. Improving the performance of coordinated checkpointers on networks of workstations using RAID techniques. in: Proceedings 15th Symposium on Reliable Distributed Systems. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 1996. 76-85

[33] Lacan J, Fimes J. Systematic MDS erasure codes based on Vandermonde matrices. Communications Letters, IEEE, 2004, 8(9): 570-572

[34] Plank J S, Xu L. Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications. in: Proceedings - Fifth IEEE International Symposium on Network Computing and Applications, NCA 2006. Cambridge, MA, United states: Inst. of Elec. and Elec. Eng. Computer Society, 2006. 173-180

[35] Plank J S, Ding Y. Note: Correction to the 1997 tutorial on Reed Solomon coding. Softw. Pract. Exper., 2005, 35(2): 189-194

[36] Bloemer J, Kalfane M, Karp R, et al. An XOR-based erasure-resilient coding scheme. Technical Report TR-95-048, International Computer Science Institute, 1995

[37] Luby M. Tornado Codes: Practical Erasure Codes Based on Random Irregular Graphs. in: Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science. London, UK: Springer-Verlag, 1998. 171

[38] Lin W K, Chiu D M, Lee Y B. Erasure code replication revisited. in: Proceedings - Fourth International Conference on Peer-to-Peer Computing(P2P2004). Zurich, Switzerland: IEEE Computer Society, 2004. 90-97

[39] Weatherspoon H, Kubiatowicz J. Erasure Coding Vs. Replication: A Quantitative Comparison. in: Revised Papers from the First International Workshop on Peer-to-Peer Systems. Springer-Verlag, 2002. 328-338

[40] Rodrigues R, Liskov B. High availability in DHTs: erasure coding vs. replication. in: Peer-to-Peer Systems IV. 4th International Workshop, IPTPS 2005. Revised Selected Papers. Berlin, Germany: Springer-Verlag, 2005. 226-239

[41] Khan O, Burns R, Plank J, et al. Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads. in: Proceedings of the 10th USENIX conference on File and Storage Technologies. San Jose, CA: USENIX Association, 2012. 20

[42] Cherkasova L, Ciardo G. Characterizing temporal locality and its impact on web server performance. in: Proceedings Ninth International Conference on Computer Communications and Networks. Piscataway, NJ, USA: IEEE, 2000. 434-441

[43] Cherkasova L, Minaxi G. Analysis of enterprise media server workloads: access patterns, locality, content evolution, and rates of change. IEEE/ACM Transactions on Networking, 2004, 12(5): 781-794

[44] Li-Pin C, Tei-Wei K. An adaptive striping architecture for flash memory storage systems of embedded systems. in: Proceedings Eighth IEEE Real-Time and Embedded Technology and Applications Symposium. Los Alamitos, CA, USA: IEEE Comput. Soc, 2002. 187-196